On the Loss Landscape of Deep Neural Networks

Christian H. X. Ali Mehmeti-Göpel

October 18, 2024

On the Loss Landscape of Deep Neural Networks

Dissertation

submitted for the award of the title

“Doctor of Natural Sciences”

to the Faculty of Physics, Mathematics, and Computer Science
of Johannes Gutenberg University Mainz
in Mainz

Christian H. X. Ali Mehmeti-Göpel

Born in Friedberg

Mainz, October 18, 2024

Christian H. X. Ali Mehmeti-Göpel
On the Loss Landscape of Deep Neural Networks
Dissertation, October 18, 2024
Oral Examination, January 15, 2025
Reviewers: Michael Wand and Katharina von der Wense

Johannes Gutenberg University Mainz
Visual Computing
Institute of Computer Science
FB08
Staudingerweg 9
55128 Mainz

Abstract

As a non-convex optimization problem, the training of deep neural networks remains poorly understood, and its success critically depends on the exact network architecture used. While the amount of new network architectures proposed in the last decade is staggering, only a handful of common patterns emerged that are shared by most successful architectures. First, using the smoothness of the optimization landscape as a heuristic for trainability, we investigate what network components render training difficult and how these patterns help alleviate such difficulties. We find that while giving networks their expressivity, deep stacks of nonlinear layers significantly increase the roughness of the optimization landscape as network depth increases. Developing prior work, we quantify this effect and show that for networks at initialization, the strength of this effect depends on the smoothness of the nonlinear layer used. We then demonstrate how residual connections and multi-path architectures reduce high frequencies in the optimization landscape, resulting in increased trainability. Second, we found that normalization layers combined with an adequate warm-up scheme compensate for the increasing roughness in lower layers by dynamically re-scaling the layer-wise gradients. We prove that in a properly normalized network, all layer-wise effective learning speeds align over time, compensating for even exponentially exploding gradients at initialization. Finally, we conduct an empirical study to determine the necessary nonlinear depth of a network to generalize effectively on common deep learning tasks. Surprisingly, we find that a shallow network extracted after training significantly outperforms a comparably shallow network trained from scratch, although their expressivity is exactly the same. We also observe that ensembles of both shallow and deep paths outperform comparable networks comprised of only deep paths, even when extracted after training. Using these insights, we aim to gain a deeper understanding of how to design deep neural networks with high trainability and strong generalization properties.

Acknowledgements

I would like to thank Giorgia D'Amico, Noah Irion and Marcel Geschke for helping me keeping my sanity in difficult times.

I also would also like to acknowledge Felix Ali Mehmeti, Jan Disselhoff and Marco Santucci for many hours of enthusiastic scientific discussion.

Finally, I would like to express my gratitude to Michael Wand for giving me the opportunity to fully realize my academic potential.

Contents

1 Introduction

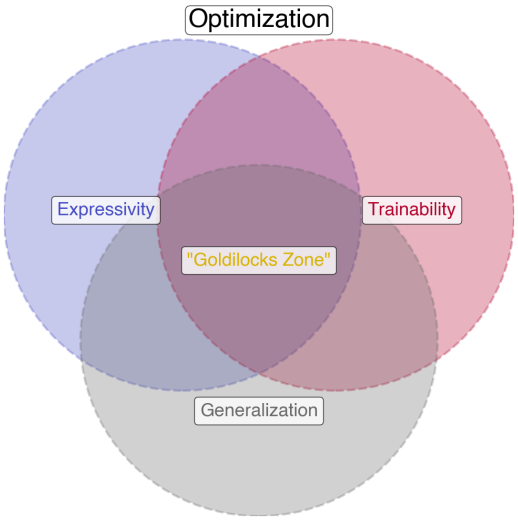


Fig. 1.1.: The three necessary ingredients for successfully training a highly performant neural network intersect at the 'Goldilocks Zone', which represents the perfect trade-off between these components.

In the past decade, combining deep neural networks with big data has enabled dramatic breakthroughs [29]. The inductive bias provided by network architectures and training protocols appears to be better aligned with natural data distributions than any other methods available today. This success is not limited to 2D image classification but extends to various domains, such as representing the shape and texture of complex 3D objects [42], text processing [43], protein folding [26], and multimodal tasks [1]. This fact makes it at least plausible that we have found a universal prior for natural data hidden within the inductive bias behind deep learning.

The problem of achieving insightful predictions on unseen data using deep neural networks is quite complex to grasp and commonly broken down into three components as shown in Figure 1.1: the existence of a solution (*expressivity*), how to find this solution (*trainability*), and whether this solution also works on in-distribution, but unseen data (*generalization*). We aim to understand how to build a neural network architecture in the 'Goldilocks Zone', representing the optimal trade-off between these three components.

Although the biggest mystery seemingly lies in the exceptional generalization capabilities of deep networks [65], its trainability i.e. managing the extremely high-dimensional, non-convex optimization process also remains poorly understood. Establishing global convergence guarantees in deep learning without wildly unreasonable assumptions (e.g. inputs follow a normal distribution [7], two-layer networks [34], etc.) is generally considered a very difficult problem, and training success critically depends on the chosen network architecture. In our analysis, we primarily investigate the impact of architecture design on trainability as well as generalization, while also addressing the question of what constitutes a sufficiently expressive network.

Providing purely analytical guarantees for trainability seems improbable, as for most problems which are usually solved using deep learning, the best mathematical description of the underlying training data distribution we possess usually is the trained network itself. In this work, we therefore resort to the smoothness of the optimization landscape in a spectral sense as a simpler heuristic for trainability as well as generalization performance, which we will motivate in the following. Taking the derivative of a function corresponds to applying a high-pass filter in the Fourier domain, which increases higher frequencies. Therefore, the presence of high frequencies in the magnitude spectrum of the loss surface directly relates to the maximum eigenvalue of the Hessian, which in turn effectively sets an upper limit on the learning rate which can be used in gradient descent to ensure convergence [31]. The smoothness of a minimizer is also closely linked to the robustness of a model's solution: a low variance in the loss function around the minimizer suggests that the model is less likely to overfit the training data, as it indicates stability in the presence of small perturbations in the input space [18].

In this work, we first apply Fourier analysis to the basic components of a neural network to determine their respective impact on the smoothness of the optimization landscape. We will come to understand that while nonlinear layers give deep networks their expressivity, they also cause roughness in the optimization landscape which correlates with low trainability. At the same time, residual connections and multi-path architectures smoothen the optimization landscape and consequently alleviate this problem. Next, after a thorough 'static' study of the spectral properties of different layers at initialization, we show that normalization layers induce a dynamic effect that aligns the layer-wise effective learning speeds over time. This compensates for exploding gradient norms in the lower layers of the network, a consequence of those being especially affected by rough optimization landscapes. As there

seems to be an expressivity-trainability trade-off regarding a network's non-linear depth, i.e. the number of nonlinear layers stacked, in a final step, we empirically investigate the minimum nonlinear depth that a network needs to sufficiently train and generalize on common deep learning problems and obtain surprising results.

In a nutshell, want to build a deeper understanding on the impact of architecture design on network trainability and generalization.

2 Roughness of the Loss Surface at Initialization: Blueshift

2.1 Introduction

Early supervised learning algorithms such as the Perceptron [57] worked as linear classifiers that slightly adjust their decision hyperplane to accommodate for every new example seen; for linearly separable data, this algorithm provably converges. The initial excitement quickly faded as scholars realized that the expressivity of such a linear classifier is not even sufficient to solve simple nonlinear problems such as the XOR-problem [56]. Consequently, researchers had to find a way to learn more complex, nonlinear functions with a similar error-minimizing approach. Although the invention is contested [50], the backpropagation algorithm eventually emerged in the early seventies [35] allowing for the iterative learning of complex nonlinear functions by gradient descent, known today as deep learning. Unlike linear classifiers, there are no known convergence guarantees for training deep neural networks using gradient descent on generic non-convex objectives, which renders the optimization of such networks an active field of research.

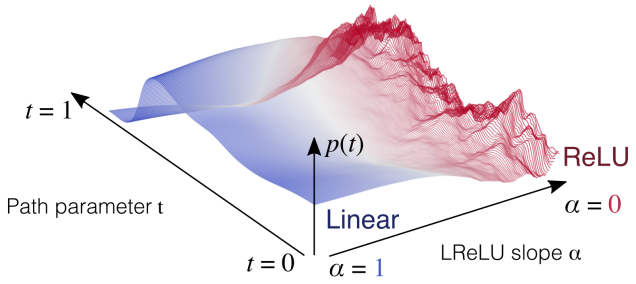


Fig. 2.1.: Continous transition of a w-o path of a ResNet56 near initialization between linear ("Linear", $\alpha = 0$) and nonlinear ("ReLU", $\alpha = 1$) networks realized by shifting the α parameter of all ReLU units in the network. We can observe the path becoming visibly rougher for increasing α.

One possible approach to gain intuitions about the learning process is observing the geometry of the *loss surface*, i.e. the manifold whose lowest point we are searching for during training. This approach has been used by He et al. [33] who show that the loss surface of deep feed-forward networks becomes visibly smoother when adding residual connections, potentially explaining the high trainability of ResNets [15]. This work attempts to quantify their qualitative approach using Fourier analysis. Despite the fact that neural networks optimize thousands or even billions of parameters during training, we can locally sample random subspaces of the loss surface and estimate their roughness. Albeit random low-dimensional cross-sections of an extremely high-dimensional object might not be entirely representative of the original topology, we hope to be able to observe at least phenomena of isotropic nature. Our motivation is to use this method to understand which network components make the loss surface rough, with the hope of finding a good predictor for network trainability.

2.2 Summary

In our work *Ringing ReLUs: Harmonic Distortion Analysis of Nonlinear Feed-forward Networks*, we study the spectral properties of the network function $f(\mathbf{x}, \mathbf{W})$ when varying weights $\mathbf{W}$ for fixed input $\mathbf{x}$, which we call the *weight-output (w-o) surface*. In this section, we extend our analysis from the paper and consider the Fourier transform $P(\omega) \in \mathbb{C}$ of a 1-dimensional *w-o path* $p(t) \in \mathbb{R}$:

$$p(t) = f(\mathbf{x}, \mathbf{W} + \alpha \cdot t \cdot \mathbf{D}) = \int_{-\infty}^{\infty} P(\omega) e^{2\pi i \omega t} d\omega, \tag{2.1}$$

for a random direction $\mathbf{D}$, sample radius α and parametrization $t \in [0, 1]$. Next, we show that applying a generic nonlinear function causes the spectrum to shift towards higher frequencies. Since by the Stone-Weierstrass theorem is possible to approximate any continuous function by a polynomial up to an arbitrary precision, we consider the effect of a polynomial nonlinearity $\phi(x) = \sum_{j=0}^{K} a_j x^j$ on our loss path. Thanks to the convolution theorem, we

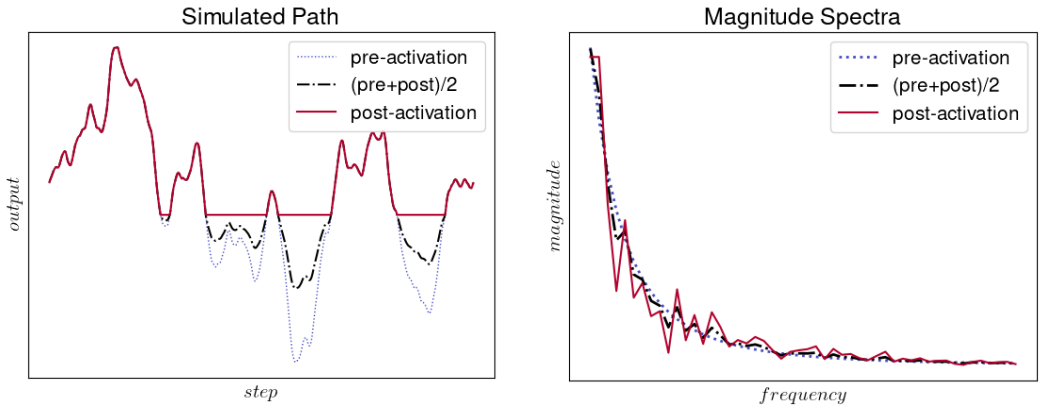


Fig. 2.2.: An illustrative example showing the creation of harmonic distortions (**right**) when applying ReLU to a FBM path (**left**). Similar to ResNets, we obtain a smoother signal by averaging the sum of the original input and the blue-shifted input.

know that taking the j-th power in time domain corresponds to computing the j-th auto-convolution in spectral domain:

$$\phi(p(t)) = \sum_{j=0}^{K} a_j \left[\int_{-\infty}^{\infty} P(\omega) e^{2\pi i \omega t} d\omega \right]^j \tag{2.2}$$

$$= \sum_{j=0}^{K} a_j \int_{-\infty}^{\infty} P^{\circledast j}(\omega) e^{2\pi i \omega t} d\omega, \tag{2.3}$$

where $^{\circledast j}$ represents the j-th auto-convolution. If we additionally assume absolute convergence of the original magnitude spectrum, i.e. $\int_{-\infty}^{\infty} |P(\omega)| d\omega < \infty$, we can use Fubini's theorem to re-write the sum as a Fourier transform $\tilde{P}(\omega)$:

$$\phi(p(t)) = \int_{-\infty}^{\infty} \underbrace{\left(\sum_{j=0}^{K} a_j P^{\circledast j}(\omega) \right)}_{:=\tilde{P}(\omega)} e^{2\pi i \omega t} d\omega. \tag{2.4}$$

The absolute convergence assumption is true for example for band-limited functions and power-law-like functions (ref. Appendix Proposition 1). Since empirically measured magnitude spectra seem to decay similarly to a power-law, we deem this to be a reasonable assumption.

For a band-limited input spectrum $P(\omega)$ with support s, each auto-convolution increases the support by s entries on each side ($2s$ overall); but since $P(\omega)$

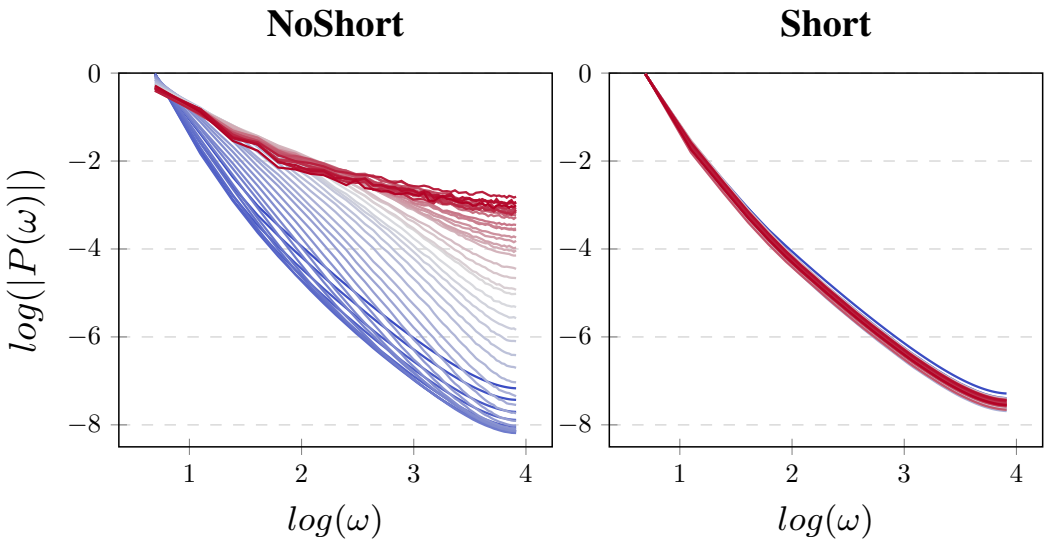


Fig. 2.3.: Normalized magnitude spectra per layer for a w-o path in a 50-layer ReLU network **at initialization**, with ("Short", **left**) and without ("NoShort", **right**) residual connections. Lower layers are shown in red, higher layers in blue.

is complex-valued, this can only be considered an upper bound. In Figure 2.2, we illustrate this principle by applying the ReLU function on a random fractal Brownian motion (FBM) path, which are empirically similar to w-o paths. We can see the resulting *harmonic distortions* as oscillating peaks in the magnitude spectrum on the right-hand side of the figure. Repeated distortions (i.e. stacking layers) shift the spectrum towards higher frequencies and we therefore, inspired by physical engineering, we call this effect *blueshift*. We can clearly observe this effect in real networks at initialization: in Figure 2.3, we can see that if we sample a path by altering the weights from a lower layer in the network, significantly more high frequencies are present in the magnitude spectrum than if we change the weights of a higher layer. In the full paper, a more detailed analysis is given which shows that the strength of the *blueshift effect* depends on smoothness of the nonlinearity used (e.g. Section 7.1 Figure 3).

Next, we investigate the apparent smoothing effects of residual connections that we could observe in Figure 2.3, where networks with residual connections exhibit no visible blueshift at initialization. We distinguish two different mechanisms that allow ResNets to dampen the creation of high frequencies: *exponential down-weighting* (ED) and *frequency-dependent signal-averaging* (FDSA). The former effect is illustrated in Figure 2.2, where we plot the magnitude spectrum of the averaged pre- and post-activation path. The magnitude spectrum of the post-activation path contains more harmonic distortions /

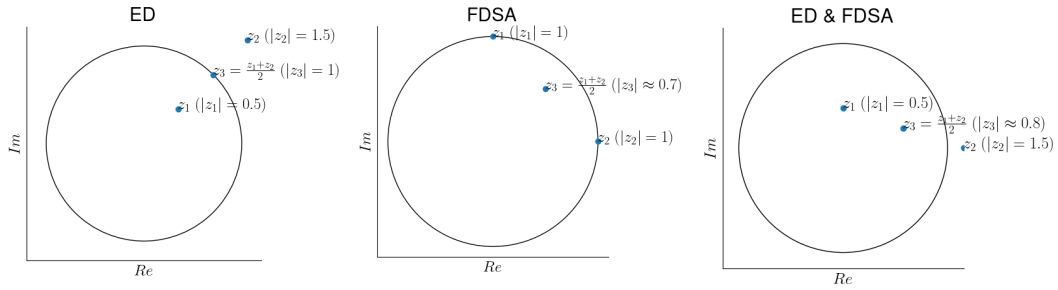


Fig. 2.4.: Illustrating the difference between the ED and FDSA mechanisms in the complex plane showing how both can act at the same time.

high frequencies as a result of applying an additional nonlinear layer, similar to the residual branch in a ResNet block. Averaging both paths in time domain corresponds to averaging a high frequent spectrum with a lower frequent one in Fourier domain, resulting in a path containing less blueshift compared to the post-activation path. This argument is similar in essence to Veit et al. [54], who show that a ResNet can be seen as an ensemble of rather shallow networks.

This effect is further amplified by FDSA, an effect we conjecture to affect all types of multi-path architectures including ResNets. In a multi-path architecture, the input signal is split into different computation paths that generally contain nonlinear layers and thus shift the signal towards higher frequencies because of blueshift. Since weights are not shared between computation branches, we expect a frequency bias, i.e. Fourier coefficients corresponding to higher frequencies have a lower correlation between branches than lower frequencies. We have indeed empirically confirmed such a frequency bias (ref. Section 7.1 Figure 6) and observed a consequent smoothing effect at initialization (ref. Section 7.1 Figure 8). In Figure 2.4, we illustrate the difference between FDSA and ED and show how both can act together in a complementary way.

Leaky ReLU units (ReLU units with variable, constant slope) allow us to continuously interpolate between a ReLU unit and a linear function by gradually shifting its slope α (ref. Figure 2.5) in the interval $[0, 1]$. We utilized this in Figure 2.1 to visualize the effect of using layers with varying nonlinearity: we see that the closer α gets to 1, the smoother the path gets. Then, we quantified this effect by defining the empirical smoothness of the w-o surface as the decay rate of the magnitude spectrum, averaged over many w-o paths in random directions. In Section 7.1 Figure 9, we measured the empirical

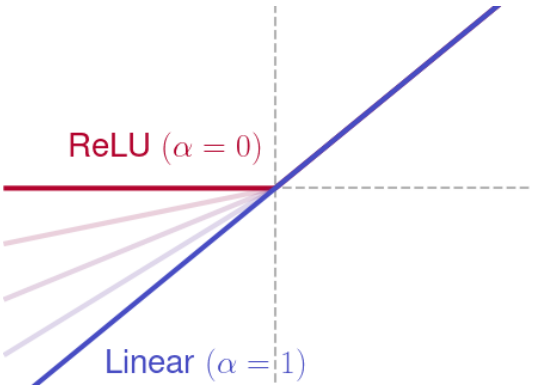


Fig. 2.5.: ReLU with variable slope (Leaky ReLU) continuously interpolating between a ReLU unit and a linear function for $\alpha \in [0, 1]$.

smoothness at initialization for LReLU networks for variable α and depth and compared it to the networks training performances. We found that as expected, for $\alpha \approx 0$, deep feed-forward networks present a very rough loss surface and have indeed difficulties training, which can be compensated to some degree by using less nonlinear layers (higher α). However, merely the networks with very rough loss surfaces had significant problems training, whereas networks with a moderately rough or smooth initial loss surface ($\alpha > 0.3$) trained equally well. We note that for no choice of α, the performance of the deep feedforward network reached the one of residual networks.

2.3 Outlook and Limitations

A limitation of our work is that the exact amount of blueshift (i.e. the resulting magnitude spectrum) depends on the type of nonlinear layer used and is hard to quantify analytically, even at initialization where weights are randomly distributed. In the previous section, we saw that applying a polynomial nonlinearity of degree j in spectral domain involves computing the j-th auto-convolution of the input signal. As to our best knowledge, there are no theoretical statements characterizing the magnitude spectrum of the j-th auto-convolution of a function, even under simplifying assumptions such as that the initial path function obeys a power-law.

Next, we found that the usual pitfalls of sampling theory (aliasing, choice of step size, choice of sampling radius) equally apply when sampling w-o paths, resulting in many difficulties whilst conducting the experiments. Especially during training, where the scale of the phenomenon we are trying to observe

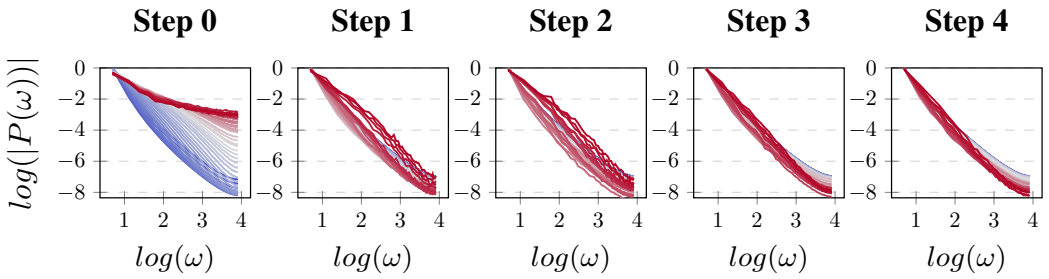


Fig. 2.6.: Normalized magnitude spectra per layer for a w-o path in a 50-layer ReLU network without residual connections **during the first steps of training**. The lower layers are drawn in blue.

might change, it is difficult to say if a sampled loss surface is truly smooth or if perhaps merely our field of observation (e.g. step size, sampling radius) is too small. Additionally, since we are sampling random directions in a high-dimensional setting, anisotropic phenomena [46] could be easily missed.

Finally, in our paper, we limit the analysis of the smoothness of the w-o surface to the initialization point, where the empirical measurements are quite concise i.e. deep stacks of nonlinear functions produce a rough surface and smoother nonlinearities produce smoother surfaces. On the contrary, after running the same experiment during training (ref. Figure 2.6), we see that to our great surprise, measured w-o paths seem to smoothen themselves after only a few steps.

3 Implicit Dynamic Re-Scaling of Loss Surface: Autotune

3.1 Introduction

In Chapter 2, we saw that at initialization, the roughness of the loss surface can be traced back to deep stacks of nonlinear layers. But only after a few steps of training (ref. Figure 2.6), the measured roughness seems to disappear! In this chapter, we investigate this surprising phenomenon.

Initially, we found it hard to identify the primary cause of this phenomenon, as we found that more than one factor is contributing to this apparent dynamic smoothing of the loss surface: the loss function itself saturates as a result of training [32], additive affine BatchNorm parameters push the data distribution towards the linear region of ReLU activations [41], and finally momentum [55] and weight decay [19] additionally influence weight dynamics. We ran experiments in a very basic setting, where all of the factors listed before can be disregarded, and still found that a strong smoothing of the loss surface takes place within the first steps.

Li et al. [33] argue that when visualizing random slices of the loss landscape similarly to Chapter 2, one should scale the random sampling direction $\mathbf{D}$ by the Frobenius norm $\|\mathbf{W}\|_2$ of the corresponding weight (channelwise) in order to compensate for the scale-invariance introduced by BatchNorm. Hoffer et al. [19] later show that the update size of the weight direction $\widehat{\mathbf{W}} := \frac{\mathbf{W}}{\|\mathbf{W}\|_2}$ is approximately proportional to the squared inverse weight norm, i.e. $\widehat{W}_{t_i} - \widehat{W}_{t_{i+1}} \sim \frac{1}{\|\mathbf{W}\|_2^2}$. Thereafter, Yang et al. [59] make the following informal observation:

> ‘However, we observe that two related phenomena occur after a single step of learning: the weights grow exponentially in the depth and the magnitude of the gradients are stable up to some threshold after which they vanish exponentially in the depth. [...] The first-step gradients dominate the weights due to gradient explosion, hence the exponential growth in weight norms, and thereafter, the

gradients are scaled down commensurately. Thus, it seems that although the gradients of batch normalized networks at initialization are ill-conditioned, the gradients appear to quickly reach a stable dynamical equilibrium. ’

Other works have touched this subject [55], but fail to capture that this effect is purely induced by the weight dynamics of deep normalized networks, and strictly depend on additional factors such as weight decay to prove the convergence to a dynamical equilibrium.

Gradient norm is closely related to smoothness in a spectral sense, as high-frequent functions possess higher gradient norms on average [40]. Measuring layer-wise gradient norms during training provides the benefits of being computationally cheap and avoiding the problems related to sampling discussed in Section 2.3. The *blueshift* phenomenon (ref. Figure 2.2) we observed in Fourier domain then simply corresponds to the well-known exploding gradients effect in deep normalized networks [59]. As globally exploding gradients (i.e. at a constant rate across all layers) can simply be balanced by adjusting the learning rate, we are only interested in how much gradient norms differ across the layers, which we call *spread*.

3.2 Summary

In our work *On the Weight Dynamics of Deep Normalized Networks*, we model this phenomenon from first principles. We start from two basic observations: first, the gradient of any layer N that is invariant to scaling in the forward pass $N(\gamma \cdot x) = N(x)$ (e.g. all normalization layers) scales inversely with its input [58]:

$$\frac{dN}{d\gamma x}(\gamma x) = \frac{1}{\gamma}\frac{dN}{dx}(x), \quad \gamma \in \mathbb{R}. \tag{3.1}$$

Secondly, since normalization layers are scale invariant, no gradient can flow towards this direction. Hence, weight updates $\nabla \mathbf{W}$ are orthogonal to the weights $\mathbf{W}$ themselves [2]:

$$\langle \nabla \mathbf{W}, \mathbf{W} \rangle = 0. \tag{3.2}$$

Assuming that the "base gradient" of a layer, meaning the gradient magnitude excluding normalization induced scaling effects, is constant during training i.e.:

$$c := \mathbb{E}(\|\mathbf{W}(t_i)\|_2 \cdot \|\nabla \mathbf{W}(t_i)\|_2), \tag{3.3}$$

we derive the following update rule for the expected layer-wise weight norm $\sigma^2(t_i) := \mathbb{E}(\|\mathbf{W}(t_i)\|_2^2)$ for vanilla SGD with a scheduled learning rate $\lambda(t_i) > 0$ that we call our *discrete model*:

$$\sigma^2(t_{i+1}) = \sigma^2(t_i) + \frac{\lambda(t_i)^2 c^2}{\sigma^2(t_i)}. \tag{3.4}$$

To account for scale invariance, we are interested in the update size of the weight direction $\widehat{\mathbf{W}} := \frac{\mathbf{W}}{\|\mathbf{W}\|_2}$. Similarly to van Laarhoven [30], by approximating $\|\mathbf{W}(t_{i+1})\|_2 \approx \|\mathbf{W}(t_i)\|_2$, we can write:

$$\widehat{\mathbf{W}}(t_{i+1}) - \widehat{\mathbf{W}}(t_i) = \frac{\mathbf{W}(t_{i+1})}{\|\mathbf{W}(t_{i+1})\|_2} - \frac{\mathbf{W}(t_i)}{\|\mathbf{W}(t_i)\|_2} \approx \frac{\mathbf{W}(t_{i+1}) - \mathbf{W}(t_i)}{\|\mathbf{W}(t_i)\|_2} = \lambda(t_i) \cdot \frac{\nabla \mathbf{W}(t_i)}{\|\mathbf{W}(t_i)\|_2}. \tag{3.5}$$

We therefore have to monitor the ratio of gradient norm to weight norm, instead of plain gradient norms as a measure of update size in normalized networks. We therefore define the *effective learning rate* (ELR) as:

$$E(t_i) := \mathbb{E}\left(\frac{||\nabla \mathbf{W}(t_i)||_F}{||\mathbf{W}(t_i)||_F}\right) = \frac{c}{\sigma^2(t_i)}. \tag{3.6}$$

We are especially interested in the evolution over time of ratios for different initial parameters c:

$$R_{jk}(t_i) := \frac{E_j}{E_k}(t_i) = \frac{c_j \sigma_k^2}{c_k \sigma_j^2}(t_i), \tag{3.7}$$

as the initial gradient norm of feedforward networks explodes exponentially in depth [60]. In summary, we obtain the following theoretical findings:

- In the gradient flow ($\lambda \to 0$) and for discrete, constant learning rates ($\lambda >> 0$), the ELR ratio of any two layers j and k converges in the time limit, regardless of their initial values, i.e. $\lim_{i \to \infty} R_{jk}(t_i) = 1$.

- For discrete, scheduled learning rates $\lambda(t_i)$, the ELR ratio of any two layers j and k nears its limit during each step, i.e. $|R_{jk}(t_{i+1}) - 1| < |R_{jk}(t_i) - 1|$, given that the current learning rate $\lambda(t_i)$ does not exceed the *flipping ratio* $\kappa_{jk}(t_i) := \sqrt{\frac{1}{E_j E_k}}(t_i)$.

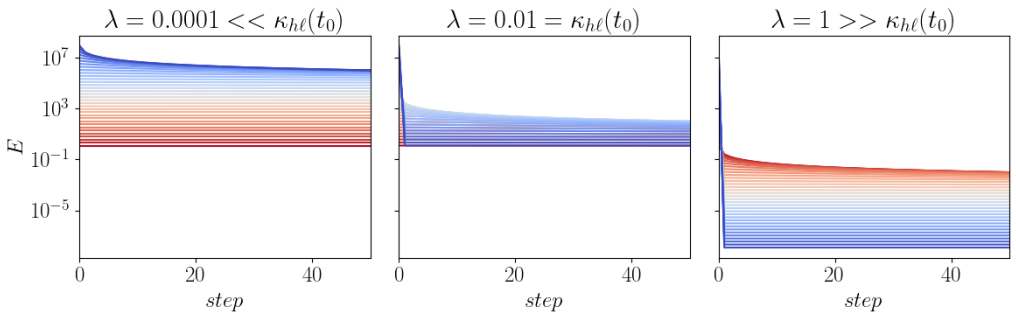


Fig. 3.1.: Simulated evolution of layer-wise effective learning rates with our discrete model, assuming initially exponentially exploding gradients. We use a constant learning rate which is sub-critical (**left**), critical (**middle**) or super-critical (**right**) in the first step. Blue color corresponds to the lower layers with higher initial gradients.

- If we chose $\lambda(t_i) = \kappa_{jk}(t_i)$ for any two layers j and k, the ELR of those two layers is equal in the next step and remains equal in any consequent timestep, i.e. $E_j(t_{i'}) = E_k(t_{i'})$ for all $i' \geq i$.

We visualize these results in Figure 3.1, where we simulated the evolution of layer-wise effective learning rates using our discrete model of Eq. 3.4 for different constant λ and assuming exponentially exploding initial gradients. On the left-hand side, we see that as predicted by our results above, the ELR spread diminishes over time for any learning rate small enough. On the right-hand side, the learning rate exceeds the *flipping ratio* of the layer ℓ with the lowest and h highest ELR in the zeroth step which we call the *critical learning rate*. We see that consequently, the ELRs of those two layers flip in the next step (i.e. $E_h(t_1) << E_\ell(t_1)$), temporarily increasing ELR spread and slowing down convergence. In the middle figure, we see that when choosing the learning rate as $\lambda = \kappa_{\ell h}(t_0)$, we obtain $E_h(t_1) = E_\ell(t_1)$ in the next and all consequent steps, resulting in a fast reduction of ELR spread.

By applying this strategy at each step instead of just the first one, we obtain a fast hyperparameter-free warm-up scheduler that massively speeds up convergence. In Figure 3.2, we simulated two different linear warm-up schemes with different hyperparameters: we see that depending on the exact values chosen, linear warm-up can be either too slow or too fast, resulting in slow convergence, especially if the critical learning rate is exceeded at a certain point in training. Critical warm-up on the other hand does not require correctly tuned hyperparameters to quickly reduce ELR spread. However, in an empirical evaluation on real networks in training (ref. Section 7.3 Figure 3),

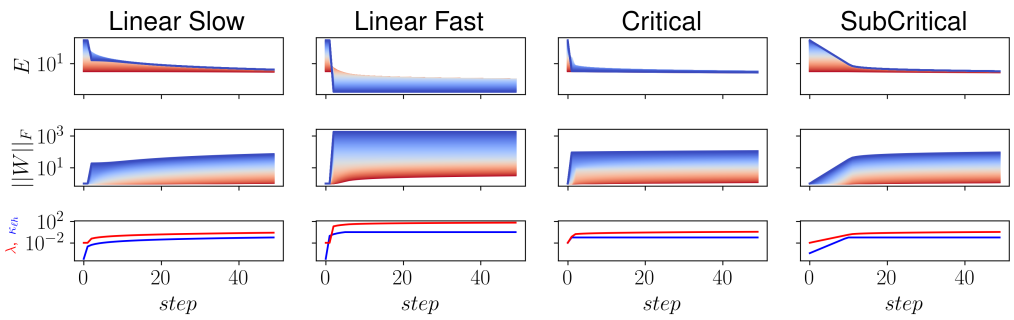


Fig. 3.2.: Simulated evolution of layer-wise effective learning rates **(top)**, weight norms **(middle)**, learning rates $\lambda(t)$ and critical LRs $\kappa_{\ell h}(t)$ **(bottom)** for different warm-up schedules. All y-axes are in logarithmic scale. Blue color corresponds to the lower layers.

we found that the predicted *critical learning rate* $\kappa_{\ell h}(t_i)$ is lower than the real value due to other unmodeled effects, such as gradients saturating due to the network learning. Luckily, we found that setting the learning rate as the sub-critical learning rate $\kappa_{hh'}(t_i)$ at each step, which represents the flipping ratio of the two layers with the highest learning rate h and h', represents a safe choice which is still guaranteed to converge in num_layers steps and seems to work in practice as it represents a much more conservative choice, i.e. $\kappa_{hh'}(t_i) << \kappa_{\ell h}(t_i)$.

Another possibility of controlling ELR spread is simply *constraining* the norm of each layer's gradient before each step, such that layer-wise effective learning rates are constant by construction:

$$\nabla W \leftarrow \nabla W \cdot \frac{E_{goal}}{E + \epsilon}, \tag{3.8}$$

for a given constant goal effective learning rate E_{goal} and a small $\epsilon > 0$. However, doing so disregards the true gradient scale of the network function.

Finally, in the experimental section, we demonstrate the catastrophic effects of high ELR spreads on the trainability of networks. We present a case study of a 110-layer feedforward convolutional network, which exhibits an extremely high ELR spread and fails to train at any constant learning rate. Consistent with our theoretical predictions, the application of *constraint ELR* or *sub-critical warm-up* effectively reduces ELR spreads and correlates with restored network trainability, but a performance gap with regard to residual architectures remains.

3.3 Outlook

Since our theoretical result holds for any neural network where scale invariance in the layer weights is given, it is applicable to most network architectures, as normalization layers are used ubiquitously. A notable exception is Transformer models [53]: since attention blocks usually do not contain normalization layers, their query and key matrices are not scale invariant. Adding auxiliary normalization layers to enforce scale invariance is usually not considered problematic, but in the case of Transformer models, it would fundamentally alter the architecture design. As the weight norm of scale-invariant layers always strictly grows (at least without weight decay), adding normalization layers to attention blocks implies that the network would not be able to 'un-learn attention' anymore, a potentially defining trait of Transformer architectures. Further research is needed to understand whether it is possible to build a scale-invariant Transformer-like architecture with comparable properties and performance.

Mechanics similar to our *ELR constrain* method find application in practice, especially in large-batch training of convolutional networks [62] and Transformer models [63]; this is because practitioners use large learning rates in order to compensate for fewer steps in large-batch training [28] and weight norms scale quadratically in λ as we have seen in Eq. 3.4. The question arises whether simply prescribing the desired ELR and disregarding the true gradient scale is already the best solution to avoid high ELR spreads, or if using other methods that preserve the true gradient scale such as warm-up could be superior in certain regards.

One such scenario would be the use of momentum-like techniques since it is common practice to use warm-up in order to slowly populate the momentum buffer. Adaptive optimizers such as AdamW [37] currently enjoy great popularity, especially in the context of training large language models. We found that it is non-trivial to extend our theory to even just simulate the evolution of the layer-wise weight dynamics for training with momentum methods, as the use of a momentum buffer breaks the orthogonality condition (ref. Eq. 3.2) which our model is based on. Using additional orthogonality assumptions, which are at least plausible in high-dimensional vector spaces, we succeeded in extending our discrete model (ref. Eq. 3.4) to training with momentum SGD in Appendix Section A.2.1. However, we did not yet extend the convergence results to this scenario, as significant additional bookkeeping is needed. In a

simulation, where we used the gradient norms corresponding to a deep feed-forward network at initialization as a starting point (ref. Appendix Section A.2.2), we found that momentum SGD with commonly used hyper-parameters leads to a higher reduction in ELR spread wrt. standard SGD. This qualitatively matches with the spreads measured on real networks and also correlates with increased trainability. In future work, we envision extending our convergence analysis to momentum SGD and both our model and the convergence analysis to adaptative optimizers such as Adam or AdamW.

Finally, we discussed that in a real training scenario, the critical ELR we predict overshoots the real value, as opposed to a random walk scenario where our predictions are accurate. This is due to the fact that we assume that the layer-wise gradient norms excluding normalization-induced re-scaling are constant; this is not true in reality since they vary during training, as empirically they get smaller as the network learns. We envision extending our theory to include such currently unmodeled effects.

4 Impact of Nonlinearity on Optimization: Nonlinear Advantage

4.1 Introduction

In Chapter 2, we came to understand that deeper networks are significantly harder to train. However it is also their depth that gives networks the expressivity needed to solve complex problems [8]. In this Section, we present the results of an empirical study with the goal to determine how much depth is actually needed to solve common deep learning problems.

The *universal approximation theorem* (UAT) [9] is a classical result, which proves that any function can be approximated by a sufficiently large neural network with a single hidden layer. Albeit technically true, this statement is not representative of how deep networks learn in practice, as in the constructive proof of the theorem the function is merely approximated (or memorized) piece-wise, akin to the Riemann integral. In practice, it is not possible for a network to memorize the entirety of all training data, as usually the amount of data fed to the network is magnitudes greater than the potential storage capacity in the network weights.

Surprisingly, even in a setting where a network has enough capacity to memorize the entire dataset, it does still present emerging generalization capabilities, also without explicit regularization [65]. A more recent variation of the UAT [27] proves a similar statement, but for a deep narrow network with finite depth: this seems like a more realistic setting, where expressivity is obtained through hierarchic abstraction (i.e. network depth) instead of pure memorization. From a theoretical perspective, a network's depth (along with its width) upper bounds the complexity of the function that it can represent [5] and therefore upper bounds the network's expressivity. In this context, several intriguing questions arise: how many nonlinear layers or units are

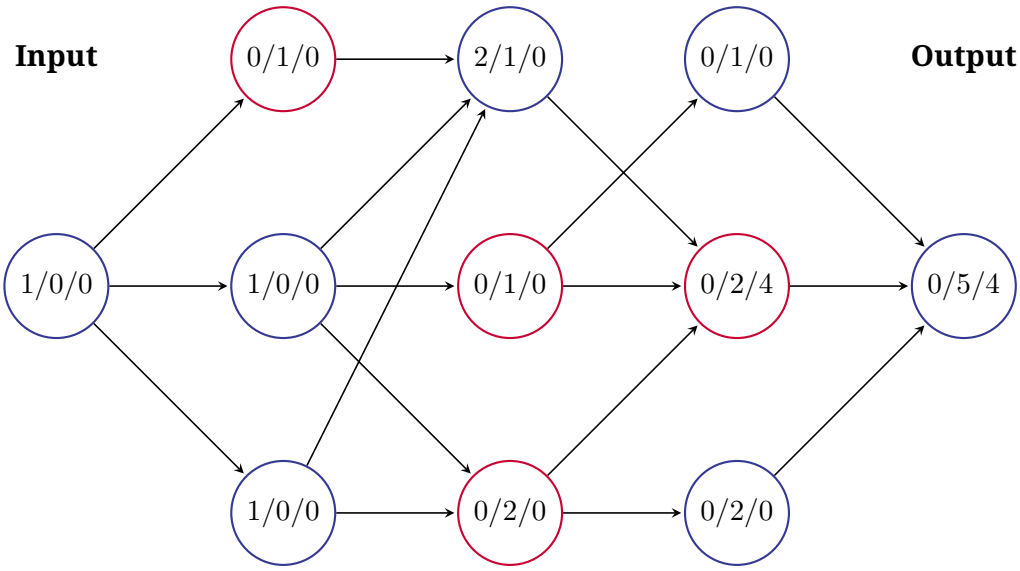


Fig. 4.1.: Example of a computing the *average path length* of a network through dynamic programming.

actually needed to learn a given problem, and what practical measure can be sensibly used to characterize the expressivity of a ReLU network?

4.2 Summary

In our work *Nonlinear Advantage: Trained Networks Might Not Be As Complex as You Think*, we aim to gain insight into these questions by studying the properties of *partly linearized networks*. Partial linearization allows us to a-posteriori shrink the 'depth' or nonlinear complexity of a trained network. We do this in two phases. First, we train the networks using a regular training protocol for a given number of epochs. Then, we try to bring the network function as close as we can to a linear function (both in inputs and weights) without suffering major performance drops. For this, we leverage once again ReLU units with variable slopes (ref. Figure 2.5), by making its slope a learnable parameter (this is called a PReLU unit [16]) and using an additional regularization term $\sum |\alpha_i - 1|^{0.5}$ that pushes the slopes α_i of all PReLU units in the network towards linearity (i.e. $\alpha_i = 1$).

Since we use PReLU units with a learnable slope parameter for each channel (as opposed to e.g. the layerwise approach of Dror et al. [12]), we need a new measure to quantify the "depth" of the resulting partially linearized

network. As a solution, we propose the *average path length* (APL): it represents the average amount of active ($\alpha_i < 1$) PReLU units in a path through the computation graph of the network (from input to output). This is reminiscent of the path-view introduced by Veit et al. [54], but still captures the nature of partly linearized networks.

The *average path length* of a network can be computed efficiently through dynamic programming in a modified forward-pass through the network, as illustrated in Figure 4.1. Each node corresponds to a channel of a PReLU unit: red circles represent channels with *active* units and blue circles represent *inactive* units. The histogram for each unit (i.e. the number of paths containing i active units from input to output) is then computed from left to right. The *average path length* of a network is then simply the weighted sum (number of active units times the number of paths) of the unit corresponding to the output.

In a broad series of experiments, using various architectures (feedforward, ResNet, Transformer), datasets (computer vision and translation tasks), and metrics (APL, ratio of active units), we came to the following conclusions:

- Networks can be linearized to a surprisingly high degree without suffering major performance drops, especially on easy problems. For example, we obtained over 90% test accuracy on Cifar10 for a linearized ResNet56 with a NAPL under 3 and over 61% test accuracy on ImageNet for a pruned ResNet50 with a NAPL under 3.5.

- When linearizing networks of different shape (i.e. depth and width), which were trained on the same problem, just up to the point where major performance drops begin, the resulting networks are approximately constant in effective width and depth. This suggests that our method finds a 'minimum nonlinear core structure' needed to solve a given problem.

- The performance of a networks linearized after training is consistently better than the performance of networks with the exact same nonlinear units trained from scratch. This is highly surprising, as their expressivity is exactly the same. We call this effect the *nonlinear advantage*.

4.3 Outlook and Limitations

A limitation of this work is that the conclusions we draw from our empirical observations are quite general, and finding useful applications is challenging. A possible application for partially linearized networks is *layer folding*, as described by Dror et al. [12], where layer-wise (instead of channel-wise, as in our work) PReLU activations are linearized in a similar way. We compared both techniques in Appendix Section A.4, and found that channel-wise linearization leads to lower NAPL values for a comparable performance. However, when a nonlinear layer is made fully linear, a stack $L_1(\phi(L_2(x)))$ of two linear layers $L_1 \in \mathbb{R}^{\ell \times m}$ and $L_2 \in \mathbb{R}^{m \times n}$ and an inactive nonlinearity ϕ can be simplified to $(L_1 \cdot L_2)(x)$. Whether this results in fewer model parameters depends on the number of hidden neurons ℓ, m, and n. This is not the case for most common architectures, except for architectures trimmed on execution speed such as MobileNet [20]. The authors state show that even if such a *layer folding* operation does not result in fewer model parameters, it can still result in a sped-up forward pass due to how operations are computed on GPUs. Our finding that networks linearized post-training outperform shallow networks trained from scratch with equivalent expressivity underscores the significance of the *layer folding* approach.

Another possible application of our work is private inference (inference on cryptographically secure, sensible data), where the amount of ReLU units is a bottleneck for inference times. Cho et al. [6] have shown that by using a linearization protocol similar to ours, the online latency of private inference methods can be sped up significantly.

In our paper, we mainly observed networks in the high-linearization regime (50-100% of all ReLU units disabled), but we think that studying the properties of networks with a low degree of linearization (0-30% of all units) is equally interesting. We conjecture that a low partial linearization could increase the smoothness of the local minimum found whilst conserving most of the network's expressivity, resulting in an increased generalization performance. This conjecture is in agreement with current research, which links the 'grokking' effect (i.e. the generalization performance increases long after the training loss stagnates) to a lower number of linear regions in input space [24]. We briefly explored this idea [44] and obtained positive results, albeit not very strong ones.

5 Discussion

5.1 On Necessary Conditions for Trainability

In training deep neural networks, certain basic numerical constraints must be met to optimize a network effectively. In this section, we review commonly used stability criteria from prior research, examine past approaches addressing these issues, and explain how our work fits into this broader context. Lubana et al. [38] (as for the first three items) and Balduzzi et al. [3] (as for the last one) proposed the following necessary conditions for model trainability:

1. **Stable Forward Propagation:** activation scale not exploding (i.e. exponentially growing) across layers during the forward pass. This is similar to the concept of covariate shift (e.g. [47] Figure 2), where the mean and variance of activations shift across layers. Activations exploding can cause numerical overflow problems and activations vanishing causes collapsing rank (see below).

2. **Informative Forward Propagation:** no rank collapse (i.e. most singular values being near 0) of the output layer's activations during the forward pass. A rank collapse occurs when the activations of the last layer become linearly dependent, reducing the effective dimensionality of the output. When this happens, the network loses the ability to distinguish between different inputs and cannot learn effectively. This effect occurs for the product of many Gaussian matrices (ref. [48] Figure 6).

3. **Stable Backward Propagation:** gradient scale not exploding (i.e. exponentially growing) across layers during the backward pass. This corresponds to the classical vanishing/exploding gradient problem [51]. We have shown in Chapter 3 that high spread, i.e. some layers learn several orders of magnitude faster than others, can be detrimental to trainability.

4. **Non-Shattering Gradients:** gradients with respect to network inputs should maintain some degree of spatial auto-correlation, meaning they

should resemble brown noise (which has a level of smoothness) rather than white noise (which lacks spatial structure) (ref. [3], Fig. 1). This continuity in the gradients is essential for step-based optimization techniques like SGD, particularly when using momentum-accelerated methods. The shattering of gradients in deep feedforward networks (see Chapter 2) is conjectured to contribute to their low trainability.

The real difficulty lies in solving all of these problems at the same time: a solution to a single problem can worsen the remaining ones, as we will see in the following.

Glorot and Bengio [13] were the first to approach this topic analytically by trying to derive an initialization scheme that guarantees a stable forward and backward propagation. Assuming that the effect of the nonlinear layers in the network is negligible (i.e. using a symmetrical nonlinearity which is approximately linear near the origin), the authors find that for layers of different width, an initialization scheme can either satisfy condition 1 or 3 at the same time. As a compromise, they derive an initialization scheme (commonly referred to as 'Glorot' or 'Xavier'), that approximately satisfies conditions 1 and 3 'well enough' for shallow architectures, but breaks down for deeper networks. He et al. later extended this idea to account for the effect of the nonlinear layer used [17] on activations and gradients, which is especially important for non-symmetric nonlinearities such as ReLU units, resulting in the well-known and currently widely used 'He initialization'.

There is a later line of research called *dynamic isometry*, which aims to initialize the network in a way that brings all singular values of the input-output Jacobian of the network function close to 1, consequently solving the first three stability issues. For linear networks at initialization, it is possible to achieve dynamic isometry for networks with arbitrary depth using orthogonal matrices [49]. Conversely, the authors also show that is not possible for Gaussian matrices of any scale. For nonlinear networks with sigmoid activations, a similar result can be found [45], but the authors leave the following comment concerning their proof strategy:

> 'In essence, this strategy increases the fraction of neurons operating in the linear regime, enabling orthogonal hard-tanh nets to mimic the successful dynamical isometry achieved by orthogonal linear nets. However, this strategy is unavailable for orthogonal ReLU networks.'

This strategy however breaks down as soon as networks start to operate in a nonlinear regime, which in terms of sufficient network expressivity, is exactly the regime we are interested in. In summary, well-thought initialization schemes allow practitioners to train deeper networks than before, but there is no possible initialization scheme that guarantees dynamic isometry for networks of arbitrary depth, shape, and nonlinearity. Additionally, the stability initially ensured by initialization schemes can deteriorate over time as the weight matrices evolve during training.

Normalization layers like Batch Normalization [25] ensure a stable forward pass not only at initialization but also throughout training by dynamically adapting in a data-dependent manner. Using random matrix theory, Daneshmand et al. [10] later show that for linear networks at initialization, BatchNorm also provably prevents a rank collapse during the forward pass, retaining at least the square root of the full rank. Unfortunately, the combination of nonlinear layers and normalization layers also causes exploding gradients in the backward pass at initialization [60, 39]. The further addition of residual connections, given a proper scaling of the residual branch, ensures a stable [11] and informative [10] forward pass, as well as a stable backward pass [64] without shattering gradients [4], which the respective authors have proven for networks at initialization.

A limiting factor of the considerations above is that the respective proofs only hold at initialization. In the following, we will put some thought into whether these considerations also hold during training. Daneshmand et al. [10] present empirical results on nonlinear networks during training, showing that the rank-preserving property of BatchNorm does not break down over the course of training. We ran similar experiments (ref. Appendix Section A.3), but obtained different results for very deep feedforward networks: we measured a rank collapse when training a ResNet110 NoShort without warm-up shortly after initialization, followed by a recovery to full rank within 50-100 epochs of training. We found that this was still the case when properly warming up the network and consequently eliminating ELR spread, confirming that those problems can occur separately. Conversely, except for very high learning rates, we found the output layer of a ResNet110 Short almost always has full rank. This suggests that residual connections could play a significant role in preserving the output rank of deep networks during training.

Our considerations from Chapter 2 combine the unraveled path view of a ResNet from Veit et al. [54] with a spectral view, where longer paths (i.e. paths

passing through more nonlinear layers) create more high frequencies in the w-o surface than shorter paths. Equation 2.3 tells us that when applying a nonlinear layer to a one-dimensional path in a random direction, the resulting spectrum is a linear combination of higher-order auto-convolutions of the original spectrum. This argumentation applies equally in input space and in weight space. Adding the fact that the derivative is simply a high-pass filter in Fourier space, we can easily connect high frequencies in w-o surface (i.e. blueshift) to the concept of shattering gradients. In essence, our conclusions are similar to the work by Balduzzi et al. [4], but our spectral view emphasizes the role of nonlinear layers in gradient shattering and shows that the smoother the shape of the nonlinearity, the less shattering occurs. Moreover, our argumentation makes it clear that the dampening of exploding/shattering gradients due to residual connections also prevails during training. We conclude that residual connections help reduce the exploding/shattering gradients problem in a static fashion, even during training.

In Chapter 3, we came to understand that in a properly normalized network, the ratio of effective learning rates between two layers always eventually converges to 1, no matter their base gradients or even initial weight norms. Theorem 3.6 in Section 7.2 tells us that this system is stateless, so we can easily extend this result to a scenario where the layer-wise base gradients change a finite number of times during training, as long as they stabilize after a certain point in time. In a perhaps more realistic scenario where we cannot assume that gradient norms stop changing (e.g. gradients slowly flatten over time), Proposition 3.7 tells us that the auto-rate tuning effect of normalization layers still brings the layer-wise effective learning rates closer together during every optimization step, given that the learning rate is not too high. The only other problem that can arise in a limited-time scenario is the effective learning rate of a layer (i.e. the ratio of the base gradient to weight norm) becomes too low and equalization is slower than the time-frame set by training. We can conclude that given a correct learning rate scheduling, even if the layer-wise base gradients change over time as a result of training, batch normalization will help dynamically to bring them closer together over time.

In conclusion, evidence suggests that for deep residual networks using batch normalization and properly scaled residual branches, all stability criteria listed above are satisfied at initialization, as well as during training. Perhaps lesser-known insights include that batch normalization dynamically reduces the layer-wise effective learning rate spread during training, while residual connections seem to play a crucial role in preventing rank collapse.

5.2 A Position on Dynamic Network Compression

In Chapter 4, we found that it is possible to reduce the number of nonlinear units in a network to a much higher degree after training compared to before, whilst maintaining a similar performance. We called this effect *nonlinear advantage*. We carefully designed different experimental settings to ensure that this performance gap could not be attributed to differences in training time, placement, or amount of nonlinear units used. The fact that the gap persisted is quite surprising, as the expressivity of the networks is strictly the same in both settings, yet the shallow network trained from scratch is unable to reach the same performance as the network linearized after training.

We were able to confirm this effect across very diverse architectures and datasets ranging from image classification using convnets to translation tasks using transformer architectures. Furthermore, our results are complementary to the the well-known *lottery ticket hypothesis* by Frankle and Carbin, where weights are pruned iteratively by their magnitude as well as other results from the 'dynamic pruning' community [14, 23]. In recent work by Schotthöfer et al. [52], the authors develop dynamically parameterized weight matrices to find low-rank networks during training. In experiments, the authors find that the the fastest rank drop-off happens during early training, in line with our results. Yet, dynamic pruning does not seem to be a popular practice for training large models.

Given the current high popularity of quantization and pruning methods for large language models such as Llama 3 [36], we argue the position that researchers training big models should consider using dynamic pruning approaches such as dynamic low-rank training or layer-folding [12], where the model size is greatly reduced after a short fraction of total training time and resources can be re-allocated. Our results also hint that the immense size of such foundation models might only be needed during very early training phases.

5.3 On Residual Connections

Residual connections are a peculiar subject of study, as the original paper that introduced them [15] cites faster training times and improved generalization capabilities as benefits but does not provide an explanation for why this is

the case. The closest thing to a motivation that can be found in the paper is the following:

> 'When deeper networks are able to start converging, a degradation problem has been exposed: with the network depth increasing, accuracy gets saturated (which might be unsurprising) and then degrades rapidly. [...] The degradation (of training accuracy) indicates that not all systems are similarly easy to optimize. Let us consider a shallower architecture and its deeper counterpart that adds more layers onto it. There exists a solution by construction to the deeper model: the added layers are identity mapping, and the other layers are copied from the learned shallower model.'

While this statement is true for fully linear networks, it does not generally hold for networks containing nonlinear layers. Commonly, linear and nonlinear layers are stacked in an alternating fashion; therefore even if some of its weight matrices would be set to the identity matrix, nonlinear layers in the network would not be skipped and the resulting network function would not correspond to a shallower network. For some particular nonlinearities which have regions that locally approximate the identity function, i.e. ReLU for positive values or TanH near the origin, it might be possible to think of a construction that bypasses the nonlinear layer, but those usually only work under some specific assumption on the input data or lose a significant amount of parameters. As per Chapter 2, once a nonlinear function is applied to the pre-activations of a given layer, blueshift is caused which generally cannot be reversed by later layers. In an experiment comparing the performance of feedforward and residual architectures, the authors further claim that that these optimization difficulties are unlikely to be caused by vanishing gradients (ref. bottom of page 5), as they use networks of moderate depth (18 and 34 layers) with batch normalization. Two years later, Balduzzi et al. published their work entitled 'The Shattered Gradients Problem: If resnets are the answer, then what is the question?' [4], naming shattering gradients as the cause of the performance deterioration of deeper feedforward networks.

In this work, we believe to have gained a more detailed perspective on this matter. First, we were able to confirm and measure the results of Balduzzi et al. by quantifying blueshift (which is in essence a similar concept to shattering gradients) as the decay factor of high frequencies in the w-o surface. The experiment of Section 7.1 Figure 9 shows that indeed a strong blueshift occurs in deep feedforward networks (above approx. 50 layers) which correlates

with reduced trainability. We see that blueshift can be reduced by using a 'less nonlinear' layer (higher α) or by employing residual connections, which consequently restores trainability, consistent with our theory of Chapter 2. Contrary to the claims of He et al. mentioned above, the experiment of Section 7.3 Figure 6 clearly shows that for very deep feedforward networks, gradients are exploding in depth up to a point where networks become un-trainable. This is unsurprising, as exploding gradients are a consequence of blueshift: in previous work [40], we demonstrated that high-frequency one-dimensional paths have high absolute gradients on average. Moreover, we showed the norm of the total derivative in a specific point can be approximated by integrating over such one-dimensional paths, which represent directional derivatives. Exploding gradients thus can be seen as a symptom of blueshift.

However, when having a closer look at the experimental results, shattering and exploding gradients alone do not seem to explain the full extent of residual connection's benefits. In the experiment of Section 7.1 Figure 9, the performance of fully linear networks ($\alpha = 1$), which do not suffer from blueshift, also seems to degrade with increasing depth. This might be a result of the singular value spectrum of the Jacobian concentrating, which has been shown to be the case for deep stacks of Gaussian matrices [49]. Interestingly, we do not observe such a degradation for deep linear networks with residual connections. Another observation supporting this conjecture is that for deeper architectures, we always observe a performance gap between feedforward and residual networks, regardless of the specific 'degree' of nonlinearity used. If merely blueshift was the problem, we would have expected to find a value α representing the optimal trade-off between expressivity and trainability, for which the performance gap disappears. However, looking at the measured fractal coefficients h, which indicates the smoothness of the w-o surface, no value of α produces smooth w-o surfaces (high h value) for networks above 50 layers, not even fully linear networks.

Next, we show that even in a setting without exploding gradients, residual connections offer performance benefits. In Section 3, we discuss how SubCritical warm-up or ELR constrain can effectively remove ELR spread in deep feedforward networks. In both cases, this is done by explicitly or implicitly re-scaling the layer-wise gradients to compensate for blueshift. As discussed in Section 5.1, we find that even removing ELR spread and ensuring equal effective learning rates between the networks, the performance gap between feedforward and residual architectures persists. This could be, at least partly, attributed to the vanishing rank problem, as we measured a rank collapse

in early training phases for deep feedforward architectures (ref. Appendix Section A.3). More experiments would be needed to determine whether this performance gap can be fully attributed to the rank collapse problem or if it would persist, even in a setting where rank collapse is removed. For this purpose, we could envision a constructive approach using a whitening normalization layer. Such approaches do already exist in literature as we discussed in the Appendix, but current approaches seem to carry their own set of issues. Preliminary experiments suggest that the performance gap might persist, even in a setting where the feedforward architecture has full rank. These considerations, which are complementary to the mean-field analysis at initialization by Yang and Schoenholz [61], hint to us that reducing blueshift / shattering gradients is not the only benefit of residual connections. Preventing the collapsing activation rank from deep stacks of matrices might be a strong candidate for another, perhaps commonly overlooked, benefit of residual connections.

Finally, we argue that we have found indications that residual connections offer generalization benefits beyond increased trainability. As ResNets can be seen as an ensemble of relatively shallow nets [54], residual connections clearly are a prior in function space. This can be understood in the path-based view established in Chapter 4, as long paths correspond to more complex functions and short paths to simpler ones. We found that our channel-wise partial linearization experiment in Section 7.2 Figure 9 yields significantly lower average path lengths for a similar performance than a comparable layer-wise technique (ref. Dror et al. [12], this is discussed on Page 8 of our paper). We additionally confirmed this in an experiment of our own (ref. Appendix Section A.4), and found the gap between channel-wise and layer-wise techniques to be bigger for feedforward than for residual architectures. The layer-wise technique only creates paths of the same length, akin to a feedforward network, whereas the channel-wise linearization technique produces paths of different length, akin to a ResNet. The fact that these networks were extracted after training suggests this result is independent of the trainability properties of networks with short paths. Still, this result should be interpreted with caution as the extraction of the sub-network is still a gradient-based process. Overall, the results indicate that the architectural prior behind residual connections, i.e. aggregating many short paths (simple functions) with fewer longer ones (complex functions), is well-aligned with natural data and presents generalization capabilities that go beyond numerical issues.

We conclude that empirical evidence suggests that residual connections present more advantages than commonly believed: beyond reducing shattering gradients/blueshift, residual connections seem to reduce the rank collapse phenomenon in deep networks, and might even possess further advantages beyond increased trainability, namely increased generalization capabilities. More research is needed to determine whether this is the case.

6 Conclusion

In this work, we used a spectral analysis of the w-o landscape as a proxy for understanding network trainability. Expanding related work, we developed a method to quantify 'rough' w-o landscapes and applied this analysis to different common network components to understand which components increase a network's trainability and which are detrimental. In experiments, we found that smooth w-o landscapes are a necessary but not sufficient criterion, as other factors such as vanishing rank can limit trainability.

Many new architectures and design patterns have been proposed over the course of the last years, but few stuck over time like BatchNorm or ResNets. Coincidentally, we found that both have more profound effects on optimization than initially thought: BatchNorm performs a dynamic re-scaling of layer-wise effective learning rates which brings them closer together over time, and residual connections might be a prior in function space with greater generalization capabilities than other numerically stable network designs.

We believe that some of the most valuable insights we gained are non-obvious connections between the different network and optimization components: for example, we found that normalization layers, warm-up, and weight decay all jointly regulate the evolution of layer-wise effective learning rates. By understanding this connection, we hope to massively reduce the architectural and procedural search-space in future experiments. In a world where compute is limited, but network design choices or possible hyper-parameters combinations are countless, we believe that finding underlying patterns that explain multiple design choices at once can be highly beneficial.

The fact that a nonlinear advantage persists between shallow architectures extracted from deeper architectures after training and similarly shallow architectures trained from scratch suggests that our understanding of network training in early training phases might be incomplete. It could be of great interest to see whether such a gap also persists for models of greater size and what optimization components (e.g. scheduler, optimizer, architecture) influence the magnitude of this gap.

Using those insights to create new architectures, learning rate schedulers, regularizers or optimizers is the next logical step, but poses significant challenges in practice. The currently exponentially growing amount of submissions at conferences like NeurIPS shows that such undergoing should be approached with great care, as many possible discoveries have already been made, maybe even accidentally through trial-and-error. Additionally, in current times, the experimental verification of a new proposed architecture requires substantial amounts of compute and time to design a careful experimental setup that can show improvements compared to state-of-the-art architectures. For this reason, we believe this to be outside of the scope of current work but an interesting direction for future work. We hope that our contributions serve as a basis for future innovations in architecture design and optimization strategies, paving the way for more efficient and robust deep learning systems.

7 List of Publications / Personal Contributions

In this Section, we discuss the personal contributions of the co-authors to the publications printed in their original version in the following.

7.1 Ringing ReLUs: Harmonic Distortion Analysis of Nonlinear Feedforward Networks

Published at ICLR 2021. **Co-Author's Contributions:** David Hartmann helped me migrate my old codebase into the research framework he wrote (MiniFlask), which was used for all experiments. He also helped with the plotting the figures in the paper. The underlying theory was developed jointly with Michael Wand.

RINGING RELUS: HARMONIC DISTORTION ANALYSIS OF NONLINEAR FEEDFORWARD NETWORKS

Christian H.X. Ali Mehmeti-Göpel
Institute of Computer Science
Johannes-Gutenberg University Mainz
Staudingerweg 9, 55122 Mainz, Germany
chalimeh@uni-mainz.de

David Hartmann
Institute of Computer Science
Johannes Gutenberg-University of Mainz
Staudingerweg 9, 55128 Mainz, Germany
dahartma@uni-mainz.de

Michael Wand
Institute of Computer Science
Johannes Gutenberg-University of Mainz
Staudingerweg 9, 55128 Mainz, Germany
mwand@uni-mainz.de

ABSTRACT

In this paper, we apply harmonic distortion analysis to understand the effect of nonlinearities in the spectral domain. Each nonlinear layer creates higher-frequency harmonics, which we call "blueshift", whose magnitude increases with network depth, thereby increasing the “roughness” of the output landscape. Unlike differential models (such as vanishing gradients, sharpness), this provides a more global view of how network architectures behave across larger areas of their parameter domain. For example, the model predicts that residual connections are able to counter the effect by dampening corresponding higher frequency modes. We empirically verify the connection between blueshift and architectural choices, and provide evidence for a connection with trainability.

1 INTRODUCTION

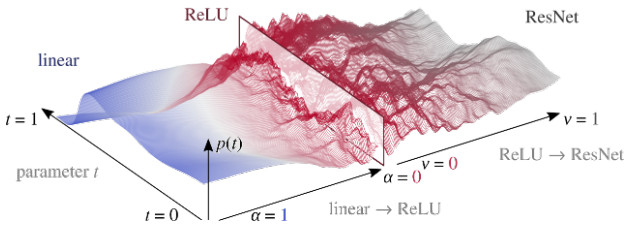


Figure 1: Continous transition of a loss path between linear feedforward ("linear"), nonlinear feedforward ("ReLU") and nonlinear residual ("ResNet") regimes. Graph: loss path near initialization of a ResNet56 v2 with LReLUs with negative slope $\alpha \in [0, 1]$ and residual branch weight $\nu \in [0, 1]$. Left: $\alpha = 0, \nu = 0$, Middle: $\alpha = 1, \nu = 0$, Right: $\alpha = 1, \nu = 1$.

In the past decade, the emergence of practical deep neural networks arguably has had disruptive impact on applications of machine learning. Depth as such appears to be key to expressive models (Raghu et al., 2017). However, depth also comes with challenges concerning training stability. Theoretical problems include vanishing and exploding gradients (Hochreiter, 1991), chaotic feed-forward dynamics (Poole et al., 2016), or decorrelation of gradients (Balduzzi et al., 2017). In practice, a number of “recipes” are widely used, such as specific nonlinearities (Glorot et al., 2011; He et al., 2015), normalization methods such as batch normalization (Ioffe & Szegedy, 2015), shortcut architectures (Srivastava et al., 2015; He et al., 2016a;b), or multi-path architecture with (Huang et al., 2017) and without shortcuts (Szegedy et al., 2016). Broadly speaking, a key research question is to understand how the shape of the network function, i.e., the map from inputs and parameters to outputs, is affected by architectural choices.

Our paper considers specifically the *roughness* of the *weights-to-outputs function* ("*w-o function*") of nonlinear feed-forward networks. Motivated by the recent visualizations of (Li et al., 2018), which show how depth increases roughness and residual connections smoothen the output again, our goal is to provide an analytical explanation of this effect, and study its implications on network design and trainability. To this end, we first formalize "roughness" as the decay-rate of the expected power spectrum of a function class. Our main contribution is to then apply *harmonic distortion analysis* to nonlinear feedforward networks, which predicts the creation of high-frequency "harmonics" (thereby "blueshifting" the power spectrum) by polynomial nonlinearities with large higher-order coefficients. Based on this model, we discuss how network depth increases blueshift and thus roughness, while shortcut connections, low-degree nonlinearities and parallel computation paths dampen it. In relation to trainability, we show an analytic link between blueshift and exploding gradients. Unlike the former model, the spectral view describes a more global behavior of the w-o function over regions in the parameter domain.

Experiments confirm the theoretical predictions: We observe the predicted effects of depth, shortcuts and parallel computation on blueshift, and are able to differentiate different types of nonlinearities by the decay rate of coefficients of a polynomial approximation. The findings are in-line with known advantages in trainability of the different architectures. We further strengthen the evidence by training a large set of networks with a different amount of nonlinearity and depth, which shows a clear correlation between blueshift and training-problems, as well as a trade-off with expressivity.

In summary, our paper explains how network architecture affects roughness, shows a connection to trainability, and thereby provides a new tool for analyzing the design of deep networks.

2 Related Work

Vanishing or exploding gradients are a central numerical problem (Hochreiter, 1991; Pascanu et al., 2013; Yang et al., 2019): If the the magnitudes of the singular values of layer Jacobians deviates from one, subspaces are attenuated ($|\sigma| < 1$) or amplified ($|\sigma| > 1$), potentially cascading exponentially over multiple layers (Pennington et al., 2017a). Formally, the behavior of stacks of matrices and nonlinear functions can be modeled by random matrix theory or Gaussian mean-field approximations (Poole et al., 2016; Pennington et al., 2017b; 2018). The gist is that at initialization, orthogonal weight matrices are needed, which is challenging for convolutional architectures. A solution for $\tanh$-networks is given by Xiao et al. (2018); for ReLU, there is a negative result (Pennington et al., 2017b). Using mean-field theory, it can be shown that batch normalization (Ioffe & Szegedy, 2015) leads to exploding gradients at initialization (Yang et al., 2019) (which equalize after a few steps, but that might be too late (Frankle et al., 2020)).

A different route is taken by Balduzzi et al. (2017), who observe an increasing decorrelation of gradients in the input space. Similar to our paper, they show that deeper networks lead to spectral whitening (starting from brown noise); however, the analysis is performed with respect to the inputs $\mathbf{x}$, not weights $\mathbf{W}$. The scale-space structure shown by our model might give further hints on the mechanisms behind training difficulties.

By visualizing random slices of the loss surface, Li et al. (2018) observe that the loss surface of deep feedforward networks transitions between nearly convex to chaotic with increasing depth; our work explains these observations by spectral analysis. Duvenaud et al. (2014) visualize pathologies on the landscape of deep gaussian processes that model deep wide-limit nonlinear networks. Fourier analysis of network functions (Candès, 1999) wrt. input (Rahaman et al., 2019; Xu et al., 2019; Xu, 2018; Basri et al., 2019; Yang & Salman, 2019) has been used to show an inductive bias towards low-frequency functions (wrt. input $\mathbf{x}$), as well as a strong anisotropy of this spectrum. Wang et al. (2020) prove under some assumptions that all "bad" local minima of a deep residual network are very shallow.

3 Harmonic Distortion

We now analyze the effect of a nonlinearity by relating the Fourier spectrum of a preactivation with that of its postactivation. Let f denote the preactivation of a single neuron of a neural network consisting of L-layers. We use $\mathbf{x}$ to denote the input to the whole network and thus to f, $\mathbf{W}$ to denote

the weights, and ϕ to denote the employed nonlinearity. Li et al. (2018) visualize "roughness" using random 2D-slices in weight space. We follow their basic idea and consider 1D slices

$$p(t) = f(\mathbf{x}, \mathbf{W} + \alpha^{-1} \cdot t \cdot \mathbf{D}) \tag{1}$$

for random directions $\mathbf{D}$ and $t \in [0, 1]$. $\mathbf{D}$ is initialized with entries from $\mathcal{N}_{0,1}$ and normalized to $||\mathbf{D}||_F = 1$. α determines the path length. By varying $\mathbf{D}$, this samples a ball of radius α around a point $\mathbf{W}$ in parameter space. For $\phi = id$, the network f is multi-linear in $\mathbf{W}$ and thus p is polynomial in t; empirically, this yields rather smooth functions (Fig. 1). To understand general nonlinearities ϕ better, we represent p by a complex Fourier series (Appendix B.1 discusses convergence and approximation quality):

$$p(t) = \sum_{k=-\infty}^{\infty} z_k \exp\left(2\pi i k t\right), \quad z_k \in \mathbb{C}. \tag{2}$$

Here, the sequence $z : \mathbb{Z} \to \mathbb{C}$ contains the Fourier coefficients for $p : [0, 1] \to \mathbb{R}$. As p is real, z is symmetric in the sense of $z_k = \overline{z_{-k}}$.

3.1 Formalizing Roughness

The roughness of a class of random functions can be characterized by the statistics of their power-spectrum (Musgrave, 1993): Given a random process that generates functions p, we consider mean μ_k and variance σ_k^2 of the Fourier coefficients z_k. For paths with short length as used in our experiments, the means μ_k are empirically very close to zero except for z_0 ("DC" coefficient), which is excluded in all experiments. Therefore, we can focus on variance: In general, functions where the variance of high-frequency components drop off more quickly will appear smoother. A common model, which often fits natural data well, is fractal Brownian motion (FBM-) noise, where the σ_k drop off according to a power law:

$$\sigma_k \sim \mathcal{O}\left(1/k^h\right) \text{ for some } h > 0. \tag{3}$$

The so-called "fractal coefficient" h describes the roughness of the noise function. A similar approach has been taken by Hoffer et al. (2017), who modeled the loss surface by analyzing the dynamics of a random walk on a random potential. In our experiments, we estimate the average power-spectrum $\mathbb{E}_{\mathbf{D}}(|z_k|^2)$ (by sampling $\mathbf{D}$ uniformly on a unit sphere) and fit a power-law to these spectra in order to quantify the roughness in a single number h.

Experiments (Section 4) and analytical arguments (Appendix B.2), show that the FBM/power-law model is a realistic model of the functions computed by the lower layers of a neural network. For higher layers the fit becomes worse, a phenomenon we will explore in the next chapter using harmonic distortion analysis.

3.2 Why is the Output Function Getting Rougher?

Intuitively, applying ReLU to a function p is reminiscent of clipping an audio signal in amplitude, which is known to produce high-frequent ringing artifacts. We describe the effect of a single nonlinearity ϕ on the spectrum of a preactivation p; Inductively, this describes the spectral shift of the whole network. For the analysis, we assume that ϕ is a K-th order polynomial:

$$\phi(x) = \sum_{j=0}^{K} a_j x^j. \tag{4}$$

The effect of polynomial nonlinear maps on the spectrum of a function can be understood by harmonic distortion analysis (see e.g. Feynman et al. (1965), Ch. 50.8). We can simply (see Appendices B.1 – B.3 for details) plug the Fourier expansion of p into the polynomial representation of ϕ:

$$\phi(p(t)) = \sum_{j=0}^{K} a_j \left[\sum_{k=-\infty}^{\infty} z_k \exp\left(2\pi i k t\right)\right]^j. \tag{5}$$

The convolution theorem tells us that j-th power of functions corresponds to convolving the spectrum z of the function j-times with itself. We designate by z the vector containing all Fourier coefficients z_k and by $\bigotimes$ the convolution operator. We can then write the spectrum of the output z' as:

$$z' := \mathcal{F}(\phi(p)) = \sum_{j=0}^{K} a_j \bigotimes_{1=1}^{j} z. \tag{6}$$

Discussion: We can make three important observations: • First, each repeated auto-convolutions in Eq. 6 broadens the spectrum by adding higher-frequency terms. We call this broadening effect *blueshift*. The exact magnitude is hard to quantify and thus left to the experiments (Appendix B.4 gives informal arguments for a growth of $\mathcal{O}(j^{1/2})$ rather than the trivial upper bound of $\mathcal{O}(j)$). • Second and correspondingly, larger coefficients a_j for larger orders j, increase the blueshift. • Third, j-fold convolutions correspond to j-th powers of the z_k. Hence, larger magnitudes $|z_k|$ also increase the blueshift (the nonlinearity becomes more visible with larger magnitude).

3.3 Common Nonlinearities

In practice, nonlinearities are usually not polynomial, and might not even have a globally convergent Taylor series. It is possible to approximate any continuous function by a polynomial (Stone-Weierstrass theorem). We conjecture that a close polynomial approximation will be sufficient for a qualitative prediction of the blueshift effect (our theoretical model does not guarantee convergence – we therefore validate this claim experimentally). We employ a Chebyshef approximation (which is reasonably stable and non-oscillatory) on the interval $[-5, 5]$ and compare the speed at which higher order coefficients drop off:

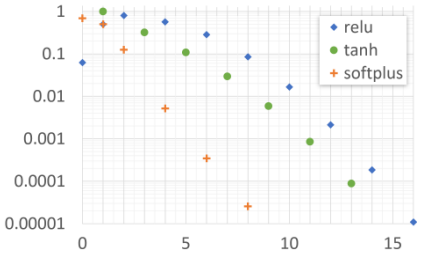


Figure 2: Absolute values of polynomial coefficients: $|t_j|$ over j; see also Fig. 19.

Fig. 2 shows the coefficient magnitudes for $\mathrm{ReLU}, \tanh$ and $\mathrm{softplus}$ (which we focus on in our experiments). $\mathrm{softplus}$ has the strongest drop-off, followed by $\tanh$ and ReLU. Appendix F.1 gives more details and covers several additional popular nonlinearities (Figures 18, 19). Figure 3, 17 show that this correlates with blueshift, as expected.

3.4 Connection to Exploding Gradients

The creation of higher-frequency harmonics directly affects gradient-based training methods because the gradient operator is a high-pass filter in the spectral domain: Taking derivatives multiplies the Fourier coefficients by $2\pi ik$, amplifying coefficients linearly with frequency k (which is trivial seen by taking the derivatives of Eq. 2). The reciprocal fractal exponent $r := h^{-1}$ has exponential influence on the average gradient magnitude: With $|z_k|^2 \sim \Theta(k^{-h}) = \Theta(k^r)$, we obtain gradients $k \cdot |z_k| \in \Theta(k^{1+r/2})$. Blueshift thus causes exploding gradients: Weights of lower layers are shifted more often, creating higher-order harmonics, which correspond to larger norm of the gradient function $\|p'(t)\| = 2\pi\sqrt{\sum_k k^2|z_k|^2}$, which means that there must be, on average, larger gradients along the path considered. Independently of this, both exploding and vanishing gradients can be caused by other effects than blueshift: Saxe et al. (2014) show that even deep linear networks can suffer from exploding gradients caused by stacking matrices with non-uniform singular spectrum.

The blueshift effect itself is independent of the condition number of the weight matrices: blueshift would occur even when only stacking nonlinearities. Although it is possible to find a locally linear region of the loss surface (e.g. CReLU + Looks Linear initialization (Balduzzi et al., 2017)) and locally achieve dynamic isometry (Xiao et al., 2018), blueshift effects become relevant again when this region is eventually left during the training process (ref. Appendix Figure 12). In contrast, residual connections affect the entire loss surface and our model is able to capture this effect.

3.5 Residual and Multipath Networks

Residual Networks exhibit *two* different dampening effects on the loss surface that can be experimentally isolated:

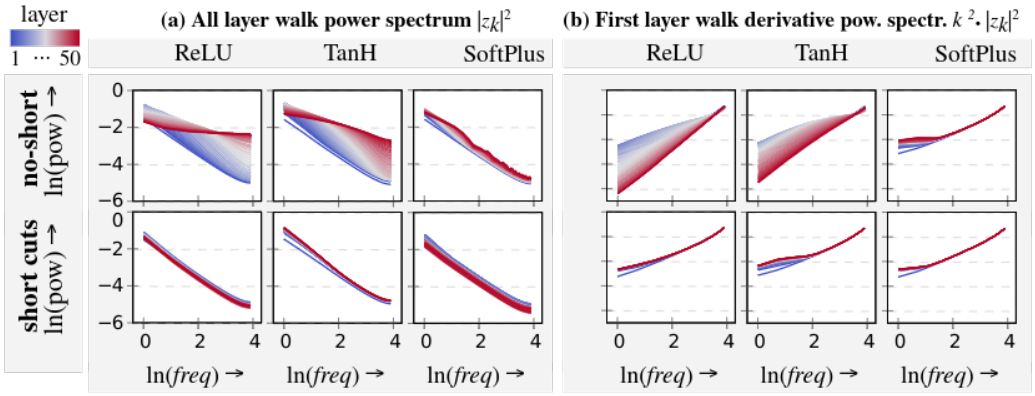


Figure 3: Normalized power spectra z_k per layer (red is deeper) in a 50-layer CNN at initialization, averaged over multiple neurons, for various ϕ. (a) On the left, all weights are varied and spectrum of $p(t)$ is shown. (b) On the right, only the first layer is varied and the spectrum of $p'(t)$ is shown.

Exponential downweighting (ED): Residual networks repeatedly compute the sum of the result of a small feedforward network with its input. By construction, this reduces the relative weight of the nonlinear output of the residual block by mixing it with the unprocessed signal. Doing this repeatedly creates an ensemble (Huang et al., 2016) in which the weight of contributions shrink exponentially with the number of nonlinear processing steps. As detailed e.g. in (Veit et al., 2016), each nonlinear block only contributes partially to the overall result. The weight of the signal component passing through nonlinearities shrinks exponentially due to vanishing gradient effects (at initialization, non-aligned non-uniform singular value spectra lead to exponential dampening; after training, a similar dampening effect is observed empirically by Veit et al.).

Frequency-dependent signal-averaging (FDSA): According to the blueshift model, the outputs of the residual blocks will contain more high-frequency Fourier coefficients (from harmonics). The additional harmonics depend on the weights in the residual block, which are statistically independent of the input (see Appendix B.5 for details). Therefore, we expect to see a reduction in the expected linear correlation between the main and residual branch for higher frequencies. As an effect, when adding the main and residual branch, higher frequencies get dampened even more.

The second effect is weaker than the first, governed by the law of large numbers (i.e., $n^{-\frac{1}{2}}$ decay for averaging n independent values), but is also more broadly applicable: It should generally occur when averaging over multiple computational paths with independent weights, while residual connections require an identity in one path.

4 Experimental Results: Measured Spectra

The model presented in the previous section gives us only qualitative hints on the magnitude of the blueshift, the behavior of non-polynomial nonlinearities, and the magnitude of ED and FDSA effects. We therefore now validate these qualitative predictions experimentally.

We consider two architectures: a basic Toy-CNN model with a constant number of features in each layer for fair per-layer statistics and ResNet v1 variants for a more "realistic" model. We used Cifar10 as input data, but all figures in this section look qualitatively similar on other datasets; A more detailed network description, parameters and experiments on MNIST can be found in the Appendix. In this section, we always measure the effect of blueshift in the region around initialization (i.e., $\mathbf{W}$ is the initialization point); During training, the loss surface tends to get smoother but the effect persists (ref. Appendix Figure 11).

4.1 Effects of blueshift on a loss path

To show the effect of nonlinearity and residual connections on the loss surface, we sample a single loss path on a ResNet56 with Leaky ReLU activations. We can now continuously tune the amount of

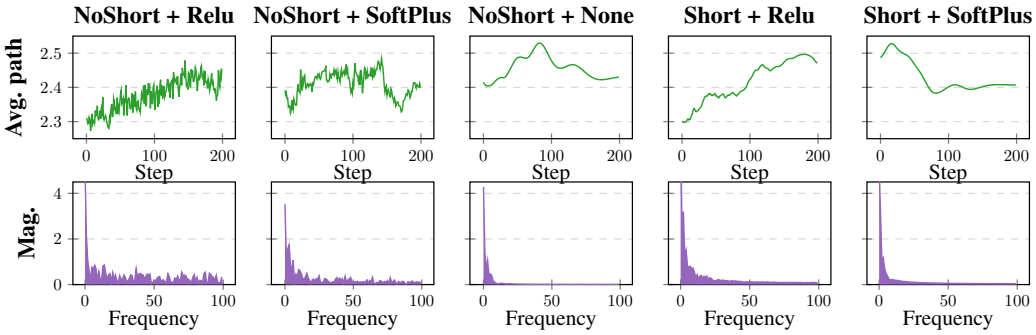


Figure 4: "Average loss path" and its respective magnitude spectrum out of 500 random loss paths for different variants of ResNet56 at initialization.

nonlinearity by adjusting its negative slope α. We can equally continuously tune the relative weight of the residual branch by multiplying it with a factor ν in all residual blocks. Figure 1 shows a transition from a linear feedforward to a nonlinear feedforward to a nonlinear residual regime; We can observe the apparition of harmonics with increasing nonlinearity that get dampened when switching on residual connections.

4.2 Effects of blueshift on the average power spectrum of the w-o surface

We now want to view the effects of blueshift on the expected power-spectrum of the w-o surface. We use a "Toy-CNN" with depth $L = 50$ and track the w-o surfaces of each layer while we walk in a single random direction in weight space. We transform the resulting 1D-paths with discrete FFT and average all power spectra over all neurons in a layer, all input receptive fields and all batch images. Fig. 3a (left side) shows the resulting average power spectrum, normalized by function norm. The right hand side (b) shows the same plot, but with $\mathbf{D}$ restricted to only changing the weights of the first layer of the network, and taking the derivative of the resulting function (by weighting by frequency, $k^2 \cdot |z_k|^2$) before normalization. As we use only 100 samples without prior bandlimiting, aliasing phenomena occur as a slight upward slant is visible at the high-frequency end of all plots.

Spectral shift over depth: The results in Fig. 3a confirm our predictions quite well: A spectral blueshift is clearly visible and, as expected, ReLU shows a strong blueshift, $\tanh$ a slightly weaker blueshift, and $\mathrm{softplus}$ only a small effect. Harmonics show up as a bump in the spectrum that travels towards higher frequencies with increasing layer. This extends to other nonlinearties as well (see Appendix, Fig. 17). The plot also confirms that the power spectrum of the lower layers is well described by a $1/k$-power law, as predicted.

Scale shift of a single layer parametrization: By only changing the weights in a single layer and taking the derivative (Fig. 3b), we see that blueshift is responsible for a gradient scale mismatch between earlier and deeper layers in networks without skip-connections: the more nonlinearities are between the modified weight and the output, the higher the gradient magnitude.

Residual connections: Our model predicts that residual connections will reduce the blueshift strongly by **ED** and **FDSA** (ref. Section 3.5). This is also consistent with our observations: higher layers show almost the same $1/k$-spectrum as the initial layer. Looking at the first layer weights, weighted by frequency, the response is almost flat, with a small emphasis on low-frequency weights.

4.3 Quantifying and measuring spectral shift

Utilizing the power-law model presented in section 3.1, we can measure blueshift by estimating the fractal exponent h of a loss path via a power-law fit of the magnitude spectrum. Figure 4 represents what the "average loss path" looks like for a ResNet 56 at initialization with different activation functions, with and without shortcuts. For each architecture, we sample paths in 500 different random directions $\mathbf{D}$ and assess the fractal exponent of every sampled path, in order to visualize the path

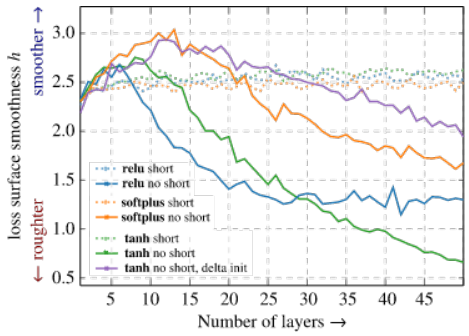


Figure 5: Smoothness of the loss surface at initialization for a Toy-CNN of varying depth with and without shortcuts.

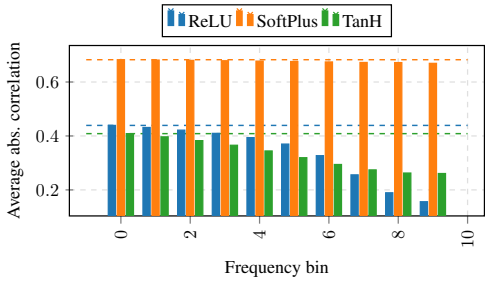


Figure 6: Average absolute path correlation per frequency bin of the main branch and the residual branch of a ResNet56 v2 with different activation functions at initialization. Lower bin number indicates lower frequencies.

with median h value and its associated spectrum. We clearly see that nonlinear architectures without skip-connections present most blueshift, the linear architecture has no blueshift and the architectures with skip-connections do also exhibit, albeit dampened, blueshift.

In Figure 5, we assess the average loss path smoothness h of a Toy-CNN with varying depth at initialization. For averaging, we initialize each network 50 times and sample 20 paths in different random directions for each initialization. The results confirm quantitatively the visual results of Li et al. (2018) that the loss surface becomes rougher with increasing network depth and that shortcut connections have a strong smoothing effect. Since our roughness measure is a non-pointwise view, we see that even the delta-initialized network that does not suffer from vanishing gradients at initialization still gets rougher with increasing depth. The power-law fit still has some limitations: As seen in Fig. 3, a power-law fit is imperfect as the higher layers are not straight lines anymore. This shows in curves of Fig. 5 as apparent increase of smoothness in the first layers for the "NoShort" architectures.

4.4 Residual networks control harmonics via ED and FDSA

In Section 3.5, we described two different mechanisms that allow ResNets to dampen high-frequency modes. We now want to verify the existence of **FDSA** and further individuate the two effects.

FDSA predicts that skip-connections have a dampening effect on high frequencies in the loss surface because the correlation between the main branch and the residual branch is decreasing with higher frequencies. We empirically verify this prediction on a ResNet 56 v2 at initialization by walking in 50 random directions and measuring the respective w-o function (we only sample 10% of the outputs for computational reasons) of the two summands in each residual block just before addition. Using Gaussian filters in Fourier domain, we can compute the linear correlation of each pair of paths (block input/output) per frequency bin and average over all batch images, neurons and blocks in the network. In Figure 6, we see that the absolute linear correlation is decaying for all architectures with increasing frequency, with ReLU showing the strongest effect, again followed by $\tanh$ and $\mathrm{softplus}$ in that order, confirming our prediction. This property seems to allow a residual network to automatically regulate the higher-frequency harmonics content of the output.

We want to individuate **FDSA** in a setting where **ED** is not present. We modify our Toy-CNN and ResNet "NoShort" architectures such that we replace every n-feature *convolution / batch normalization / activation* sequence in the network with a $a \cdot n$ feature sequence where the a feature groups get averaged after the activation (in reality we only use summation, since normalizing is taken care of by batch normalization). Since high frequencies added by the nonlinearity are less correlated between features (ref. Fig. 6), we expect that averaging feature groups smoothen the w-o function; we see on the right of Figure 8 that this is indeed the case.

Finally, we want to demonstrate that the positioning of the activation function is crucial. Our model predicts that since ResNet v2 blocks (He et al., 2016b) use summation after ReLU, we will see a stronger averaging effect than in ResNet v1. Indeed, looking at their respective average power spectra in a random direction in Figure 7, we see this confirmed.

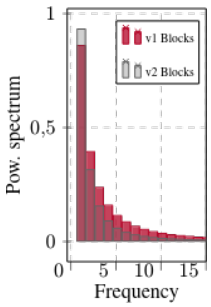


Figure 7: Normalized power spectrum of the last layer of a ResNet194 v1/v2 (post/pre-activation) at initialization.

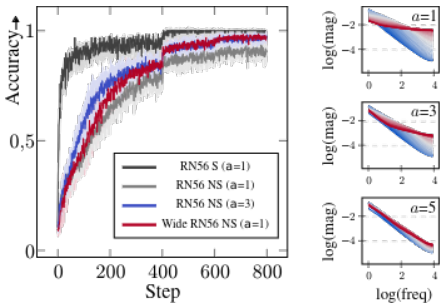


Figure 8: **Left**: Comparing the training performance on Cifar10 of a feature-averaging ResNet56 v1 "NoShort" to its vanilla, wide and residual equivalent. **Right**: Spectrum shift of averaging Toy-CNN 50 layers at initialization, varying averaging factor a.

5 Empirical Results: Training

In this section, we want to demonstrate a correlation between blueshift and network trainability. Again, we use variants of the ResNet v1/v2 network, exact network parameters and training hyperparameters can be found in tabular form in Appendix A. We use Cifar10 as training data, training results for Cifar100 can be found in Appendix C

Controlling Blueshift via feature averaging: We want to investigate the impact of the smoothing of the loss surface via **FDSA** (ref. Section 3.5) on training speeds. We choose a ResNet56 "NoShort" as our architecture, since at this depth simple feedforward networks with batch-normalization start to become difficult to train. We experiment on the averaging network (ref. Section 4.4) with $a = 3$ since we experimentally determined that for $a > 3$ the performance stops increasing. For fairness, we included a Wide ResNet56 "NoShort" with approximately the same parameter count than the averaging network. We see the averaged results over five runs in Figure 8 (shaded areas represent the standard deviation). We observe that the averaged network outperforms both the original network and the wide network while still performing worse than the network with skip-connections. This is consistent with our theory since **ED** has a bigger smoothing effect than **FDSA**.

Controlling Blueshift with Leaky ReLU: As demonstrated in Figure 1, the amount of blueshift in a network can be controlled by tuning Leaky ReLU's negative slope α. We now want to show that networks with a very strong blueshift are hard to train but conversely some amount of nonlinearity is needed for expressivity. For this, we train a ResNet56 v1/v2 with and without shortcuts for 30 epochs once on Cifar10 for different values of α and visualize the training accuracy (average over the last 25 values) and compare it to the blueshift of the network at initialization. On Figure 9, we see that for the networks without skip-connections, training becomes more difficult with regard to network depth, which correlates with increased blueshift. We see that this effect can be alleviated by making the network more linear (by increasing α) and thus reducing blueshift. For networks with skip-connections, we see that blueshift is greatly reduced and that no deterioration of trainability with regard to network depth is noticeable. Conversely, networks that near a linear regime for high values of α tend to train worse since they lack expressivity. Interestingly, this effect is stronger for ResNet v2 blocks than for ResNet v1 blocks because the former make the loss surface smoother than v1 blocks (ref. Figure 7).

6 Discussion

Our experimental findings validate the predictions of our model with respect to the roughness of the w-o function: Nonlinearities with larger higher-order polynomial coeffiecnts create larger variances in the high-frequency part of the spectrum. While related results on increasing complexity with depth have been given earlier (Poole et al., 2016; Schoenholz et al., 2017), the harmonics model offers a simple and more global view of the roughness of the loss surface and how it is affected by nonlinearity choice and residual connections. Our observations on training speed are consistent with the hypothesis that spectral blueshift impedes training: Architectural choices (residual connections,

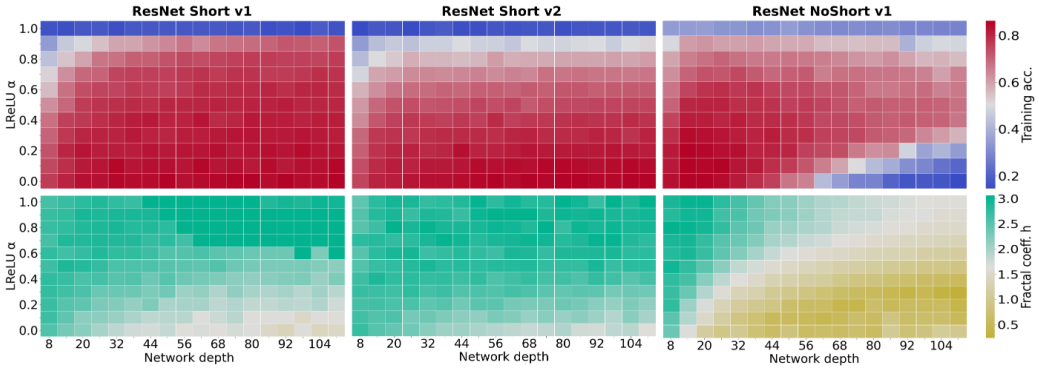


Figure 9: Comparing w-o smoothness and trainability for ResNets with varying depth and amount of nonlinearity. **Upper row**: training accuracy after 30 epochs of training on Cifar10. **Bottom row:** blueshift at initialization indicated by the fractal coefficient of the network's last layer.

ResNet v2 blocks, LReLU) that control harmonics generation are also those that are easier to train to the their full performance. Our model is also consistent with the empirical observations of Han et al. (2017), who found that ResNet performance varies with the location and number of ReLUs. Further, the results in Figure 9 give strong hints for a correlation between high-frequency content and problems in trainability for deep but very nonlinear networks (small α). From an analytical perspective, architectures that are prone to blueshift are prone to vanishing/exploding gradients: We have shown that high magnitudes of high-frequencies in the w-o functions lead to an increase in the expected gradient magnitude. Further, in a simple multi-layer network without shortcuts, we expect more high-frequency content on lower than on higher layers as soon as the nonlinearity of the activation function is actually is utilized (and therefore must also creates harmonics). A rescaling of layers can reduce these discrepancies, but the normalization will still necessarily depend on the position in parameter space. As far as accuracy is concerned, there is an obvious trade-off between blueshift and expressivity: networks with low-degree polynomials as nonlinearities contain less high-frequent harmonics but cannot approximate complex functions (Kidger & Lyons, 2020). Future work could consist in further exploring this trade-off. Further, it would be interesting to examine potential architectural alternatives to residual connections with comparable performance and trainability.

7 Conclusion and Future Work

In this paper, we have analyzed the effect of different architectures of nonlinear networks on the smoothness of the computed function. Specifically, we have linked polynomial nonlinearity to harmonics creation and validated experimentally that the mean power spectrum shifts towards higher frequencies ("blueshift"). Further, we have described two distinct effects that explain the smoothing effect of shortcut connections, which we also confirmed experimentally. Finally, we empirically linked reduced blueshift to increased trainability and training speeds.

In future work, we hope to derive explicit closed-form equations for how architectural designs like network depth, residual connections and nonlinearity choice affect the w-o function. It could be also interesting to further explore the roughness-expressability trade-off that we described earlier.

8 Acknowledgements

This work has been partially supported by the RMU Network for Deep Continuous-Discrete Machine Learning (DeCoDeML project). We wish to thank Jan Disselhoff for valuable discussions and the annonymous reviewers for their helpful feedback.

REFERENCES

David Balduzzi, Marcus Frean, Lennox Leary, J. P. Lewis, Kurt Wan-Duo Ma, and Brian McWilliams. The shattered gradients problem: If resnets are the answer, then what is the question? In Doina Precup and Yee Whye Teh (eds.), *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pp. 342–350. PMLR, 2017.

Ronen Basri, David W. Jacobs, Yoni Kasten, and Shira Kritchman. The convergence rate of neural networks for learned functions of different frequencies. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*, pp. 4763–4772, 2019.

Emmanuel J Candès. Harmonic analysis of neural networks. *Applied and Computational Harmonic Analysis*, 6(2):197–218, 1999.

David Duvenaud, Oren Rippel, Ryan P. Adams, and Zoubin Ghahramani. Avoiding pathologies in very deep networks. In *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics, AISTATS 2014, Reykjavik, Iceland, April 22-25, 2014*, volume 33 of *JMLR Workshop and Conference Proceedings*, pp. 202–210. JMLR.org, 2014.

Richard P Feynman, Robert B Leighton, and Matthew Sands. The feynman lectures on physics; vol. i. *American Journal of Physics*, 50(8), 1965.

Jonathan Frankle, David J. Schwab, and Ari S. Morcos. The early phase of neural network training. *CoRR*, abs/2002.10365, 2020.

Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 315–323, 2011.

Guy Gur-Ari, Daniel A. Roberts, and Ethan Dyer. Gradient descent happens in a tiny subspace. *CoRR*, abs/1812.04754, 2018.

Dongyoon Han, Jiwhan Kim, and Junmo Kim. Deep pyramidal residual networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pp. 6307–6315. IEEE Computer Society, 2017. doi: 10.1109/CVPR.2017.668.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*, pp. 1026–1034. IEEE Computer Society, 2015. doi: 10.1109/ICCV.2015.123.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pp. 770–778. IEEE Computer Society, 2016a. doi: 10.1109/CVPR.2016.90.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling (eds.), *Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part IV*, volume 9908 of *Lecture Notes in Computer Science*, pp. 630–645. Springer, 2016b. doi: 10.1007/978-3-319-46493-0_38.

S. Hochreiter. Untersuchungen zu dynamischen neuronalen netzen. Diploma thesis, TU Munich, 1991.

Elad Hoffer, Itay Hubara, and Daniel Soudry. Train longer, generalize better: closing the generalization gap in large batch training of neural networks. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pp. 1731–1741, 2017.

Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q Weinberger. Deep networks with stochastic depth. In *European conference on computer vision*, pp. 646–661. Springer, 2016.

Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pp. 2261–2269. IEEE Computer Society, 2017. doi: 10.1109/CVPR.2017.243.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Francis R. Bach and David M. Blei (eds.), *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, volume 37 of *JMLR Workshop and Conference Proceedings*, pp. 448–456. JMLR.org, 2015.

Patrick Kidger and Terry J. Lyons. Universal approximation with deep narrow networks. In Jacob D. Abernethy and Shivani Agarwal (eds.), *Conference on Learning Theory, COLT 2020 , 9-12 July 2020, Virtual Event [Graz, Austria]*, volume 125 of *Proceedings of Machine Learning Research*, pp. 2306–2327. PMLR, 2020.

Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the loss landscape of neural nets. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada*, pp. 6391–6401, 2018.

F.K. Musgrave. *Methods for Realistic Landscape Imaging*. PhD thesis, Yale University, 1993.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013*, volume 28 of *JMLR Workshop and Conference Proceedings*, pp. 1310–1318. JMLR.org, 2013.

Jeffrey Pennington, Samuel Schoenholz, and Surya Ganguli. Resurrecting the sigmoid in deep learning through : theory and practice. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 30*, pp. 4785–4795. Curran Associates, Inc., 2017a.

Jeffrey Pennington, Samuel S. Schoenholz, and Surya Ganguli. Resurrecting the sigmoid in deep learning through dynamical isometry: theory and practice. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pp. 4785–4795, 2017b.

Jeffrey Pennington, Samuel S. Schoenholz, and Surya Ganguli. The emergence of spectral universality in deep networks. In Amos J. Storkey and Fernando Pérez-Cruz (eds.), *International Conference on Artificial Intelligence and Statistics, AISTATS 2018, 9-11 April 2018, Playa Blanca, Lanzarote, Canary Islands, Spain*, volume 84 of *Proceedings of Machine Learning Research*, pp. 1924–1932. PMLR, 2018.

Ben Poole, Subhaneil Lahiri, Maithra Raghu, Jascha Sohl-Dickstein, and Surya Ganguli. Exponential expressivity in deep neural networks through transient chaos. In Daniel D. Lee, Masashi Sugiyama, Ulrike von Luxburg, Isabelle Guyon, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pp. 3360–3368, 2016.

Maithra Raghu, Ben Poole, Jon Kleinberg, Surya Ganguli, and Jascha Sohl Dickstein. On the expressive power of deep neural networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 2847–2854. JMLR.org, 2017.

Nasim Rahaman, Aristide Baratin, Devansh Arpit, Felix Draxler, Min Lin, Fred A. Hamprecht, Yoshua Bengio, and Aaron C. Courville. On the spectral bias of neural networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov (eds.), *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pp. 5301–5310. PMLR, 2019.

Andrew M. Saxe, James L. McClelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. In Yoshua Bengio and Yann LeCun (eds.), *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014.

Samuel S. Schoenholz, Justin Gilmer, Surya Ganguli, and Jascha Sohl-Dickstein. Deep information propagation. In *Proceedings of the 5th International Conference on Learning Representations*, 2017.

Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. *CoRR*, abs/1505.00387, 2015.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2818–2826, 2016.

Andreas Veit, Michael J. Wilber, and Serge J. Belongie. Residual networks behave like ensembles of relatively shallow networks. In Daniel D. Lee, Masashi Sugiyama, Ulrike von Luxburg, Isabelle Guyon, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pp. 550–558, 2016.

Lifu Wang, Bo Shen, Ning Zhao, and Zhiyuan Zhang. Is the skip connection provable to reform the neural network loss landscape? In Christian Bessiere (ed.), *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*, pp. 2792–2798. ijcai.org, 2020. doi: 10.24963/ijcai.2020/387.

Lechao Xiao, Yasaman Bahri, Jascha Sohl-Dickstein, Samuel Schoenholz, and Jeffrey Pennington. Dynamical isometry and a mean field theory of CNNs: How to train 10,000-layer vanilla convolutional neural networks. In Jennifer Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 5393–5402, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

Zhi-Qin John Xu, Yaoyu Zhang, and Yanyang Xiao. Training behavior of deep neural network in frequency domain. In Tom Gedeon, Kok Wai Wong, and Minho Lee (eds.), *Neural Information Processing - 26th International Conference, ICONIP 2019, Sydney, NSW, Australia, December 12-15, 2019, Proceedings, Part I*, volume 11953 of *Lecture Notes in Computer Science*, pp. 264–274. Springer, 2019. doi: 10.1007/978-3-030-36708-4_22.

Zhiqin John Xu. Understanding training and generalization in deep learning by fourier analysis. *CoRR*, abs/1808.04295, 2018.

Greg Yang and Hadi Salman. A fine-grained spectral perspective on neural networks. *CoRR*, abs/1907.10599, 2019.

Greg Yang, Jeffrey Pennington, Vinay Rao, Jascha Sohl-Dickstein, and Samuel S. Schoenholz. A mean field theory of batch normalization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

A Network details

A.1 Network architectures

The implementation for our experiments is based on PyTorch 1.5 and are provided as supplementary material.

Toy-CNN: We create a simple convolutional network consisting of units of 3x3-convolutions with 16 feature channels on all layers ($16 \cdot a$ for the averaging networks), followed by batch normalization and non-linear activation, stacked-up L-times. Optional short-cut connections are added (only if explicitely stated) between each unit (i.e., residual blocks of depth 1). No pooling or striding is used. The input layer uses a stride of 5 to reduce memory demands. The classification layer is composed of a fully-connected layer, softmax and cross-entropy loss.

Residual Network: As a "real-world" example, we use a standard Residual Network v1 (BasicBlock) He et al. (2016a) with varying depth, with ("Short/S") and without ("NoShort/NS") skip-connections. ResNet v2 blocks are used in Figure 1 (v2 blocks show slower transitions) and 6 (correlations are decaying faster in ResNet v2, ref. 3.5). We use the standard number of planes for Cifar-10 training : 16/32/64 (28/56/112 for the wide network).

A.2 FOURIER WALK HYPERPARMETERS

We sample paths consisting of 100 samples each and with path length $\alpha = 1$ (Eq. 1) in all experiments, except for Figures 1, 4 and 6 where we double both values for higher frequency resolution and Figure 12, where we want to walk in a different radius ($\alpha \in \{0.1, 1, 10\}$). All experiments are performed in training mode and with data augmentation at initialization. Datapoints for each architecture are measured on a different mini-batch to exclude bias by batch selection.

A.3 TRAINING HYPERPARMETERS

The hyper-parameters below usually reach the standard test-accuracy of approximately 92-93% for a ResNet56 on Cifar10.

Dataset	Cifar10 (Cifar100 for Figure 10)
Epochs	200
Scheduler	Multistep ($\gamma = 0.1$)
Milestones	100, 150
Learning rate	0.1
Batch size	128
Optimizer	SGD + Momentum
Momentum	0.9
Weight decay	0.0001
Augmentation	Random Flip

B FORMAL DETAILS CONCERNING HARMONIC NETWORK ANALYSIS

B.1 SERIES REPRESENTATION AND CONVERGENCE

In our paper, we consider feed-forward networks consisting of L layers. Each layer $l = 1...L$ can be one of the following:

- An **affine map**
$$\mathbf{x} \mapsto \mathbf{W}^{(l)}\mathbf{x} + \mathbf{b}^{(l)} \tag{7}$$
that transforms the input using a linear map and an offset vector.
- A **nonlinearity**
$$\mathbf{x} \mapsto \phi^{(l)}(\mathbf{x}) \tag{8}$$
that applies a (potentially layer-specific) nonlinear function to each input.

All of the discussed architectures (including convolutional networks, striding/average pooling, residual connections, parallel computations, and layer-dependent nonlinearitiy) can be expressed as a composition of these two layer types. We currently do not address max-pooling, and understand batch-normalization layers statically as a fixed scaling and shifting (not analyzing their training dynamics). The loss function itself (such as a sequence of softmax and cross-entropy) is also not included in our analysis – we restrict our consideration to layer outputs before any classification. As logistic regression (or similar problems) are numerically well-understood already, this is not a major obstacle. All our experiments show layer outputs before classification (softmax).

The main restriction of our formal model is that we assume that all nonlinearities are polynomials of finite degree[1]. Under this condition, all outputs and intermediate results of the whole network can be

[1]This means that the analytical results are currently not established for practical non-polynomial nonlinearities. The results in our paper obtained from polynomial approximations should therefore be considered experimental results at this point. An analysis of infinite series approximations covering a more general class of nonlinearities is still left for future work.

represented by a large multi-variate polynomial, and any *linear section* $p(t) = f_i^{(l)}(\mathbf{W} + \alpha t \mathbf{D}, \mathbf{x})$ of any output i of any layer l is a univariate polynomial of finite degree.

Fourier analysis: Polynomial functions on a finite domain are Lipschitz-continuous and have finite variation; both conditions are sufficient for the convergence of the Fourier series in an L_2-sense. Further, as the Fourier-basis forms a Schauder-basis of $L_2[0, 1]$, the series expansion is unique.

In our derivation (Equation 5), we plug the series representation into the polynomial and conclude that it can be represented in the spectral domain as a sum of convolutions (Equation 6). To make this more rigorous, we proceed in smaller steps: Let again $p : \mathbb{R}^n \to \mathbb{R}$ denote the linear section along a linear direction in parameter space of any layer output. Our nonlinearity expands to

$$\phi(p(t)) = \sum_{j=0}^{K} a_j p(t)^j. \tag{9}$$

As $p(t)$ is a polynomial of finite degree, $p(t)^j$ for any finite j is also a polynomial of finite degree. This implies that there is a unique Fourier series that represents p, characterized by the coefficients sequence $z : \mathbb{Z} \to \mathbb{C}$. According to the convolution theorem for the complex Fourier series, the Fourier series of $p \cdot p$ is given by $z \otimes z$, where

$$[z \otimes u]_k := \sum_{m=-\infty}^{\infty} z_m \cdot u_{k-m}. \tag{10}$$

Because the series must exist (it converges for both p and p^2) and it is unique, the product of the series must be represented by the auto-convolution. Inductively, we obtain p^j corresponding to the j-fold auto-convolution $\bigotimes_{q=1}^{j} z$.

Note: We use the notation $[\mathbf{x}]_i := x_i$ denote the indexing of vectors and sequences that result from a computation.

Approximation quality and finite transforms: The Fourier representation is accurate in an L_2-sense; it converges in L_2-norm, which is an integral measure. This still permits any finite series expansion to have large point-wise errors (no convergence in infinity-norm: large error regions rather shrink to a zero set in the limit). A common example of such problems is Gibb's phenomenon: A discontinuous function (such as the gradient function of a ReLU) will show "overshooting" of constant magnitude near the discontinuities; only the area affected shrink with the frequency order of the Fourier series. Further, even higher-order discontinuities lead to significant high-frequency harmonics (i.e., show bad approximation behavior for truncated series).

While the theory is not affected by this, this might appear to be a concern for the discrete Fourier transformation (DFT) we use in our experiments. The DFT operates at a finite frequency order, using a matching set of regular samples of the continuous function. Here, it is important to note that the DFT, as a mapping from $\mathbb{C}^d$ to $\mathbb{C}^d$ will reproduce the function values faithfully at each sample point for any frequency order (the DFT is bijective and even unitary, thus even allowing numerically accurate and stable reconstructions). Evaluating the obtained series in between sampling points, however, would reveal issues at discontinuities; as in most applications of DFT, our experiments do not perform such continuous evaluations.

An issues that does remain is aliasing: Nonlinearities in general and in particular discontinuous ones (even with higher order discontinuities) broaden the spectrum (this effect is particularly strong for first-order discontinuous functions such as ReLU). Therefore, the sampling frequency has to be chosen sufficiently high to capture these effects. Otherwise, the high-frequency harmonics will alias as lower-frequency signal components, which might lead to an underestimation of the blueshift effect.

B.2 Inputs to the Second Layer Nonlinearities are FBM-Noise

For the first layer of a ReLU network, the following calculation also supports the experimental findings of an $\mathcal{O}(1/k)$-power law from an analytical perspective. Let $p^{(1)}$ be a w-o path of the output of a given neuron from the first layer. It follows that:

$$p_i^{(1)}(t) = \text{ReLU}\left[\sum_j \left(w_{ij}^{(1)} + \alpha^{-1} t \cdot d_{ij}^{(1)}\right) \cdot x_j\right] \tag{11}$$

$$= \text{ReLU}\left[m \cdot t + n\right], \text{ for some } m, n \in \mathbb{R}. \tag{12}$$

This is a piecewise linear function that is constant zero on one interval and linear with slope m on another interval. With $a < b \in [0, 1]$ bounding the activation interval, the Fourier series becomes:

$$z_k^{(1,i)} = \int_a^b \exp\left(-2\pi i k \left(mt + n\right)\right) dt. \tag{13}$$

By the chain rule of derivations, this implies

$$|z_k^{(1,i)}| \in \mathcal{O}\left(\frac{1}{k}\right). \tag{14}$$

Correspondingly, the preactivations of the second layer will be random linear combinations of FBM-noise with $h = 1$, which again forms FBM-noise with that spectral variance (every Fourier coefficient is a sum of independent coefficients; therefore, variance is additive).

Similar findings where also previously shown in input space by Balduzzi et al. (2017), which, for the first layer, is just the dual to varying the weights (and therefore must yield the same result).

B.3 Blueshift and Depth

We now aim at formally understanding the blueshift effect in a multi-layer network. For simplicity, we consider a simple stack of linear and nonlinear layers with a fixed nonlinearity:

$$f(\mathbf{x}, \mathbf{W}) = \phi\Big(\mathbf{W}^{(L)}\phi\Big(\mathbf{W}^{(L-1)} \cdots \phi(\mathbf{W}^{(1)}\mathbf{x}) \cdots\Big)\Big). \tag{15}$$

We now consider a single output i of layer l:

$$f_i^{(l)}(\mathbf{x}, \mathbf{W}) = \phi\left(\mathbf{w}_i^{(l)} \cdot \phi\Big(\mathbf{W}^{(l-1)}\phi\Big(\mathbf{W}^{(l-2)} \cdots \phi(\mathbf{W}^{(1)}\mathbf{x}) \cdots\Big)\Big)\right). \tag{16}$$

Next, we consider a linear section $t \mapsto \mathbf{W} + t\mathbf{D}$ with $t \in [0, \alpha]$:

$$p_i^{(l)}(t) = \phi\left(\mathbf{w}_i^{(l)} \cdot \phi\Big((\mathbf{W}^{(l)} + t\mathbf{D}_l)\phi\Big((\mathbf{W}^{(l-1)} + t\mathbf{D}^{(l-1)}) \cdots \phi(\mathbf{W}^{(1)} + t\mathbf{D}^{(1)}\mathbf{x}) \cdots\Big)\Big)\right).$$

By replacing ϕ with finite polynomials (Equation 4), we obtain multi-variate polynomials as outputs of all of these function.

Varying parameters in a single layer: If we now select a layer k by setting all $\mathbf{D}^{(l)} = \mathbf{0}$ for all $i \neq k$, and only using the k-th matrix as direction vector, the parameter t will pass through $l - k + 1$ nonlinearities:

$$p_i^{(l)}(t) = \phi\left(\mathbf{w}_i^{(l)} \cdot \phi\Big(\mathbf{W}^{(l-1)} \cdots \phi\Big((\mathbf{W}^{(k)} + t\mathbf{D}^{(k)})q^{(k-1)} \cdots\Big) \cdots\Big)\right). \tag{17}$$

with a constant vector $q^{(k-1)}$.

Correspondingly, each nonlinearity will act on the spectrum of $q^{(k)}$ as a blueshift operator (forming the sum of repeated auto-convolutions of the spectrum, weighted by the polynomial coefficients). Therefore, we see that parameters of earlier layers (small layer index k, closer to the input layer) are blueshifted more frequently than later layers.

The equation also shows that the singular value spectrum of $W^{(l)}$ affects the results as well, in addition to the spectral shifts caused by the nonlinearities: the values rescale the input domain

(shrink/expand space wrt. parameter t), thereby changing the frequency reciprocal to the scale factor. However, a linear transformation scales the Fourier spectrum as a whole while nonlinearities spread the spectrum irreversibly by applying blueshifts (weighted sums of auto-convolutions of the spectrum). The magnitude of this nonlinear effect is, nontheless, dependent on the signal magnitude and thus affected by choices of $\mathbf{W}^{(l)}$.

Varying parameters in layer $0..l$: Similar arguments hold if we vary only parameters in the first l layers of the network. However, we obtain a mix of spectra that have been blueshifted (and scaled by weight matrices) a different number of times.

B.4 Spectral Broadening through Auto-Convolutions

The exact amount of broadening of a spectrum $z : \mathbb{Z} \to \mathbb{R}$ by an auto-convolution is non-trivial to quantify; we therefore resort to experiments for determining the effect in practice.

In special cases, we can analyze the effect:

Band limited functions: Assuming that z is bandlimited, specifically $z_k = z_{-k} = 0$ for all $k > K$, and $|z_k| = |z_{-k}| > 0$ for $k \leq K$ it is trivial to see that an auto-convolution extends the support by k entries to each side ($2k$ overall). This means, after j convolutions, we obtain a non-zero spectrum within $k = -jK...jK$. The proof is obtained by considering the left-most and right-most terms in the definition of the discrete convolution.

Central limits: If we assume that z is real and positive, and has finite second moments, the central limit theorem guarantees convergence towards a Gaussian function with standard deviation $\mathcal{O}\left(\sqrt{j}\right)$. The first assumption is obviously unrealistic in practice.

In the general case the j-th power of a Fourier series

$$\left[\sum_{k=-K}^{K} z_k \exp\left(2\pi i k t\right)\right]^j \tag{18}$$

expands to the sum

$$\sum_{k_1=-K}^{K} \cdots \sum_{k_j=-K}^{K} z_{k_1} \cdots z_{k_j} \exp\left(2\pi i (k_1 + k_2 + \cdots + k_j)\right). \tag{19}$$

If we only count the number of terms of the same frequency $k = k_1 + \cdots + k_j$, the number of occurrences of each frequency k will tend towards a normal distribution with variance j. However, as the complex coefficients can cancel out, the limit distribution is only an upper bound.

B.5 Frequency Dependence in Averaging of Multiple Computation Paths

We observe that averaging functions after nonlinearities leads to some smoothing of the result by decreasing the correlation between higher-frequency Fourier coefficients. In order to understand the effect, we look at a simple 1D model case: We consider a the composite functions

$$f_{post}(x) = \phi\big(\underbrace{w_1 \cdot g(x)}_{=:h'_1} + \underbrace{w_2 \cdot g(x)}_{=:h'_2}\big) \tag{20}$$

and

$$f_{pre}(x) = \underbrace{\phi(w_1 \cdot g(x))}_{=:h_1} + \underbrace{\phi(w_2 \cdot g(x))}_{=:h_2} \tag{21}$$

where w_1, w_2 are random numbers, drawn from the same normal distribution.

Let z_k be the sequence of Fourier coefficients of g, and u_k that of the result. In the first case, f_{post}, a random linear combintation of the input spectra is performed, which is then blueshifted:

$$u = \sum_{j=1}^{K} \bigotimes_{q=1}^{j} \left(w_1 z + w_2 z\right) \tag{22}$$

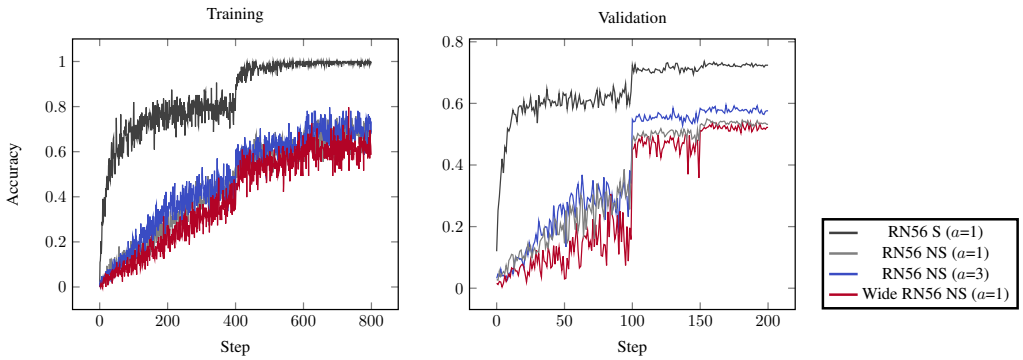


Figure 10: Repeating the experiments of Figure 8 for a single run on the Cifar100 dataset.

In the second case,

$$u = \sum_{j=1}^{K} \bigotimes_{q=1}^{j} (w_1 z) + \sum_{j=1}^{K} \bigotimes_{q=1}^{j} (w_2 z), \tag{23}$$

we first blueshift the randomly-scale spectra and then form the sum. This creates a frequency dependence because larger powers (larger values of j, corresponding to more repeated convolutions, and thus j-fold products of input Fourier coefficients) behave more nonlinearly and thus react more strongly to the weight scaling: While a linearly weighted average will be linearly correlated with the original, averages of nonlinearly transformed functions will loose linear correlation (and thus appear more random to a simple averaging operation). The harmonic distortion analysis that expresses the powers as auto-convolutions of the spectrum shows that higher-frequency components (created by the blueshift) are also the ones that behave more nonlinearly.

C Training on Cifar100

We repeat the experiment on averaging-networks for the Cifar100 dataset, holding out 1% of the training data for validation. We leave all hyperparameters untouched. In Figure 10, we still see the advantage of the averaging-network over the regular network, especially in validation accuracy.

D Other spectral shift figures

D.1 Spectral shift during training

To show that the blueshift effect is not limited to the initialization, we show the spectral shift of a Toy-CNN 50 at 0 and 20 epochs of training in Figure 11. ResNets still do not show any major blueshift at any time. The ReLU and TanH architectures without skip-connections show less show blueshift after 20 epochs of training, indicating that blueshift is more of a problem in early phases of training (Frankle et al., 2020). When increasing the sampling radius to $\alpha = 10$, blueshift is clearly visible again for ReLU and TanH activations in the networks without skip-connections.

D.2 Spectral shift with delta-orthogonal initialization

The experiments of Figure 3 with batch normalization are subject to exploding gradients. Xiao et al. (2018) describe an initialization scheme that fixes exploding gradients for TanH activations. Fig 12 shows that for a very small region near the orthogonal initialization point, blueshift vanishes. As soon as we walk further from the initialization point, blueshift becomes visible again.

D.3 Spectral shift in gradient direction

The subspace of the loss surface where gradient descent operates has been studied (Gur-Ari et al., 2018) and attributed a specific behavior. We want to see how blueshift behaves when slicing the

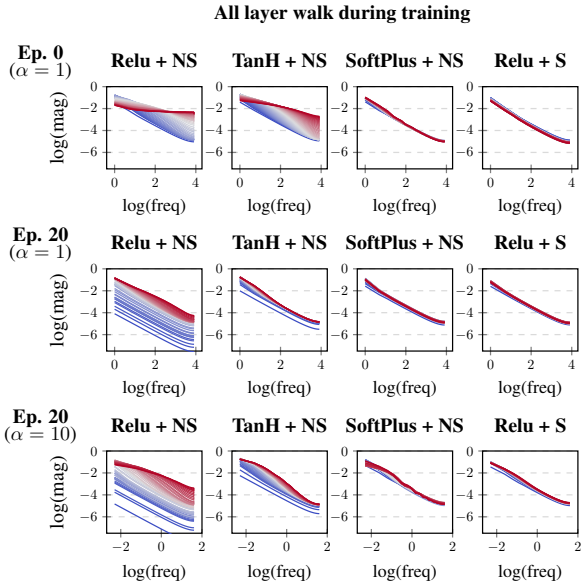


Figure 11: Repeating the experiments of Figure 3 (all layers, unscaled) during training.

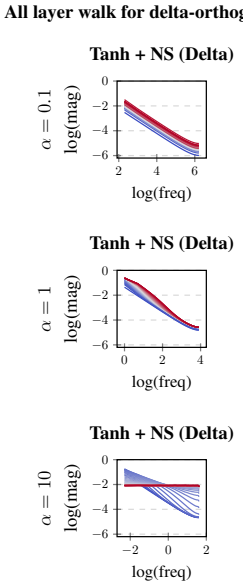


Figure 12: Repeating the experiments of Figure 3 (all layers, unscaled) for TanH activations with delta-orthogonal initialization for different path lengths α.

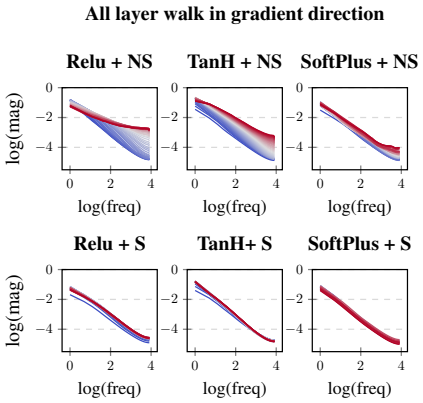


Figure 13: Repeating the experiments of Figure 3 (all layers, unscaled) at initialization in gradient direction.

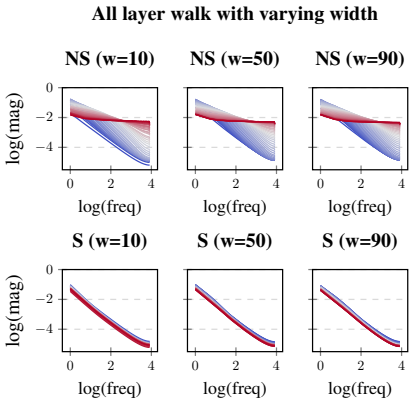


Figure 14: Repeating the experiments of Figure 3 (all layers, unscaled) for variable network width.

network in the direction of gradient descent. For this, we repeat the experiments of Figure 3, but compute the gradient at initialization, normalize it and use it as direction $\mathbf{D}$ of equation 1. In Figure 13, we see that the effect is still visible although dampened for ReLU and tanh activations. The effect seems a little more visible for softplus activations.

D.4 VARYING WIDTH

To see if layer width has any influence on blueshift, we repeat the experiment of Figure 3 for a Toy-CNN 50 with number of filters w. We can see on Figure 14 that layer width doesn't seem to have a notable impact on blueshift strength.

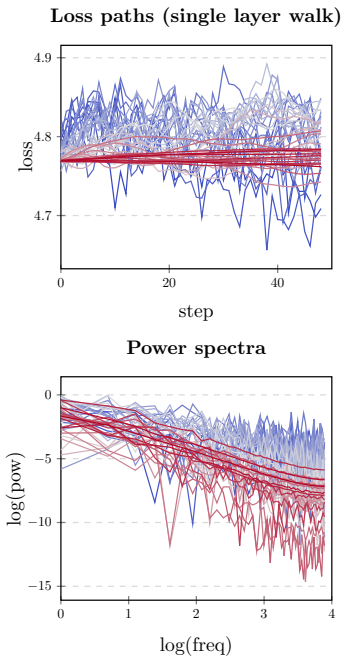


Figure 15: Resulting loss paths and respective path-normalized power spectra when varying parameters in only one layer of a Toy-CNN 50 at initialization. The color of a path indicates in which layer parameters were varied (red is deeper).

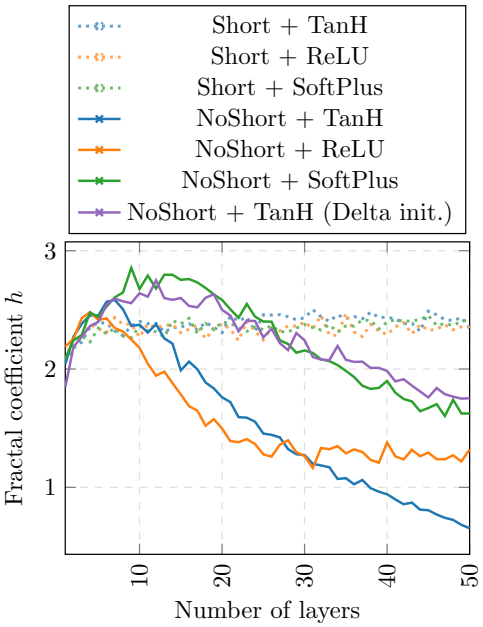


Figure 16: Smoothness of the loss surface at initialization for a Toy-CNN of varying depth with and without shortcuts on MNIST for a batch size of 256.

E WALK PER LAYER

To see the contribution of each layer to a loss path, we sample a random direction $\mathbf{D}$ and measure the resulting loss path when only modifying weights in a single layer. We realize this for a Toy-CNN with 50 layers at initialization in Figure 15. We see that varying weights in higher layers (red) results in smooth paths whereas varying weights in lower layers (blue) results in rougher paths. In the normalized power-spectra, we can see the increased blueshift of paths resulting from varying earlier layers.

F SMOOTHNESS MEASUREMENTS ON MNIST

To show that our smoothness measurements are mostly independent of batch size and dataset, we revisit Figure 5. This time, we use the MNIST dataset and a batch size of 256. All other hyperparameters are maintained. We see that although the absolute values slightly differ, the relative behavior of the nonlinearities is qualitatively the same. The results are shown in Figure 16

F.1 MORE ACTIVATION FUNCTIONS

To demonstrate that blueshift occurs with every nonlinear activation function, we repeat the experiment of Figure 3 for exponential linear units (ELU), gaussian error linear units (GELU), hard tanh (HTANH), leaky relu (LReLU), scaled exponential linear unit (SELU) and sigmoid activations.

In order to demonstrate the that spectral spread depends on the magnitude of higher-order polynomial coefficients, we show the plot of the magnitude of the coefficients of a polynomial approximation with Chebyshev nodes of degree 25 in Figure 19. The fit is performed for the interval $[-5, 5]$ (slightly different from Figure 2 in the main paper, which shows a fit for [-1,1]). Figure 19(a) also shows again

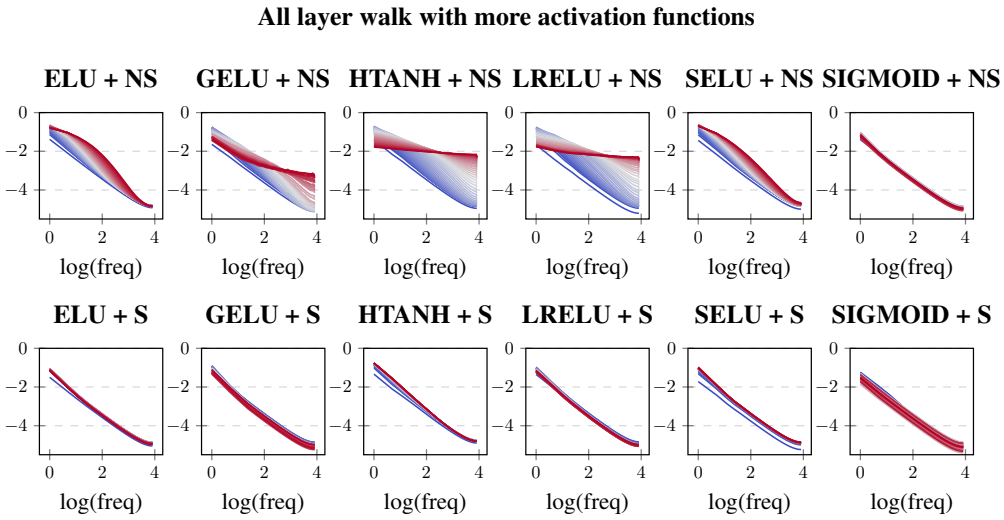


Figure 17: Repeating the experiments of Figure 3 (all layers, unscaled) for more nonlinearities.

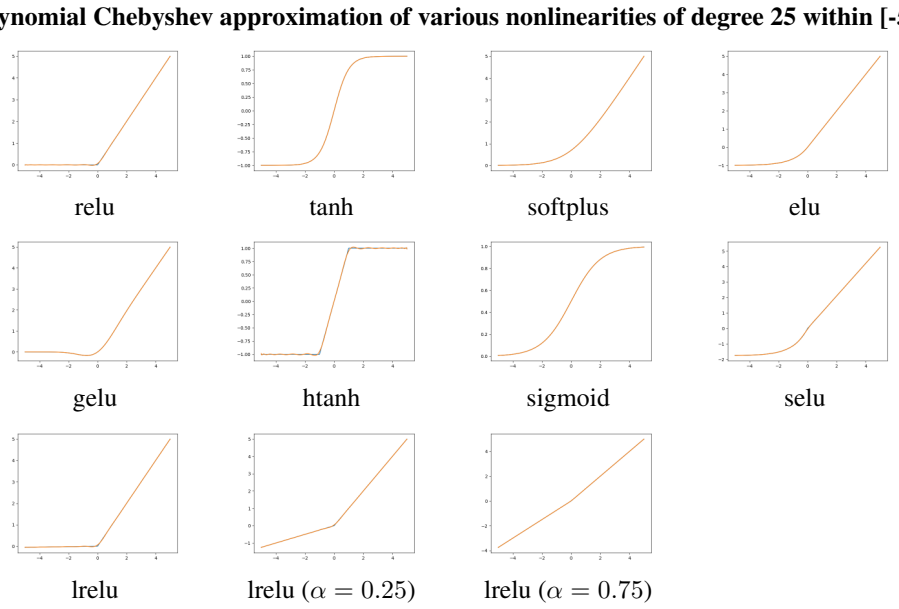
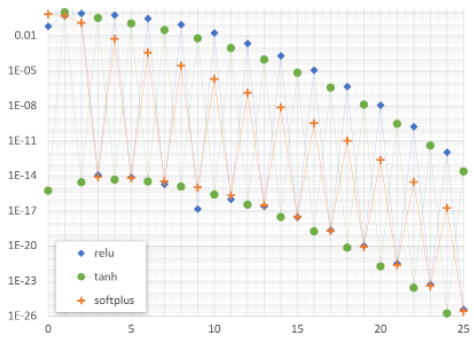


Figure 18: Various Nonlinearities with their Chebyshev-Polynomial Approximation within the interval $[-5, 5]$.

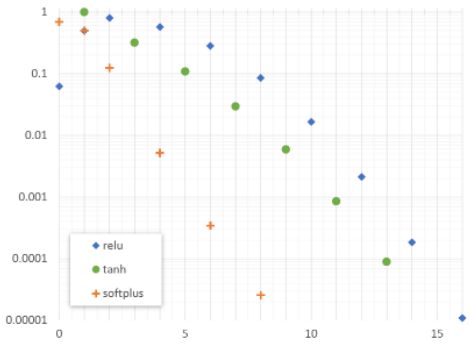
the comparison between ReLU, $\tanh$, and $\mathrm{softplus}$, uniformely with this Chebyshef approximation; this shows the differences more clearly (please also note the logarithmic scale of the y-Axis).

The approximations are quite tight within the interval, see Figure 18 for reference.

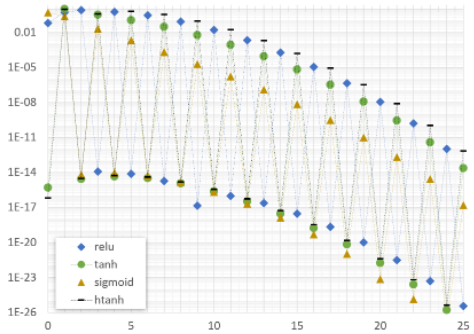
Comparison of the magnitude of the polynomial coefficients $|a_j|$
(for the degree 25 approximation of Figure 18)



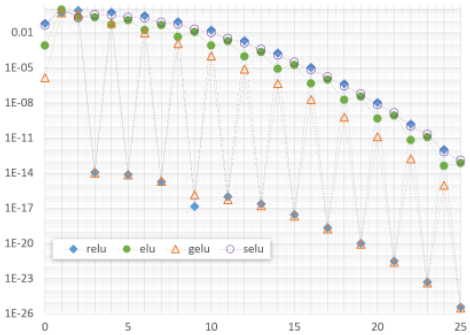
(a) comparison of relu, tanh, softplus



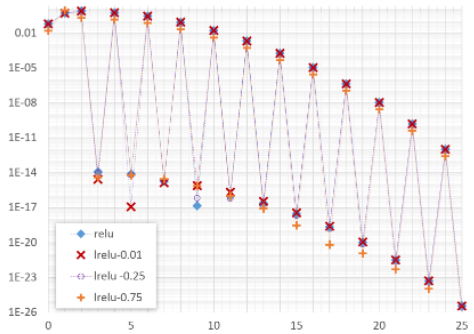
(b) close-up of (a)



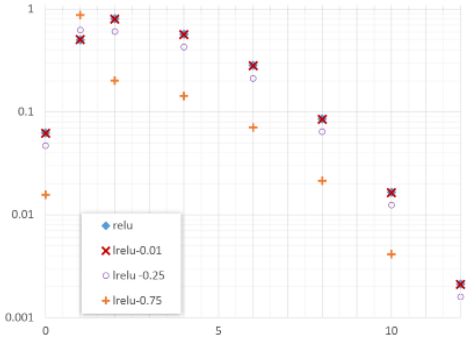
(c) comparison of relu,tanh,sigmoid,hardtanh (relu for reference)



(d) comparison of relu, elu, gelu, selu



(d) different variants of leaky-relu (default is $\alpha = 1E-2$)



(e) closeup of (d)

Figure 19: Absolute value of the polynomial coeffients of the Chebyshev polynomial approximation. All fits are done within the interval $[-5, 5]$. Note: Different interval from Fig. 2 in the main paper! (to capture the shape of all nonlinearities well).

	1	t	t^2	t^3	t^4	t^5	t^6
relu	6.2512213E-02	5.0000000E-01	8.0859691E-01	1.1924015E-14	-5.7216603E-01	-7.6820090E-15	2.8328407E-01
tanh	5.1805002E-16	9.9793926E-01	-2.7609686E-15	-3.2070438E-01	4.6215225E-15	1.0881659E-01	-3.2234542E-15
softplus	6.9314722E-01	5.0000000E-01	1.2499948E-01	-8.0003321E-15	-5.2070252E-03	6.6261234E-15	3.4591520E-04
elu	7.9148528E-04	9.4044077E-01	2.1895988E-01	-2.0584738E-01	-4.8542686E-02	1.1656771E-01	1.6778199E-02
gelu	1.4022057E-06	5.0000000E-01	3.9892182E-01	9.9860124E-15	-6.6440646E-02	-7.1476355E-15	9.9256757E-03
htanh	-6.4522701E-17	9.5248313E-01	3.1201720E-15	3.9176956E-01	-5.5233592E-15	-6.5558052E-01	4.0680108E-15
lrelu	6.1887091E-02	5.0500000E-01	8.0051094E-01	2.6432027E-15	-5.6644437E-01	1.2012168E-17	2.8045123E-01
selu	-4.2829527E-02	1.2996891E+00	-1.8704691E-01	-3.6190014E-01	3.1940644E-01	2.0493761E-01	-1.7089694E-01
sigmoid	5.0000000E-01	2.4999950E-01	-6.5774789E-15	-2.0830617E-02	9.6140255E-15	2.0789179E-03	-5.0960174E-15
lrelu -0.25	4.6884160E-02	6.2500000E-01	6.0644768E-01	9.4213822E-15	-4.2912452E-01	-6.5801233E-15	2.1246305E-01
lrelu -0.75	1.5628053E-02	8.7500000E-01	2.0214923E-01	4.0171812E-15	-1.4304151E-01	-5.9161867E-15	7.0821018E-02

	t^7	t^8	t^9	t^{10}	t^{11}	t^{12}	t^{13}
relu	1.8094049E-15	-8.5698408E-02	1.4490894E-17	1.6559778E-02	-9.7673166E-17	-2.1229556E-03	2.3344256E-17
tanh	-2.9502585E-02	1.1578462E-15	5.9214502E-03	-2.4554185E-16	-8.5681435E-04	3.3129449E-17	8.8697276E-05
softplus	-3.2764837E-15	-2.5656019E-05	1.0262503E-15	1.9069649E-06	-2.1426831E-16	-1.2924529E-07	3.0466611E-17
elu	-4.5051680E-02	-4.3845220E-03	1.1333777E-02	7.8485900E-04	-1.9013645E-03	-9.6036723E-05	2.1786860E-04
gelu	2.0301054E-15	-1.1630904E-03	-1.4744780E-16	1.0800433E-04	-5.4350867E-17	-7.9444190E-06	1.6680823E-17
htanh	3.3738173E-01	-1.5872645E-15	-9.5223816E-02	3.6951250E-16	1.6867158E-02	-5.4818441E-17	-1.9874811E-03
lrelu	-1.3304595E-15	-8.4841423E-02	7.6609561E-16	1.6394181E-02	-2.1268001E-16	-2.1017260E-03	3.5039736E-17
selu	-7.9205328E-02	5.2914487E-02	1.9925905E-02	-1.0334500E-02	-3.3427877E-03	1.3329332E-03	3.8303464E-04
sigmoid	-2.0739824E-04	1.3731145E-15	1.9820840E-05	-2.1665218E-16	-1.7123664E-06	2.1314190E-17	1.2507122E-07
lrelu -0.25	1.6962453E-15	-6.4273806E-02	-6.5505775E-17	1.2419834E-02	-6.1909935E-17	-1.5922167E-03	1.6274033E-17
lrelu -0.75	2.8720156E-15	-2.1424602E-02	-7.1728555E-16	4.1399446E-03	1.0211438E-16	-5.3073889E-04	-8.1962396E-18

	t^{14}	t^{15}	t^{16}	t^{17}	t^{18}	t^{19}	t^{20}
relu	1.8443579E-04	-2.8820282E-18	-1.0903101E-05	2.1693327E-19	4.3182633E-07	-1.0312989E-20	-1.0961345E-08
tanh	-2.9550924E-18	-6.5355890E-06	1.7658317E-19	3.3887465E-07	-6.9979237E-21	-1.2057256E-08	1.7656046E-22
softplus	7.3223066E-09	-2.9626728E-18	-3.2257985E-10	1.9577707E-19	1.0362969E-11	-8.6141997E-21	-2.2542826E-13
elu	8.0862081E-06	-1.7269223E-05	-4.6740594E-07	9.4564265E-07	1.8201823E-08	-3.5094476E-08	-4.5598427E-10
gelu	4.5500348E-07	-2.2272244E-18	-1.9681332E-08	1.7448146E-19	6.1548299E-10	-8.5046796E-21	-1.3032819E-11
htanh	5.3651152E-18	1.5990800E-04	-3.5046880E-19	-8.8268590E-06	1.5117444E-20	3.2893167E-07	-4.1329037E-22
lrelu	1.8259143E-04	-3.6811891E-18	-1.0794070E-05	2.5320591E-19	4.2750807E-07	-1.1364747E-20	-1.0851732E-08
selu	-1.1625322E-04	-3.0361009E-05	6.8910899E-06	1.6625337E-06	-2.7347262E-07	-6.1699575E-08	6.9523720E-09
sigmoid	-1.3305631E-18	-7.2716001E-09	5.1348407E-20	3.1974208E-10	-1.0979379E-21	-1.0107329E-11	7.6307576E-24
lrelu -0.25	1.3832684E-04	-2.0580252E-18	-8.1773260E-06	1.5611204E-19	3.2386975E-07	-7.4361410E-21	-8.2210090E-09
lrelu -0.75	4.6108947E-05	2.9304331E-19	-2.7257753E-06	6.3999589E-21	1.0795658E-07	-1.1416736E-21	-2.7403363E-09

	t^{21}	t^{22}	t^{23}	t^{24}	t^{25}
relu	3.0296993E-22	1.6113608E-10	-5.0295283E-24	-1.0429164E-12	3.6122193E-26
tanh	2.7986355E-10	-2.5687606E-24	-3.8131280E-12	1.6410322E-26	2.3116680E-14
softplus	2.4105425E-22	2.9464505E-15	-3.8765757E-24	-1.7393697E-17	2.7255018E-26
elu	8.4227758E-10	6.6327810E-12	-1.1791444E-11	-4.2559630E-14	7.3104430E-14
gelu	2.5429110E-22	1.6629623E-13	-4.2790160E-24	-9.6171035E-16	3.1074839E-26
htanh	-7.9089452E-09	6.4866408E-24	1.1077249E-10	-4.4517362E-26	-6.8651287E-13
lrelu	3.2080448E-22	1.5952472E-10	-5.1716715E-24	-1.0324872E-12	3.6319847E-26
selu	1.4808077E-09	-1.0232631E-10	-2.0730531E-11	6.6293325E-13	1.2852485E-13
sigmoid	2.1491988E-13	1.3841516E-25	-2.7389162E-15	-2.2476903E-27	1.5757713E-17
lrelu -0.25	2.1839093E-22	1.2085206E-10	-3.6215891E-24	-7.8218726E-13	2.5980741E-26
lrelu -0.75	5.0212219E-23	4.0284020E-11	-1.0326312E-24	-2.6072909E-13	8.4909511E-27

Table 1: Numerical values for the polynomial coefficients used in Figure 18 and 19.

7.2 Nonlinear Advantage: Trained Networks Might Not Be As Complex as You Think

Published at ICML 2023. **Co-Author's Contributions:** Jan Disselhoff helped me in general discussions about the subject and in proof-reading the paper.

Nonlinear Advantage: Trained Networks Might Not Be As Complex as You Think

Christian H.X. Ali Mehmeti-Göpel [1] **Jan Disselhoff** [1]

Abstract

We perform an empirical study of the behaviour of deep networks when fully linearizing some of its feature channels through a sparsity prior on the overall number of nonlinear units in the network. In experiments on image classification and machine translation tasks, we investigate how much we can simplify the network function towards linearity before performance collapses. First, we observe a significant performance gap when reducing nonlinearity in the network function *early* on as opposed to *late* in training, in-line with recent observations on the time-evolution of the data-dependent NTK. Second, we find that after training, we are able to linearize a significant number of nonlinear units while maintaining a high performance, indicating that much of a network's expressivity remains unused but helps gradient descent in early stages of training. To characterize the depth of the resulting partially linearized network, we introduce a measure called average path length, representing the average number of active nonlinearities encountered along a path in the network graph. Under sparsity pressure, we find that the remaining nonlinear units organize into distinct structures, forming core-networks of near constant effective depth and width, which in turn depend on task difficulty.

1. Introduction

Deep learning as such is based on the idea that concatenations of (suitably chosen) nonlinear functions increase expressivity so that complex pattern modeling and recognition problems can be solved. While initial approaches such as AlexNet (Krizhevsky et al., 2012) only used moderate depth, improvements such as batch normalization (Ioffe & Szegedy, 2015; Santurkar et al., 2018; Arora et al., 2019) or residual connections (He et al., 2015b) made the training of networks with hundreds or even thousands of layers possible, and this did contribute to significant practical gains.

From a theoretical perspective, a network's depth (along with its width) upper bounds the complexity of the function that it can represent (Bartlett et al., 2019) and therefore upper bounds the network's expressivity. Indeed, deeper networks tend to enhance performance (Tan & Le, 2019), but gains seem to taper off and saturate with increasing depth. Very deep networks are also known to be more difficult to analyze (Allen-Zhu et al., 2019), more computationally expensive to train or infer and suffer from numerous stability issues such as vanishing (Hochreiter, 1991), exploding (Zhang et al., 2019) and shattering (Balduzzi et al., 2017) gradients. Similar arguments can be made about layer width: a sufficiently large network can memorize any given function (Cybenko, 1989), but under standard initialization and training infinite width networks degrade to Gaussian processes (Jacot et al., 2018).

From a practical perspective, there are now many successful recipes for creating networks of a prescribed depth, but it is still difficult to understand – empirically or analytically – how many nonlinear layers and features per layer are actually needed to solve a problem, and how effective a chosen architecture actually is in exploiting its expressive potential in the sense of a deep stack of concatenated nonlinear computations.

Our paper addresses this question from an empirical perspective: we use a simple setup that associates every nonlinear unit with a cost at channel granularity, which can be raised continuously, while simultaneously trying to maintain performance. As PReLU activations (He et al., 2015a) can be used to interpolate continuously between a ReLU function and a linear function (Ali Mehmeti-Göpel et al., 2021), we replace each ReLU layer with channel-wise PReLU activations and regularize their slopes towards linearity. Such a linearized feature channel therefore only forms linear combinations of existing feature channels, thereby not effectively contributing to the nonlinear complexity of the network. To measure the nonlinear complexity or "effective depth"

[1]Department of Computer Science, University of Mainz, Germany. Correspondence to: Christian H.X. Ali Mehmeti-Göpel <chalimeh@uni-mainz.de>.

Proceedings of the 40^{th} International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023.

of the resulting partially linearized networks, we introduce a metric called **average path length (APL)**: the average amount of nonlinear units a given input traverses until it passes the final layer. We thereby disregard subsequent linear mappings along computational paths, as these do not increase expressivity in a nonlinear sense.

Using this tool, we make a series of experiments on common convolutional and transformer architectures on standard computer vision and machine translation tasks by partially linearizing networks with regard to the network's inputs at different stages of training and find that networks linearized later on in training obtain a significantly higher performance than networks linearized earlier on in training. This is non-trivial, since the network's expressivity is the same whether it is linearized early or late in training. We find the biggest differences in the early training phase, complementarily to the findings of (Fort et al., 2020) that establish a similar, but much less surprising effect when fully linearizing the network with regard to the network's weights.

Analyzing these partially linearized networks extracted after training, we find that we can extract very shallow networks with a surprisingly high performance for their effective depth. These findings are consistent with the lottery ticket hypothesis (Frankle & Carbin, 2019) that there is a core nonlinear structure in networks, and with the subsequent findings of You et al. (2020) that it forms within the first epochs of training. Our method allows us to compute an approximate lower bound for depth and width that the networks needs to solve a task before performance collapses and we find that these are approximately constant for a given task and regularization strength, independently of the width and depth of the initial network. We also find that the effective depth of this core nonlinear structure grows with problem complexity for a fixed regularization strength.

2. Related Work and Contributions

Different approaches to network pruning were explored in recent years: magnitude-based weight pruning, weight-regularization techniques, sensitivity-based pruning and search-based approaches (Neill, 2020). Frankle & Carbin (2019) extract a highly performant, sparse and re-trainable subnetwork by removing all low-magnitude weights after a given training time, re-initializing the network and iterating this process. This motivates the "lottery ticket hypothesis" of a network consisting of a smaller core structure embedded in the larger, overparametrized and redundant network, which, in their case, can be extracted by weight pruning. Our paper prunes nonlinear units instead of weights, but comes to similar findings of a problem-difficulty-dependent minimal set, embedded in a much larger and deeper network, when considering nested nonlinear computations. You et al. (2020) claim that the final accuracy of lottery tickets drawn after at early training is already drastically higher than at initialization. Su et al. (2020) conduct "sanity checks" on the lottery ticket hypothesis and conclude that only the number of remaining weights matters for a given dataset. We conduct similar checks and find that transferring simple statistics such as how many nonlinear units are active per layer are not sufficient to recover full performance.

Simplification of networks by reducing nonlinearity has become a major area of interest. A lot of recent work has studied the neural tangent kernel (NTK) approximation, which linearizes the network function with regard to its parameters. It arises in the infinite width limit (under mild conditions) or by explicitly performing a linear Taylor-approximation of a finite network (Jacot et al., 2018; Fort et al., 2020). As it fully linearizes training, the NTK has been tremendously useful for gaining a better understanding of the training of deep network, such explaining double-descent generalization (Belkin et al., 2019; Wilson & Izmailov, 2020). Maybe unsurprisingly, linearized training hurts performance in practice (Fort et al., 2020) and theory: Roberts et al. (2022) attribute it to the loss of detection of higher-order moments in the data distribution). Fort et al. have coined the term *nonlinear advantage* for the observed discrepancy in performance between the network's NTK and the nonlinear network function that vanishes over time when training with low learning rate. Within the NTK framework, the impact of ReLUs can be captured by path kernels (Lakshminarayanan & Singh, 2020), the learning of which improves results and generalizes when retraining, and can be used to understand pruning methods at initialization (Gebhart et al., 2021). Our APL measures are tightly related to the proposed (gated) path-integral formulation there. Our paper simplifies the network function itself by reducing the number of nonlinear units in the network and therefore partially linearizing it in both inputs and weights, finding a similar, but difficulty-dependent nonlinear advantage.

Dror et al. (2021) use a methodology similar to ours, but applied layer-wise and aiming at improved performance characteristics at inference time. Our channel-wise approach allows us to reduce significantly more nonlinear units in the network whilst maintaining a similar performance as well as characterize the "effective width" of the emerging core network. A training time dependent effect as we show it, is not studied by the authors.

3. Reducing Nonlinear Feature Channels

In order to reduce the amount of nonlinear feature channels in a network, we take network architectures and replace their ReLU activations with PReLUs. We then use a single PReLU weight for every channel and add a sparsity regularization of $L_{0.5} = \sum |1 - \alpha_i|^{0.5}$ to the regular training loss scaled with a *regularization weight* ω, where α_i

is the variable slope of the i-th PReLU. We chose channel-wise PReLU units because the latter allows a much bigger reduction in overall nonlinearity compared to layer-wise units, and pixel-wise PReLUs would entail an unreasonable amount of additional parameters. Since the regularization loss term is discontinuous at 1, we disable a PReLU unit if their slope gets close enough to one. We call such a unit *inactive*, while all other units are *active*.

By regularizing PReLU units this way, the slope of inactive units is locked to $\alpha_i = 1$, but the slope of active units is an arbitrary number between 0 and 1. After reaching a goal percentage of disabled PReLU units, it is possible to regularize the slope of the remaining active units back to 0, effectively transforming the network back to a regular ReLU network and relating to (Hanin & Rolnick, 2019), but we found that this method can be hurtful to performance and therefore refrain from using it.

3.1. Average Path Length (APL)

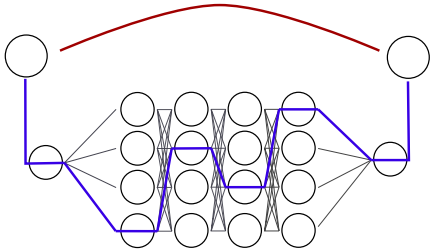


Figure 1. In this Figure, a residual connection (red) skips 4 small fully connected layers. The left-most and right-most nodes are connected by 257 different paths. Using *unnormalized* average path length implies a $1/257$ chance of selecting the red path and the same chance of selecting the blue path. Using *normalized* average path length, we have a $1/2$ chance of selecting the red path and a $1/512$ chance of selecting the blue path.

The depth of a network, according to its traditional notion, corresponds to the *maximum* amount of nonlinear units encountered when following the computation graph of a network from input to output. After partially linearizing such a network, this value remains the same, assuming at least one channel per layer remains active (non-linear). Therefore, computing the *average* amount of nonlinear units instead of the maximum seems like a more sensible characterization for partially linearized networks.

Let $G = (V, E)$ be the directed acyclic graph that represents the computation graph of a given feedforward neural network. Since we are only interested in the nonlinear structure of the graph, a node in the graph corresponds to a PReLU unit in the network. We denominate $V_d \subset V$ the subset of nodes that correspond to the d-th layer in the network. Let $v_{in}, v_{out} \in V$ be the respective input and output vertices of the graph. Let $R \subset V$ be a subset of the vertices that represent the blocks containing an active PReLU i.e. $|\alpha - 1| > \epsilon$, where α is the weight of the PReLU.

Let $\mathcal{P}^{(n)} \subset V^n$ be the set of all paths of length n in G that originate in v_{in}, i.e. $p_1 \in v_{in}$ and $(v_i, v_{i+1}) \in E$ for all $1 \leq i \leq n-1$. We define the *effective path length* of a path $p \in \mathcal{P}^{(n)}$ as the number of active PReLU activations it traverses:

$$\phi(p) := |\{v_i \in p \mid v_i \in R,\ 1 \leq i \leq n\}|,$$

which is always smaller or equal to its regular length $|p|$. Let $v \in V$ be a vertex of the graph, we then define its path histogram function as the number of paths from v_{in} to v of effective length l :

$$\phi^{(l)}(v) := |\{p \in \mathcal{P}^{(l')} \mid p_l = v,\ \phi(p) = l,\ 1 \leq l' \leq l\}|.$$

We finally define the *average path length (APL)* of the network as:

$$APL(G) := \frac{\sum_{i=0}^{d} i \cdot \phi^{(i)}(v_{out})}{\sum_{i=0}^{d} \phi^{(i)}(v_{out})},$$

where d is the depth of the network. In order to effectively compute the APL of a network, we resort to dynamic programming.

Proposition 3.1. *Let $v \in V$ be a vertex in the network. We can then compute its path histogram function by summing over all vertices that have an outgoing edge to v:*

$$\phi^{(l)}(v) = \begin{cases} \sum_{v' \in V} \mathbb{1}_{(v',v) \in E} \cdot \phi^{(l-1)}(v') & \text{if } v \in R, \\ \sum_{v' \in V} \mathbb{1}_{(v',v) \in E} \cdot \phi^{(l)}(v') & \text{otherwise.} \end{cases}$$

Proof. Assume that we have the full histogram of all vertices of layer $l-1$ and lower and want to calculate the histogram of a given vertex $v \in V$ in layer l. Let $V' := \{v' \in V | (v', v) \in E\}$ be the set of all vertices that have an edge to v. Since G is a DAG, all paths from v_{in} to v must go through exactly one node of V'. The histogram of v can therefore be decomposed as shown above, shifting if v contains an active PReLU. □

Implementation details: We implement this recursion in a modified forward pass through the network by re-using the batch dimension of the input tensor as "histogram dimension" that saves the path histogram function $\phi^{(l)}(v)$ for a given neuron v. By setting all weights of a linear (fully connected or convolutional) layer to one and biases to zero, executing the layer then automatically outputs for each neuron the sum of all inputs and therefore sums the path histogram functions of all incoming nodes. We then just need to "shift" the obtained histogram if an active PReLU is present to obtain the correct histogram function for v. Other layers such as batch normalization or pooling layers

are ignored. We finally use a constant $(1, 0, \ldots, 0)$ input for network and extract the obtained histograms in the last layer. For reasons discussed below, it can be useful to normalize the histograms before adding them inside a ResBlock; we call the resulting value the *normalized average path length*. An illustrative example for a histogram computation of a non-residual and a residual network is shown in the Appendix in Figure 12.

By Proposition 1, the *unnormalized average path length* (APL) describes the expected number of active PReLU units a path contains if we draw a path *uniformly from the set of all possible paths*. In networks with residual connections, this heavily favors longer paths, as every additional layer used increases the number of possible paths exponentially (ref. Appendix Figure 13). In this measure, despite residual connections, the initial path length of a ResNet is only slightly lower than its depth which might seem unintuitive.

The *normalized average path length* (NAPL) describes, as illustrated in Figure 1, the expected number of active PReLUs a path contains if we follow a random outgoing edge at every node in the path. In a residual network this means that inside a ResBlock, both summands (main branch and residual connection) have equal weight.

Both APL and NAPL do not depend on the absolute number of active PReLUs in a layer but rather on their relative proportion. For this reason, we will also use the simple measure of *effective network width* or ENW of a network. It is the absolute number of active PReLUs per layer, averaged over all layers. This measure depends only on the extracted "core" network and is therefore useful for comparing architectures of different width.

4. Experiments

In this section, we apply linearization to network architectures at different stages of training to show the existence of a discrepancy in performance between networks partially linearized earlier and later in training that we call **nonlinear advantage**. Once that we established this effect, we observe the performance and shape of the networks resulting when linearizing after training to convergence.

As our techniques requires networks with ReLU activations, we chose a ResNet (He et al., 2016), PyramidNet (Han et al., 2017) with ("Short") and without ("NoShort") residual connections as well as Transformer (Vaswani et al., 2017) as examples of standard architectures. For computer vision tasks, we work with standard image classification datasets of variable but well-known difficulty: CIFAR-10, CIFAR-100 (Krizhevsky, 2009), CINIC-10 (Darlow et al., 2018), Tiny ImageNet (Le & Yang, 2015) and ILSVRC 2012 (called ImageNet in the following) (Deng et al., 2009). As for NLP tasks, we use the Multi30k (Elliott et al., 2016) machine translation task (german to english). We train all networks from scratch except on ImageNet where we use a pre-trained ResNet50 from the Torchvision library for linearization.

In the experiments of Figure 3 and 23, we switch on our linearizing regularizer described in Section 3 at the indicated time during training, in order to capture a time-dependent effect. In all other experiments, the partial linearization happens in a separate phase, after conventionally training the network:

Training Phase: We conventionally train the network with the most basic setup that is capable of delivering benchmark results for the chosen architectures: ReLU units, momentum SGD, a multistep learning-rate scheduler and weight decay.

Linearization Phase: The ReLU units are replaced by regularized PReLUs (with initial negative slope 0) and we resume training in a shorter post-training step. PReLUs are considered inactive and frozen if their slope is higher than 0.99 (1% margin). Concerning learning-rate scheduling in the post-training step, we need a big learning rate initially to reach the target nonlinearity and a lower learning rate afterwards in order to reach a good performance. We therefore revert to the initial learning rate and use the same multistep scheduling as in the regular training phase adapted to the shorter post-training phase.

Further details about architectures and training regimes used can be found in the Appendix at Section C. We decided to use the normalized average path to avoid overflows for deeper networks (the absolute number of paths through the network grows exponentially in depth) and because it is inline with previous works discussing path lengths in ResNets (Veit et al., 2016). Results with unnormalized path length yield similar results albeit the absolute numbers are higher as shown in the Appendix at Section B.

4.1. The Nonlinear Advantage

In this section, we want to establish the existence of a **nonlinear advantage** by we comparing the final performance of a network that is linearized at different stages of training. We carefully choose our experimental setup such that the difference in performance can be purely attributed to the difference in nonlinear units and not to other factors such as training time, learning rates or architectural differences.

4.1.1. Re-training Partially Linearized Networks From Scratch

In a first step, we consider the most extreme case of comparing a network partially linearized *after being fully trained* to a network of the same architecture that contains the same amount of nonlinear units trained *from scratch*. We want to see whether we can train a network containing the same amount of nonlinear units as the extracted network to the

Architecture	Base	Linear.	Exact	Layerwise P.	Global P.
CIFAR-10					
ResNet56S	92.7	89.2	89.0 ± 0.0013	90.0 ± 0.0021	89.7 ± 0.0027
ResNet56NS	84.8	81.3	81.8 ± 0.0033	81.8 ± 0.0065	74.0 ± 0.0063
PyramNet110S	94.7	91.5	91.4 ± 0.0003	91.7 ± 0.0012	91.6 ± 0.0014
CIFAR-100					
ResNet56S	69.9	68.3	66.8 ± 0.0042	66.7 ± 0.0134	65.2 ± 0.0044
ResNet56NS	56.7	54.8	45.3 ± 0.0110	45.8 ± 0.0263	47.0 ± 0.0218

Figure 2. Test accuracy of the extracted partially linearized network compared to baseline networks where we re-train the same network from scratch with the same amount of inactive PReLUs but differently distributed. Standard deviation over five runs is indicated for the re-trained networks.

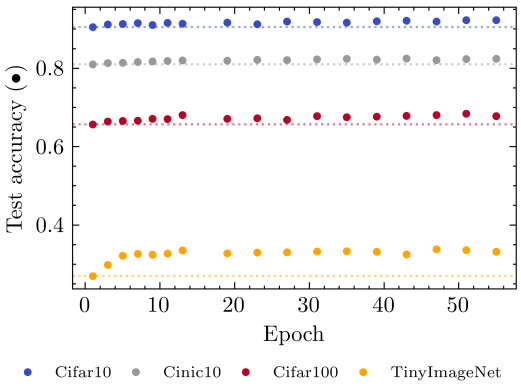


Figure 3. Test accuracy of networks partially linearized at different epochs during training on different datasets. The dotted lines indicate the height of the first data point for visual reference.

performance of the latter and whether transferring simple statistics (eg. the amount of active PReLUs per layer) is sufficient to do so.

The experimental setup is the following: we regularly train a ReLU network and linearize it in a post-training phase, replacing ReLUs by regularized PReLUs as described above. We then transfer the amount of nonlinear units to a newly initialized network and re-train the network 5 times, using the same number of epochs and schedule as in the original training phase. We also use (non-regularized) PReLU units in the network for re-training, so that the amount of parameters is comparable. Apart from the mask of inactive PReLUs, everything else (network weights, optimizer etc.) is re-initialized (with a random seed) and the network is trained from scratch. We consider three different ways of transferring the distribution of inactive PReLU units from the partially linearized network to the new network that work at different granularity: exact, layer-wise and network-wise:

- **Exact:** The exact binary masks of inactive PReLU are kept.
- **Layer-Wise Permutation:** The binary masks of inactive PReLUs are kept but shuffled with all PReLU units within the same layer.
- **Global Permutation:** The binary masks of inactive PReLUs are kept but shuffled with all PReLU units in the network.

We summarized the results in Figure 2, where we abbreviated "S" for Short and "NS" for NoShort networks. We see that for the "easy" dataset CIFAR-10, the nonlinear advantage is nonexistent since all networks trained with the exact and layerwise permutated nonlinearities reach the full performance of the network that was partly linearized after training. The slight gain in performance can be attributed to the higher number of epochs where the network can adapt to the missing nonlinearity. Further we see that only for the ResNet56 NoShort, the network which differs most from a uniform distribution in its remaining PReLU units (ref. Figure 4), the full performance was not reached with a global permutation in PReLU masks whereas for all other architectures, full performance was reached. We conclude that nonlinear advantage is nonexistent for this easy dataset and the layerwise distribution of PReLU units matters only for networks with a very distinct (non-uniform) structure in its remaining PReLU units. As for the significantly more difficult dataset CIFAR-100, we see that no setting can reach the performance of the network that was partly linearized after training, not even re-training where the exact PReLU masks are transferred; this indicates the existence of a nonlinear advantage for harder problems.

4.1.2. Nonlinear advantage is stronger in early training

In a second step, we want to break down how linearizing a network at different stages of training affects its performance. For this, we train a ResNet56 Short on datasets of varying difficulty since previous results indicate that we can only measure it on harder datasets. At different stages of the training, we activate our regularizer with a fixed regularization weight, resume training and measure the final performance of the network after a given number of epochs. As regularizing the network at different stages of training with the same regularization weight can result in massive differences in the amount of inactive PReLUs, we slightly modified our regularizer to stop when a goal percentage ($80 \pm 1\%$, an amount high enough to impact performance) of inactive PReLUs over all layers is reached. In order not to overshoot our goal percentage, we lower the regularization weight when close to our target. We carefully tune the regularization weight in order to avoid undershooting the

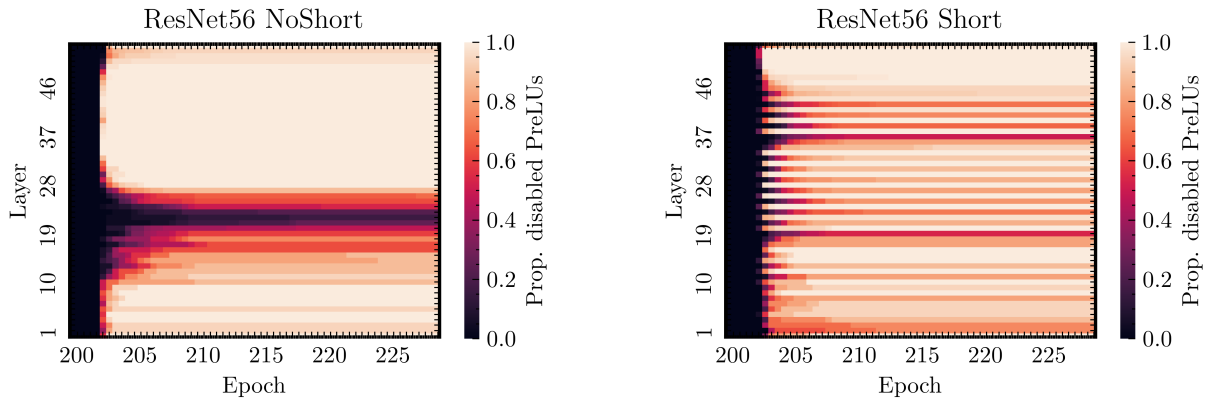


Figure 4. Proportion of inactive PReLUs when partially linearizing a ResNet56 NoShort / Short with $\omega = 0.003$.

target percentage. We see in Figure 3 that networks regularized later in training are significantly more performant than networks partially linearized earlier in training. The biggest differences are visible in the first 15 epochs of training and the effect is particularly pronounced for the harder datasets, indicating a correlation between effect strength and the hardness of the task at hand. In the Appendix in Figure 23 we have shown that for a transformer architecture training on a machine translation task, the test perplexity of networks is higher for networks partially linearized early as opposed to later epochs. The biggest difference occurs within the first 20 epochs of training.

4.2. Performance and Structure of Partially Linearized Networks

Having established in the previous sections that the performance of a network linearized after training cannot simply be recovered by training with a similar amount of nonlinear units from scratch, we now want to understand better *how shallow* we can make a trained network until performance collapses and *where the remaining nonlinearities are located* in the network.

We observe the temporal evolution of the linearization process for two different network architectures in Figure 4. When evaluating the proportion of inactive PReLUs per layer, we note that every architecture presents a distinct pattern: for the ResNet56 NoShort, we see that the remaining nonlinearity is concentrated in a connected block, whereas for the ResNet56 Short, the remaining active PReLUs are distributed more evenly over the layers. Interestingly, the connected block of remaining nonlinearity in the ResNet56 NoShort is located in the middle of the network and not on either end, excluding simple vanishing/exploding gradients at initialization effects as a cause. The fact that for the ResNet56 NoShort, many layers are fully linearized *without explicit incentive to do so* might indicate that such network architectures might not use their full expressive potential. The stripe-like structure of remaining nonlinearities in the ResNet56 Short corresponds to the placement of the residual connections and indicates that these might help in utilizing the full depth of the network. We found the qualitative behavior for both architectures to be consistent on the CIFAR-100 dataset (ref. Appendix), albeit the exact location of the connected nonlinear layer block changes. We conclude that despite regularizing every nonlinear unit equally, distinct patterns form in the remaining nonlinearities in the network that depend on network architecture.

Second, we want to demonstrate the effects of partial linearization on generalization performance for different architectures and datasets. We plotted the performance of partially linearized networks for different choices of ω on Cifar10 in Figure 5 (ref. Appendix Figure 19 for Cifar100). Darker colors represent a higher regularization weight and the disk sizes represents the global proportion of inactive PReLUs. We can see that for all networks, the top-1 test performance remains high even for a comparably small NAPL values until it collapses. We also see that depending on network architecture, for a similar NAPL value, different networks architectures present a distinct percentage of inactive PReLUs, further supporting our claim of a network-dependent structure being extracted by linearization. A similar plot showing explicitly the proportion of active PReLU units instead of NAPL can be found in the Appendix in Figure 16 (resp. 17 for Cifar100) and shows qualitatively the same behavior. We further verified our claims for a ResNet50 on the ImageNet dataset in Figure 6; note that this network contains a non-ReLU activation layer (maximum pooling) that we included in our calculations by increasing all to the NAPL values by one.

4.3. Analyzing the Shape of the "Core Network"

In this section, we investigate whether we can find some regularities in the shape of the resulting network if we partially linearize networks of different initial shape.

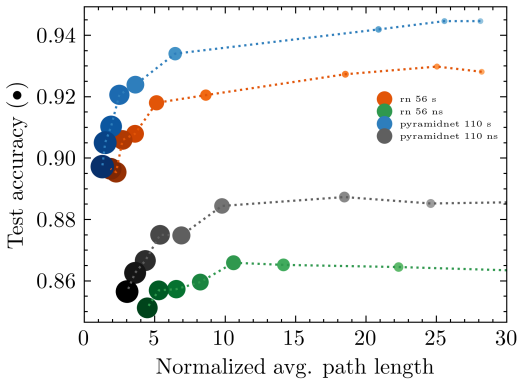


Figure 5. NAPL and test accuracy for different partially linearized network architectures with $\omega \in [0.0005, 0.005]$.

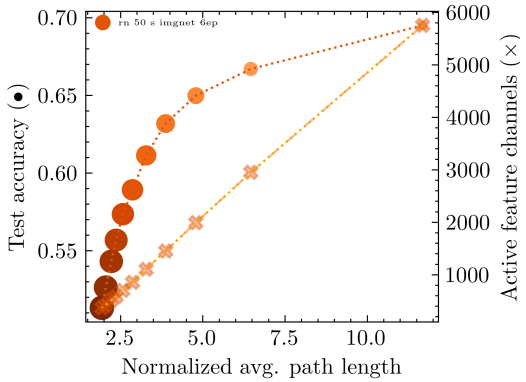


Figure 6. NAPL, accuracy and active features channels for a partially linearized ResNet50 pre-trained on Imagenet with $\omega \in [0.0005, 0.0025]$.

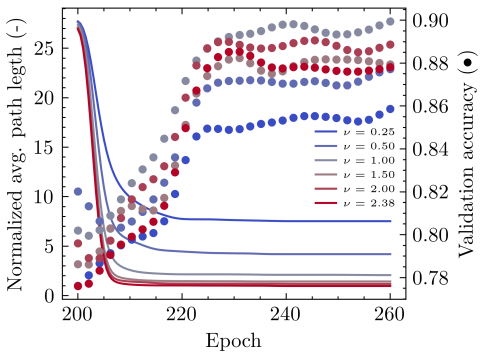


Figure 7. Validation accuracy and average path length during linearization of ResNets56 of different width for $\omega = 0.003$.

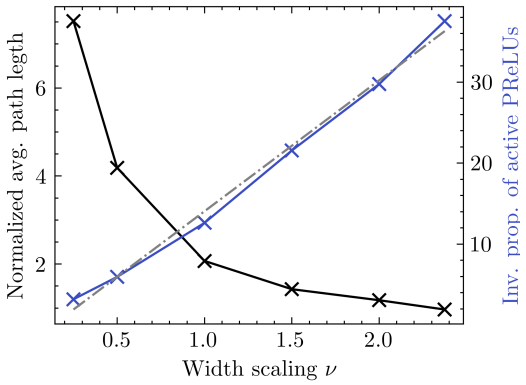


Figure 8. Inverse proportion of active PReLUs and NAPL for ResNet56 Short of different width after linearization for $\omega = 0.003$. A linear fit to the blue curve is drawn in grey.

4.3.1. ENW Converges Approx. Independently of Initial Width

We now want to analyze the effect of network width on the shape of the resulting partially linearized network. We therefore apply our linearization technique to different versions of ResNet56 Short scaled in width by a factor of $\nu \in \{0.25, 0.5, 1, 1.5, 2, 2.38\}$ and measure the NAPL and performance of the resulting networks. In Figure 7, we see that wider networks seem to have better performance but lower NAPL. The relationship between the number of filters in a layer and the average path length seems reciprocal: this would imply that the average number of active neurons per layer remains approximately equal. To confirm this, in Figure 8 we plotted the inverse proportion of active PReLUs after post-training linearization for all networks. We can see a linear relationship, confirming that the average amount of active neurons per layer remains roughly constant - independently of the initial width chosen.

4.3.2. NAPL Converges Approx. Independently of Initial Depth

We saw that independently of the network width chosen, there was a similar amount of neurons per layer that remained active. We now want to establish if we can make a similar statement with regard to network depth. Therefore, we repeated the experiment of the last section for networks of different depth. In Figure 9, we see that independently of the initial network chosen, the resulting network's NAPL converges to a similar value while having comparable training performances. Shallower networks seemingly converge to a marginally higher NAPL but this is merely an artifact of how we scaled ResNet blocks and is not observable on a simple convolutional network with constant width (ref. Appendix Figure 24). The results hint the existence of a core nonlinear structure that forms during training that is necessary to learn a given task that is approximately constant in depth and width, regardless of the initial network with chosen. Similar experiments on depth and width on

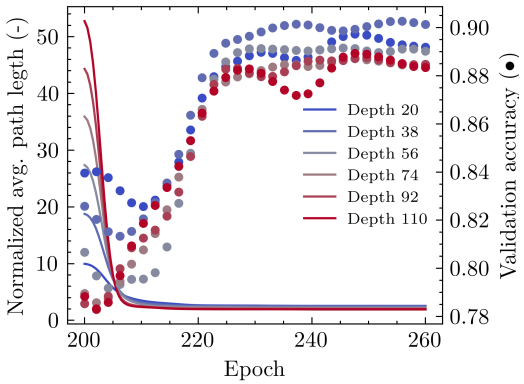


Figure 9. Validation accuracy and average path length during linearization of ResNets of different depth for $\omega = 0.003$.

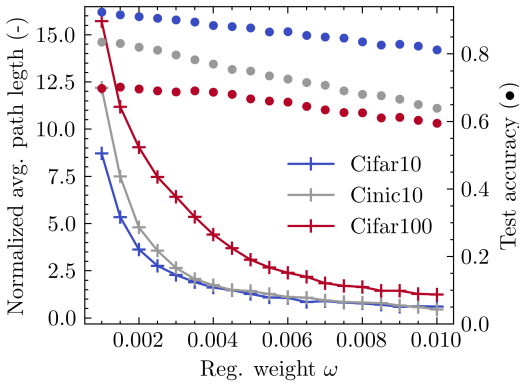


Figure 10. Test accuracy and average path length after linearization of ResNet56 for different regularization weight choices ω on the Cifar10, CINIC-10 and CIFAR-100 dataset.

CIFAR-100 in Figure 20 in the Appendix show qualitatively the same behaviour but with different NAPL values.

Note that our method yields better performances than (Dror et al., 2021) for a lower effective depth. The authors obtain a performance of 90.29% / 67.04% for a ResNet56 reduced to 10 layers (that would correspond to a NAPL close to 5) on Cifar10 / Cifar100. We obtained 90.59% / 67.64% for a ResNet56 with NAPL 2.7 / 3.1.

4.3.3. NAPL Depends on Task Difficulty

To understand how the difficulty of the task to learn affects the shape of the resulting partially linearized networks, we regularized many instances of ResNet56 Short for different choices of ω on CIFAR-10, CINIC-10 and CIFAR-100. Looking at Figure 10, we first note that by increasing the regularization weight, the NAPL of the resulting network is decreased as expected, but this effect seems to saturate exponentially while the test accuracy is only reduced linearly in ω. This seems to imply that there is a minimum NAPL necessary to learn a given task. We also see that the NAPL measured is consistently higher for harder datasets (datasets where the networks reach a low top-1 accuracy), except for very high regularization values where the CINIC-10 and CIFAR-10 curves converge. We conclude that our method is able to extract a network with minimal nonlinearity able to learn a given task.

4.4. Impact on Feature Visualizations

Finally, we visualized some feature channels from ResNet50 Short trained on ImageNet and partly linearized to different degrees ($\omega \in [0, 0.0005, 0.0025]$) with the Lucent (Kiat, 2021) library; the performance and amount of remaining active PReLU units in these networks is shown in Figure 6. We only consider feature channels that are still active in all three networks and chose one of the deeper layers for visualization in Figure 11. We see that in most cases, we can recognize a given feature over the different networks and the image seems subjectively sharper for the partially linearized networks. To confirm this, we measured the magnitude of the image gradient, averaged over all 1721 feature channels remaining in all three network and rgb-channels. This sharpness measure (indicated as S) is indeed significantly higher for the linearized networks, but we cannot discern a further increase for the network with higher degree of linearization.

5. Summary of Contributions and Discussion

In our work, we observe that linearized networks extracted *after* training outperform networks with the same amount of nonlinear units trained *from scratch* in experiments on convolutional and transformer architectures trained on computer vision and machine translation tasks. This is a highly surprising observation, as the final network architecture and thus it's expressivity are the same, but still, worse minima are found when training shallow architectures from scratch as opposed to training a deeper architecture and making it shallower afterwards. This is, to our best knowledge, the first time such an effect is described in literature. Complementary findings in recent literature describe a similar effect of finding a simpler "core structure" contained in the network in early training: for fully linearizing the network with regard to its weights (time/data-dependent NTK) (Fort et al., 2020) and for drawing "lottery tickets" (You et al., 2020). By reducing nonlinearity at channel-level, our method is able to extract networks containing significantly less nonlinear units while maintaining a similar performance, compared to previous attempts (Dror et al., 2021). We conclude that theoretical results analyzing very shallow networks e.g. (Safran et al., 2022) might have higher significance on networks used in practice than previously thought and that network depth mostly benefits the training process in early stages,

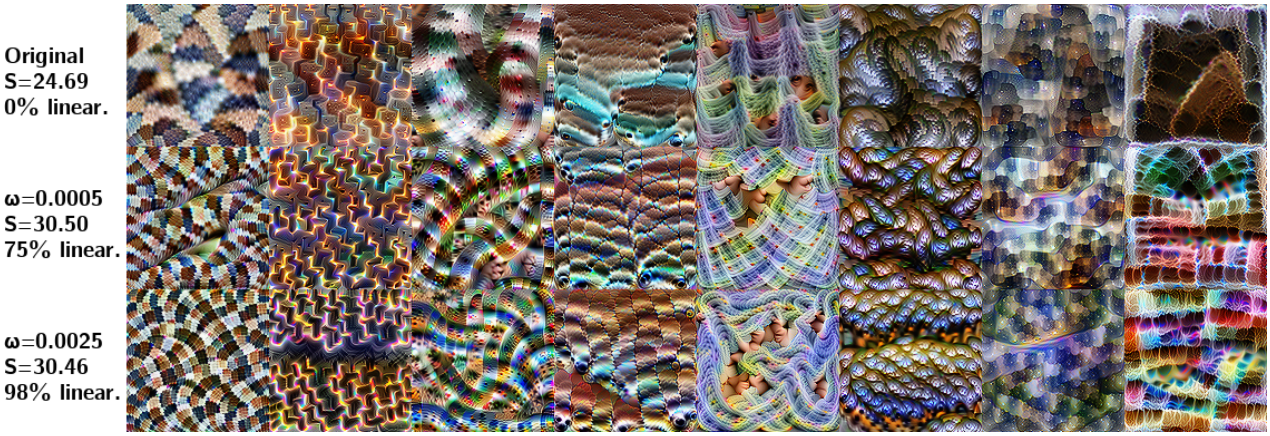


Figure 11. Visualization of *the same* feature channel per column for a ResNet50 trained on ImageNet across different degrees of partial linearization. The value of S indicates the average image sharpness, measured by the average image gradient magnitude.

as most of their nonlinear expressive power or depth is not utilized after training.

Since our method prunes nonlinearity at a channel-level as opposed to network weights, it is easier to relate the reduction in complexity to function space in terms of effective width and depth. In order to sensibly characterize the depth of a network with many linearized channels, we introduce the average path length of a network, a measure that counts the average number of nonlinear units encountered over all paths in the computation graph of the network. Linearizing networks to different degrees then allows us to characterize the minimum effective width and depth needed in order to solve a given problem. In experiments on multiple computer vision datasets, we found that these values are roughly constant for a given problem and fixed regularization weight, independently of the initial network chosen and that that the effective depth of a network grows with problem complexity. Further, we found that the essential nonlinear units in a network are distributed rather uniformly over the layers for residual networks as opposed to plain feedforward networks where they seem form clusters. This indicates that in deep networks without residual connections, there could be large connected blocks of layers that contribute very little to the learned function's nonlinear complexity.

Finally, since our method drastically reduces the amount of nonlinear feature channels in a network, we envision it can be useful for researchers trying to explain a network's behavior through visual inspection of its features. Our method not only reduces the number of nonlinear feature channels but also measurably increases the sharpness of gradient-based feature visualizations.

6. Acknowledgments

The authors acknowledge funding from the Emergent AI Center funded by the Carl-Zeiss-Stiftung. The authors would like to thank Daniel Franzen and Michael Wand for their helpful discussions.

References

Ali Mehmeti-Göpel, C. H. X., Hartmann, D., and Wand, M. Ringing relus: Harmonic distortion analysis of nonlinear feedforward networks. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL https://openreview.net/forum?id=TaYhv-q1Xit.

Allen-Zhu, Z., Li, Y., and Liang, Y. Learning and generalization in overparameterized neural networks, going beyond two layers. In Wallach, H. M., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E. B., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pp. 6155–6166, 2019. URL https://proceedings.neurips.cc/paper/2019/hash/62dad6e273d32235ae02b7d321578ee8-Abstract.html.

Arora, S., Li, Z., and Lyu, K. Theoretical analysis of auto rate-tuning by batch normalization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL https://openreview.net/forum?id=rkxQ-nA9FX.

Balduzzi, D., Frean, M., Leary, L., Lewis, J. P., Ma, K. W., and McWilliams, B. The shattered gradients problem: If resnets are the answer, then what is the question? In Precup, D. and Teh, Y. W. (eds.), *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pp. 342–350. PMLR, 2017. URL http://proceedings.mlr.press/v70/balduzzi17b.html.

Bartlett, P. L., Harvey, N., Liaw, C., and Mehrabian, A. Nearly-tight vc-dimension and pseudodimension bounds for piecewise linear neural networks. *J. Mach. Learn. Res.*, 20:63:1–63:17, 2019. URL http://jmlr.org/papers/v20/17-612.html.

Belkin, M., Hsu, D., Ma, S., and Mandal, S. Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences*, 116(32):15849–15854, 2019.

Cybenko, G. Approximation by superpositions of a sigmoidal function. *Math. Control. Signals Syst.*, 2(4): 303–314, 1989. doi: 10.1007/BF02551274. URL https://doi.org/10.1007/BF02551274.

Darlow, L. N., Crowley, E. J., Antoniou, A., and Storkey, A. J. CINIC-10 is not imagenet or CIFAR-10. *CoRR*, abs/1810.03505, 2018. URL http://arxiv.org/abs/1810.03505.

Deng, J., Dong, W., Socher, R., Li, L., Li, K., and Fei-Fei, L. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2009), 20-25 June 2009, Miami, Florida, USA*, pp. 248–255. IEEE Computer Society, 2009. doi: 10.1109/CVPR.2009.5206848. URL https://doi.org/10.1109/CVPR.2009.5206848.

Dror, A. B., Zehngut, N., Raviv, A., Artyomov, E., Vitek, R., and Jevnisek, R. J. Layer folding: Neural network depth reduction using activation linearization. *CoRR*, abs/2106.09309, 2021. URL https://arxiv.org/abs/2106.09309.

Elliott, D., Frank, S., Sima'an, K., and Specia, L. Multi30k: Multilingual english-german image descriptions. In *Proceedings of the 5th Workshop on Vision and Language, hosted by the 54th Annual Meeting of the Association for Computational Linguistics, VL@ACL 2016, August 12, Berlin, Germany*. The Association for Computer Linguistics, 2016. doi: 10.18653/v1/w16-3210. URL https://doi.org/10.18653/v1/w16-3210.

Fort, S., Dziugaite, G. K., Paul, M., Kharaghani, S., Roy, D. M., and Ganguli, S. Deep learning versus kernel learning: an empirical study of loss landscape geometry and the time evolution of the neural tangent kernel. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H. (eds.), *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/405075699f065e43581f27d67bb68478-Abstract.html.

Frankle, J. and Carbin, M. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL https://openreview.net/forum?id=rJl-b3RcF7.

Gebhart, T., Saxena, U., and Schrater, P. A unified paths perspective for pruning at initialization. *CoRR*, abs/2101.10552, 2021. URL https://arxiv.org/abs/2101.10552.

Han, D., Kim, J., and Kim, J. Deep pyramidal residual networks. In *2017 IEEE Conference on Computer Vi-*

sion and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017, pp. 6307–6315. IEEE Computer Society, 2017. doi: 10.1109/CVPR.2017.668. URL https://doi.org/10.1109/CVPR.2017.668.

Hanin, B. and Rolnick, D. Deep relu networks have surprisingly few activation patterns. In Wallach, H. M., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E. B., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pp. 359–368, 2019. URL https://proceedings.neurips.cc/paper/2019/hash/9766527f2b5d3e95d4a733fcfb77bd7e-Abstract.html.

He, K., Zhang, X., Ren, S., and Sun, J. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*, pp. 1026–1034. IEEE Computer Society, 2015a. doi: 10.1109/ICCV.2015.123. URL https://doi.org/10.1109/ICCV.2015.123.

He, K., Zhang, X., Ren, S., and Sun, J. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*, pp. 1026–1034. IEEE Computer Society, 2015b. doi: 10.1109/ICCV.2015.123. URL https://doi.org/10.1109/ICCV.2015.123.

He, K., Zhang, X., Ren, S., and Sun, J. Identity mappings in deep residual networks. In Leibe, B., Matas, J., Sebe, N., and Welling, M. (eds.), *Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part IV*, volume 9908 of *Lecture Notes in Computer Science*, pp. 630–645. Springer, 2016. doi: 10.1007/978-3-319-46493-0_38.

Hochreiter, S. Untersuchungen zu dynamischen neuronalen netzen. Diploma thesis, TU Munich, 1991.

Ioffe, S. and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Bach, F. R. and Blei, D. M. (eds.), *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, volume 37 of *JMLR Workshop and Conference Proceedings*, pp. 448–456. JMLR.org, 2015.

Jacot, A., Gabriel, F., and Hongler, C. Neural tangent kernel: Convergence and generalization in neural networks. volume 31, pp. 8571–8580. Curran Associates, Inc., 2018.

Kiat, L. S. Lucent. https://github.com/greentfrapp/lucent, 2021.

Krizhevsky, A. Learning multiple layers of features from tiny images. pp. 32–33, 2009. URL https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In Bartlett, P. L., Pereira, F. C. N., Burges, C. J. C., Bottou, L., and Weinberger, K. Q. (eds.), *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States*, pp. 1106–1114, 2012. URL https://proceedings.neurips.cc/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html.

Lakshminarayanan, C. and Singh, A. V. Neural path features and neural path kernel : Understanding the role of gates in deep learning. In *Advances in Neural Information Processing Systems(NeurIPS)*, volume 33, 2020.

Le, Y. and Yang, X. S. Tiny imagenet visual recognition challenge. 2015.

Neill, J. O. An overview of neural network compression. *arXiv preprint arXiv:2006.03669*, 2020.

Roberts, D. A., Yaida, S., and Hanin, B. *The Principles of Deep Learning Theory*. Cambridge University Press, 2022. https://deeplearningtheory.com.

Safran, I., Vardi, G., and Lee, J. D. On the effective number of linear regions in shallow univariate relu networks: Convergence guarantees and implicit bias. *CoRR*, abs/2205.09072, 2022. doi: 10.48550/arXiv.2205.09072. URL https://doi.org/10.48550/arXiv.2205.09072.

Santurkar, S., Tsipras, D., Ilyas, A., and Madry, A. How does batch normalization help optimization? In Bengio, S., Wallach, H. M., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pp. 2488–2498, 2018. URL https://proceedings.neurips.cc/paper/2018/hash/905056c1ac1dad141560467e0a99e1cf-Abstract.html.

Su, J., Chen, Y., Cai, T., Wu, T., Gao, R., Wang, L., and Lee, J. D. Sanity-checking pruning methods: Random tickets can win the jackpot. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H. (eds.), *Advances in Neural Information Processing Systems 33:*

Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/eae27d77ca20db309e056e3d2dcd7d69-Abstract.html.

Tan, M. and Le, Q. V. Efficientnet: Rethinking model scaling for convolutional neural networks. In Chaudhuri, K. and Salakhutdinov, R. (eds.), *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pp. 6105–6114. PMLR, 2019. URL http://proceedings.mlr.press/v97/tan19a.html.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need. In Guyon, I., von Luxburg, U., Bengio, S., Wallach, H. M., Fergus, R., Vishwanathan, S. V. N., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pp. 5998–6008, 2017. URL https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html.

Veit, A., Wilber, M. J., and Belongie, S. J. Residual networks behave like ensembles of relatively shallow networks. In Lee, D. D., Sugiyama, M., von Luxburg, U., Guyon, I., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pp. 550–558, 2016. URL https://proceedings.neurips.cc/paper/2016/hash/37bc2f75bf1bcfe8450a1a41c200364c-Abstract.html.

Wilson, A. G. and Izmailov, P. Bayesian deep learning and a probabilistic perspective of generalization. *Advances in neural information processing systems*, 33:4697–4708, 2020.

You, H., Li, C., Xu, P., Fu, Y., Wang, Y., Chen, X., Baraniuk, R. G., Wang, Z., and Lin, Y. Drawing early-bird tickets: Toward more efficient training of deep networks. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL https://openreview.net/forum?id=BJxsrgStvr.

Zhang, H., Dauphin, Y. N., and Ma, T. Fixup initialization: Residual learning without normalization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL https://openreview.net/forum?id=H1gsz30cKX.

A. Details About Histogram Computation

A.1. Normalizing the Histogram

Veit et al. (2016) also study the distribution of path lengths in neural networks. To model the path distribution in a residual network, the authors simply use a binomial distribution, giving the main and residual branch of a residual block equal weight. Since the authors use a binomial distribution to model path length, the average path length of a ResNet of depth d before linearization is $d/2$ in their model. Figure 13 (left) makes it clear that in our (non-normalized) model as described above, the average path length would be much closer to d since the main path contribution is exponentially bigger than the residual path contribution and thus vanishes in depth.

In order to obtain an equal contribution from the main branch and the residual branch of residual block, we can modify our model to normalize the histograms before adding them together as shown in Figure 13 (right). Since we explicitly modeled the length of residual connections (how many layers are skipped), the path length of a ResNet before linearization depends on ResBlock size in our model. For a standard ResNet using BasicBlock (ResBlock size 2), we found the initial NAPL to be close to $d/2$, making our normalized average path length model similar to the one from Veit et al. (2016).

B. Complementary Experiments

B.1. Plots for APL Instead of NAPL

In this section, we repeat the experiments of Figure 5, 9 and 7 showing APL instead of NAPL. In Figure 18, 14 and 15, we see fundamentally no differ ence in the results except for all unnormalized path length values being consistently higher than the normalized ones.

B.2. Plots for Active PReLU Percentage Instead of NAPL

In Figure 16 (resp. 17 for Cifar-100) show the results of Figure 5 (resp. 19) but plotting the global percentage of active PReLU units instead of NAPL on the x-axis. We see that qualitatively, we obtain a very similar result. This serves as sanity-check our our (N)APL measure, showing that the measurements made qualitatively correspond to the ones made with a much simpler measure.

B.3. Experiments on Cifar100

In Figure 22, 19, 20 and 21 we repeated the experiments of Figure 4, 5, 9, 7 on CIFAR-100 and see qualitatively the same behaviour although all measured NAPL value are consistently higher than on CIFAR-10.

B.4. APL of Networks with Constant Width

In Figure 9, we analyzed the average path length of ResNets of different size after applying our post-training linearization procedure. In Figure 9 all networks seem to converge approximately to the same APL independently of network depth, but shallower networks seem to yield a slightly higher APL. We wanted to further investigate this pattern.

In Figure 24, we repeated the same experiment with a simplistic Toy-Net having constant width and no striding after the first layer and see the pattern dissapear completely. We conclude that the observed pattern is an artifact of scaling ResBlocks of different width and striding operations in the network.

B.5. Nonlinear Advantage on Transformer Architectures

In Figure 23, we repeated the experiment of Figure 3 for a transformer network training on the Multi30k german-english translation task. We decided to linearize 100% of all PReLU activations, as transformer architectures contain more nonlinearities than just ReLU units and we need to remove enough nonlinearity in the network in order to significantly impact performance. We see the result of the main section confirmed: networks linearized at a later stage of training outperform networks linearized earlier on.

C. Architecture and Training Details

C.1. Architecture Details

As described in the main paper, we used a ResNet architecture with BasicBlock v2 and BasicBlockPyramid with and without residual connections for the CIFAR-10 / CINIC-10 / CIFAR-100 runs. We used shortcut option "A" (padding) for all networks except in Section 4.3.1 where option "B" (1x1 convolutions) is needed to make the network work with different widths. We used the default number of planes $num\_planes = (16, 32, 64)$ for each BasicBlock, except for Figure 7 where the number of planes is multiplied by a constant ν. For Figure 9, we used $num\_blocks = (i, i, i),\ i \in \{3, 6, 9, 12, 15, 18\}$ to scale the number of blocks in the ResNet. PyramidNet 41 resp. 110 uses $num\_blocks = (3, 4, 6)$ resp. $18, 18, 18$ and $num\_planes = (32, 128)$ resp $num\_planes = (16, 100)$ with shortcut option "A".

For the ImageNet runs, we used the ResNet50 architecture from the official TorchVision repository.

For Figure 24, we used a simple Conv-BN-ReLU ToyNet with constant width (32 Filters), no striding after the first layer, residual connections of length 1 and a final fully connected layer.

Details of the transformer architecture can be found in Figure 25 (left).

C.2. (Post-)Training Hyperparmeters & Hardware

The experiments in the paper were made on computers running Arch Linux, Python 3.10.5, PyTorch Version 1.11.0+cu102. The GPUs used were NVIDIA GeForce GTX 1080 Ti and NVIDIA GeForce RTX 2080 Ti.

The hyper-parameters in Figure 26 were used to (post-) train on the CIFAR-10, CINIC-10 and CIFAR-100 and usually reach the standard test-accuracy of approximately 92.7 for a ResNet56 on CIFAR-10. As for the ImageNet runs, we used a pre-trained model from the torchvision model-zoo. For post-training, the hyper-parameters in Figure 27 were used. For the CINIC-10 post-training, we adapted the number of epochs and the multistep scheduler milestones to approximately maintain the same number of batches since the total number of training images is different.

Note: The experiments of Figure 2 and 4 have a shorter post-train phase of 30 epochs instead of 60 epochs (the multistep milestones are 10/20) to save compute, as these experiments do not aim for maximum accuracy.

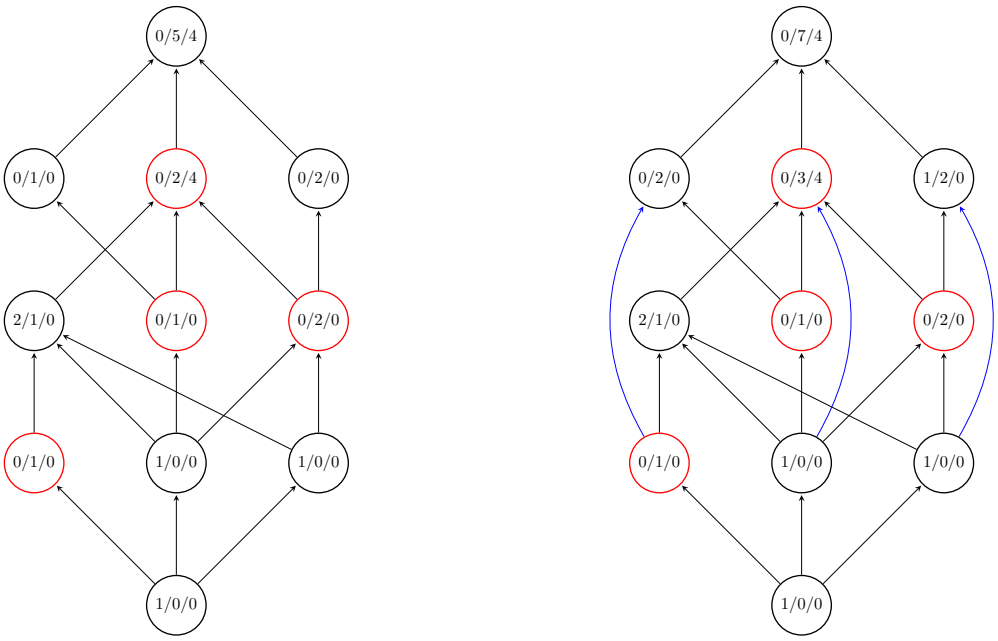


Figure 12. Computing the histogram of path lengths through dynamic programming for a non-residual (left) and a residual (right) network. Red circles designate nodes with an active PReLU activation, blue edges designate residual connections.

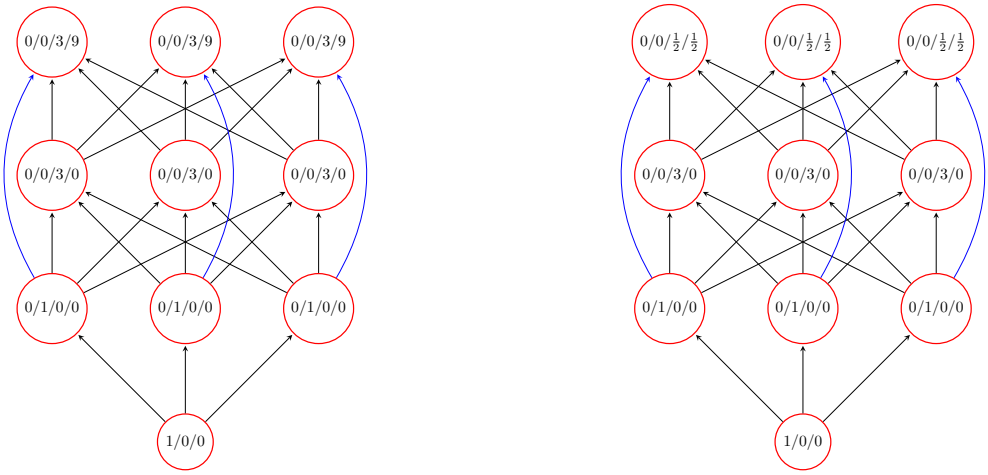


Figure 13. Computing the unnormalized (left) and normalized (right) histogram of a residual network where all PReLUs are active. Without normalization, the residual contribution vanishes.

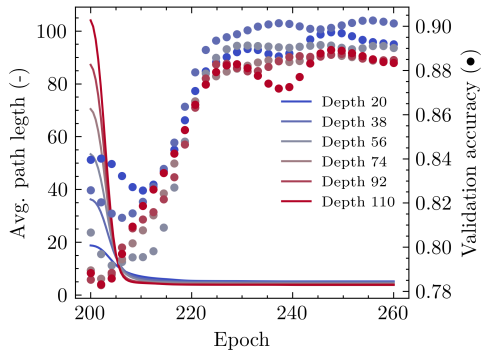


Figure 14. Validation accuracy and **unnormalized** average path length during linearization of ResNets of different depth for $\omega = 0.003$.

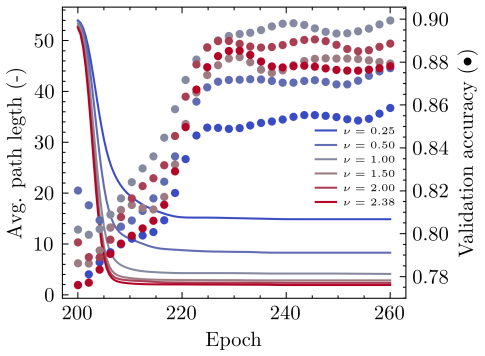


Figure 15. Validation accuracy and **unnormalized** average path length during linearization of ResNets of different width for $\omega = 0.003$.

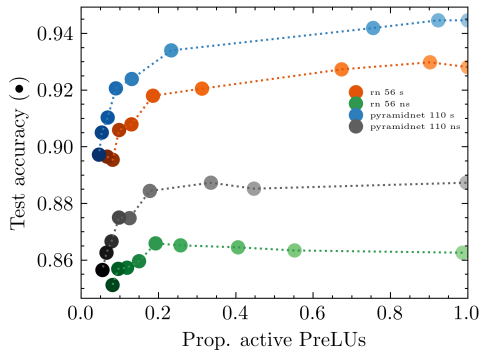


Figure 16. **Proportion of active PReLU units** and test accuracy for different network architectures with regularization weight $\omega \in [0.0005, 0.005]$.

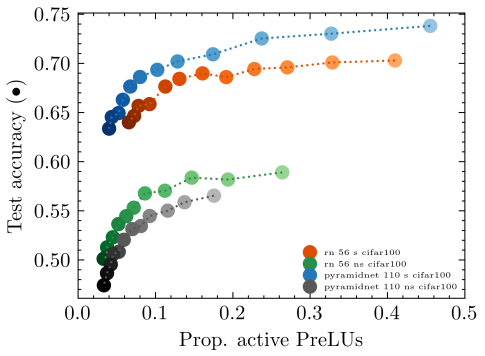


Figure 17. **Proportion of active PReLU units** and test performance (left) for different network architectures with regularization weight $\omega \in [0.0015, 0.007]$ **on CIFAR-100**..

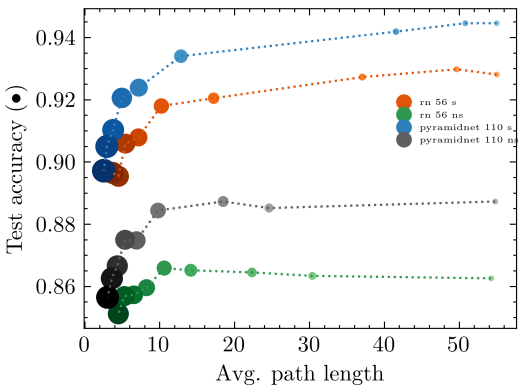


Figure 18. **Unnormalized** average path length and test accuracy for different network architectures with regularization weight $\omega \in [0.0005, 0.005]$.

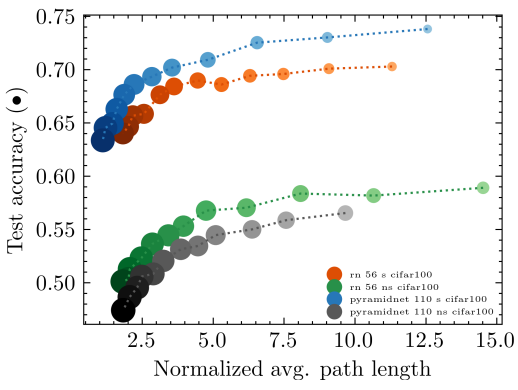


Figure 19. Normalized average path length and test performance (left) for different network architectures with regularization weight $\omega \in [0.0015, 0.007]$ **on CIFAR-100**.

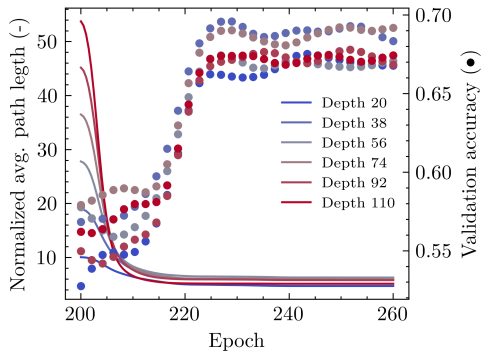


Figure 20. Validation accuracy and NAPL during linearization of ResNets of different depth for $\omega = 0.003$ **on CIFAR-100**.

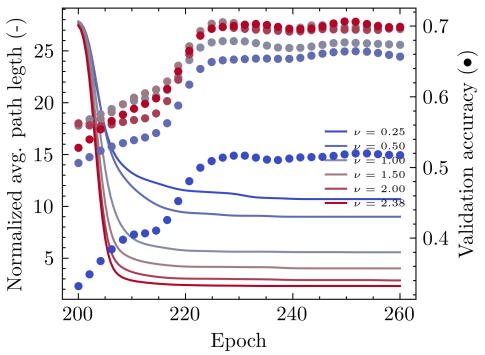


Figure 21. Validation accuracy and NAPL during linearization of ResNets56 of different width for $\omega = 0.003$ **on CIFAR-100**.

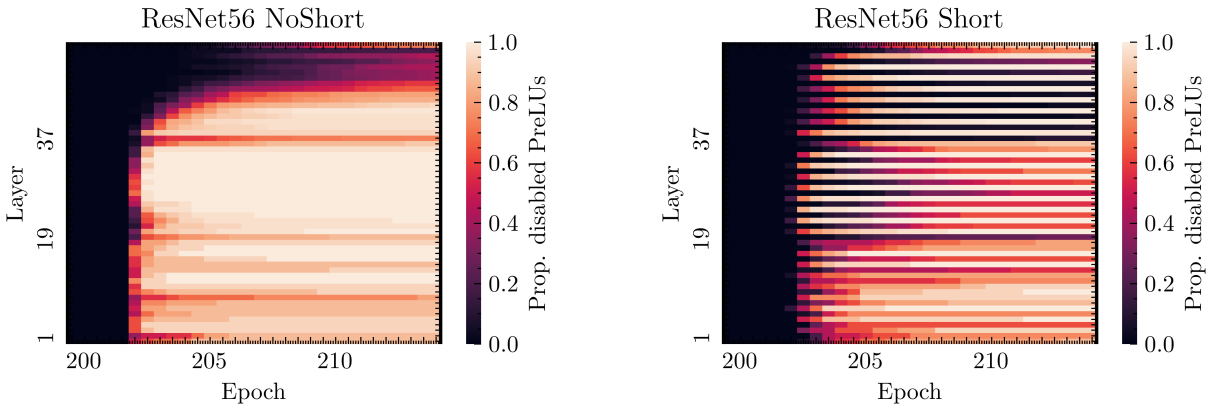


Figure 22. Proportion of inactive PReLUs when linearizing a ResNet56 NoShort / Short with $\omega = 0.003$ **on CIFAR-100**.

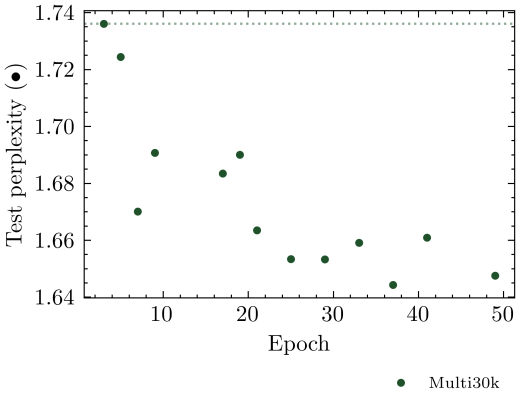


Figure 23. Test perplexity of a transformer network partially linearized at different epochs during training on a translation task. The dotted line indicates the height of the first datapoint for visual reference.

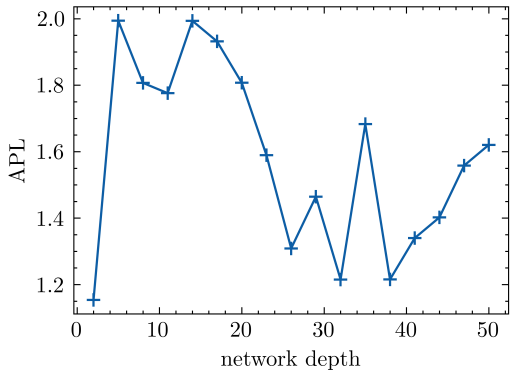


Figure 24. Average path length of Toy-Nets of different depth after post-training linearization for $\omega = 0.003$.

Architecture	Transformer
d_{ff}	2048
d_{model}	512
h	8
N	6
$p_{dropout}$	0.1

Training	Multi30k
Epochs	200
Scheduler	Warmup+Multistep ($\gamma = 0.1$)
Warmup steps	2900
Milestones	160, 180
Learning rate	0.00082
Batch size	30
Gradient accumulation	10 steps
Optimizer	ADAM
(β_1, β_2)	(0.9, 0.98)
Weight decay	0
Label Smoothing	0.1

Figure 25. Architecture details and training regime for the NLP task.

Training	CIFAR-10 / CINIC-10 / CIFAR-100
Epochs	200
Scheduler	Multistep ($\gamma = 0.1$)
Milestones	100, 150
Learning rate	0.1
Batch size	256
Optimizer	SGD + Momentum
Momentum	0.9
Weight decay	0.0001
Augmentation	Random Flip

Training	TinyImagenet
Epochs	80
Scheduler	Multistep ($\gamma = 0.1$)
Milestones	70, 75
Learning rate	0.1
Batch size	128
Optimizer	SGD + Momentum
Momentum	0.9
Weight decay	0.0001
Augmentation	Random Flip

Figure 26. Details of the training regime for CV tasks.

Post-Training	CIFAR-10 / CINIC-10 / CIFAR-100
Epochs	60 (34 for CINIC-10)
Scheduler	Multistep ($\gamma = 0.1$)
Milestones	20, 40 (10, 22 for CINIC-10)
Learning rate	0.1
Batch size	256
Optimizer	SGD + Momentum
Momentum	0.9
Weight decay	0.0001
Augmentation	Random Flip

Post-Training	Imagenet
Epochs	6
Scheduler	Multistep ($\gamma = 0.1$)
Milestones	2, 4
Learning rate	0.01
Batch size	40
Optimizer	SGD + Momentum
Momentum	0.9
Weight decay	0.0001
Augmentation	Center Crop (224 px.)

Figure 27. Details of the post-training regime.

7.3 On the Weight Dynamics of Deep Normalized Networks

Published at ICML 2024. **Co-Author's Contributions:** The underlying theory was developed jointly with Michael Wand. Jan Disselhoff and Daniel Franzen helped me in general discussions about the subject.

On the Weight Dynamics of Deep Normalized Networks

Christian H.X. Ali Mehmeti-Göpel [1] **Michael Wand** [1]

Abstract

Recent studies have shown that high disparities in effective learning rates (ELRs) across layers in deep neural networks can negatively affect trainability. We formalize how these disparities evolve over time by modeling weight dynamics (evolution of expected gradient and weight norms) of networks with normalization layers, predicting the evolution of layer-wise ELR ratios. We prove that when training with any constant learning rate, ELR ratios converge to 1, despite initial gradient explosion. We identify a "critical learning rate" beyond which ELR disparities widen, which only depends on current ELRs. To validate our findings, we devise a hyper-parameter-free warm-up method that successfully minimizes ELR spread quickly in theory and practice. Our experiments link ELR spread with trainability, a relationship that is most evident in very deep networks with significant gradient magnitude excursions.

1. Introduction

In the past decade, combining neural networks and big data has enabled dramatic breakthroughs (Krizhevsky et al., 2012; OpenAI, 2023), and network *depth* has been a key factor: Compositions of many individual layers provide rich function spaces that empirically appear to be better-aligned with real-world data distributions than any other inductive biases we are aware of today. A fundamental problem of deep networks, maybe easily brushed over as technicality at first sight, is the problem of vanishing and exploding gradients. Propagating signals through a multi-layer networks is not easy: In the forward pass, the magnitude input signals easily increases or decreases, thus leading to an exponential excursion of *signal magnitude*. Similarly, during the backward pass, we easily obtain similar excursion of *gradient magnitudes* (Yang et al., 2019). Further, using deep stacks of layers also easily increase correlations, thereby causing *vanishing dimensionality* (Saxe et al., 2013). Proper Initialization (He et al., 2015a) can reduce the problem; trying to prevent it completely in a simple feed-forward network is challenging though (Pennington et al., 2017).

Modern architectures (He et al., 2016a; Vaswani et al., 2017) thus usually address these issues by combining *residual connections* (He et al., 2016a) and variants of *normalization layers* such as *batch normalization (BN)* (Ioffe and Szegedy, 2015). The former implicitly performs a down-weighting of deep paths, exponentially with depth (Veit et al., 2016), and in combination with normalization layers, this effect is further increased (at initialization) by decreasing the weight of the residual branch (De and Smith, 2020). The central objective of our paper is to understand how the *dynamics* (over training time) of gradient *magnitude* excursions (we do not consider correlations) are affected by normalization layers (BN and the similar).

2. Related Work and Contributions

Understanding of the benefits of BatchNorm *standalone* is not straightforward and still subject to debate. The initial claim of reduced "internal covariate shift" was quickly refuted (Awais et al., 2021) and many alternative explanations were proposed, such as smoothing of the loss surface (Santurkar et al., 2018) or enabling bigger learning rates (Bjorck et al., 2018). Salimans and Kingma (2016) introduced WeightNorm, a method to decouple a layer's length and direction by training them as independent network parameters. They also demonstrated that in weight-normalized networks, gradients are orthogonal to layer weights, allowing update size calculation via the Pythagorean theorem. Hoffer et al. (2018) showed that the "effective step size" in normalized networks is approximately proportional to $\frac{1}{||W||_2^2}$; this shows that scale invariance gives us an additional degree of freedom, as scaling a layer's weights is equivalent to inversely scaling its gradients or learning rate. You et al. (2017) have observed that the ratio $\frac{||\nabla W||_F}{||W||_F}$, which we call effective learning rates (ELR), can vary wildly (up to a factor ~ 250 in AlexNet-BN) between layers after only one step of gradient descent. The authors conjecture that this can create instability in training, especially for large

*Equal contribution [1]Department of Computer Science, Johannes-Gutenberg University, Mainz, Germany. Correspondence to: Christian H.X. Ali Mehmeti-Göpel <chalimeh@uni-mainz.de>.

Proceedings of the 41^{st} International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024.

batch training requiring high learning rates, and propose to re-scale gradients by their effective learning rate. You et al. (2020) have later proposed a modified version of this algorithm for increased performance with transformer models. Brock et al. (2021) have combined a similar re-scaling of the gradients with gradient clipping and are able to train normalizer-free networks using this technique. Bernstein et al. (2020) supported this intuition by showing that in a perturbed gradient descent, an optimization step decreases the loss function if all layer-wise ELRs are bounded by a term that depends on the perturbation angle. Arora et al. (2019) described the auto rate-tuning effect, proving that gradient descent asymptotically converges to a stationary point without manual tuning of learning rates for specific layers, given certain assumptions. Wan et al. (2021) prove that the "angular update" (a measure similar to ELR) of a given normalized layer eventually converges to a constant limit value which does not depend on initial conditions, but rely on weight decay for their demonstration. Interestingly, Li and Arora (2020) show that using weight decay with a constant learning rate schedule is mathematically equivalent to using no weight decay and an exponentially increasing learning rate schedule.

The above-mentioned works show that the learning speeds of different layers do eventually align, but glance over the importance of correct learning rate scheduling in the early training phase, which we believe to be crucial in practice. We find that while convergence is always guaranteed given simplifying assumptions and over an indefinite number of iterations, choosing an excessively high learning rate, especially in the first steps of training, can drastically increase imbalances in layer-wise learning speeds (ELR spread) to a degree where recovery is impossible within a realistic time frame. Furthermore, Li and Arora show a connection between weight decay and warm-up but do not demonstrate how these techniques affect ELR spread.

In this work, we model the dynamic effects solely induced by normalization layers and assume that the layer-wise gradient magnitude excluding normalization effects (base gradient magnitude) remains constant over time. In this setting, we derive a model predicting the evolution of a network's weight dynamics (expected layer-wise gradient and weight norms). In the gradient flow, this behavior reduces to a non-linear ODE with a closed-form solution, where all ELR ratios between layers smoothly converge to 1. When training with higher learning rates, the behavior changes fundamentally, as the layer with the highest ELR can flip even below the layer with the lowest ELR in a single step if a certain *critical learning rate* is exceeded, which in turn increases ELR spread of the network. When training with constant learning rates, ELR spread can increase only during the first step, slowing down convergence, but still eventually converging. From there, we derive a warm-up scheme that is guaranteed to converge in num_layers steps. Empirically, we were able to show that high ELR spreads indeed seem to correlate with low trainability: by using techniques that control ELR spread (gradient normalization and warm-up), we are able to reduce the high (initially exponential in the number of layers) ELR spreads of a 110 layer feedforward network and render the previously un-trainable network trainable. In summary, we create a theoretical framework that shows how the dynamical effects of normalization layers can help counter gradient magnitude excursions in deep neural networks.

3. Auto Rate-Tuning Effect and Its Dynamics

The core observation is that for any layer N that is invariant to scaling in the forward pass $N(\gamma \cdot x) = N(x)$ (e.g. all normalization layers), its gradient scales inversely with its input:

$$\frac{dN}{d\gamma x}(\gamma x) = \frac{1}{\gamma}\frac{dN}{dx}(x). \tag{1}$$

This is a simple consequence of the chain rule and has been shown for BatchNorm by Wu et al. (2018) and for LayerNorm by Xiong et al. (2020). Secondly, Arora et al. (2019) show that since normalization layers are scale-invariant, no gradient can flow in this direction. Hence, weight updates ∇W are orthogonal to the weights W themselves:

$$\langle \nabla W, W \rangle = 0. \tag{2}$$

We now explore how this affects a network's weight dynamics. Intuitively, the weight norm of layers with high gradient norms grows fast and thus down-scales the gradient, leading to auto-regulation: this effect is called *auto rate-tuning*. We would like to point out that in a realistic scenario, the data tensor is multi-dimensional and condition 1 is satisfied along a subset of its dimensions (e.g. the batch, height and width axis for BatchNorm); auto-rate tuning is therefore given along those dimensions.

3.1. Sufficient Conditions for Auto Rate-Tuning / Correct Placement of Normalization Layers

A necessary condition for auto rate-tuning of a linear layer L is the invariance of the network's output with respect to re-scaling the weights in L (Arora et al., 2019). We deduct that any type of normalization layer (e.g. BatchNorm, LayerNorm) induces auto-rate tuning and that placing a normalization layer directly after every linear layer, as it is the case in most convolutional networks, is sufficient to achieve scale-invariance. In Transformer models, this was initially not the case, and we conjecture that this could explain the improvements when adding additional normalization layers in the feedforward blocks (Shleifer et al., 2021) or query/key blocks (Henry et al., 2020). Arora et al. also note that the

scale invariance property is not disrupted by positive homogeneous functions of degree 1. We infer the following classification of commonly used layers:

Auto-tuning passes through	Breaks auto-tuning
Linear layers w/o bias	Linear layers w. bias
Homogeneous nonlin. of deg. 1	Other nonlinearities
Dropout	MaxPool

If a residual connection is placed in-between a linear layer and the next normalization layer, it can break auto rate-tuning; this is the case e.g. in a ResNet v2 (He et al., 2016b).

3.2. Training Dynamics Induced by Auto Rate-Tuning

To model training dynamics, we assume that weights of a given layer, as well as their gradients, are random matrices where entries are normally distributed with zero mean and a time-dependent standard deviation that is uniform in each layer. We parameterize training time $t \in \mathbb{R}$ such that $t_i = i \cdot \lambda^2$ after i optimization steps with a constant learning rate $\lambda > 0$. In this notation, gradient descent updates can be written as $W(t_{i+1}) = W(t_i + \lambda^2) = W(t_i) - \lambda \nabla W(t_i)$. The updates preserve zero norm and uniform variance of all entries in W. Assuming the independence of all entries in the weight and gradient matrices, we can deduce the following update rule from the orthogonality condition (2):

$$||W(t_{i+1})||_F^2 = ||W(t_i)||_F^2 + \lambda^2 ||\nabla W(t_i)||_F^2. \quad (3)$$

Condition (1) implies that gradient updates are inversely proportional to the current layer weights. We now assume that the "base gradient" of a layer, meaning the gradient magnitude excluding normalization induced scaling effects, is constant during training i.e.

$$\mathbb{E}(||W(t_i)||_F \cdot ||\nabla W(t_i)||)_F = c, \quad (4)$$

for a constant $c \in \mathbb{R}$ at all times-steps t_i. We discuss the limitations of this assumption in Section 4.1.2. Using shorthand $\sigma^2(t_i) := \mathbb{E}(\|W(t_i)\|_F^2)$ and $\sigma(t_i) = \sqrt{\sigma^2(t_i)}$, we obtain:

$$\sigma^2(t_{i+1}) = \sigma^2(t_i) + \frac{\lambda^2 c^2}{\sigma^2(t_i)}, \quad (5)$$

for a constant base gradient $c > 0$ depending only on layer depth and initial weights norm that we assume to be strictly positive $\sigma^2(0) > 0$. We call this the **discrete model**.

3.3. Gradient Norms at Initialization

Feed-forward networks: The dynamics of Eq. 12 apply to all normalized layers equally, but the initial gradient norms $\|\nabla W_i(0)\|_F$ differ substantially across layers $i \leq L$: Yang et al. (2019) show that in feedforward networks with Batch Normalization, the gradient norm at initialization grows as:

$$c_i \sim \alpha^{L-i}, \quad (6)$$

with $\alpha := \sqrt{\pi/(\pi-1)} \approx 1.21$ for ReLU activations and He. initialization. See also Luther (2020) for a simplified derivation.

ResNets: When considering residual networks, as per the multivariate chain rule, the gradient of residual blocks is additive instead of multiplicative (He et al., 2016b). Additionally, frequency-dependent signal-averaging further dampens gradients in a ResNet (Ali Mehmeti-Göpel et al., 2021). It follows from the consideration for fully-connected network above and He et al. (2016b) Eq. 5 that for a residual network using ReLU units:

$$c_i \sim 1 + \lfloor \frac{L-i}{s} \rfloor \alpha^s, \quad (7)$$

where s is the number of ReLU units in a residual block.

3.4. Auto Rate-Tuning Affects Each Layer Separately

In this section, we establish that the dynamic re-scaling of gradients explored above applies to each layer independently and does not affect layers above or below, showing that a simple layer-wise view is sufficient.

Proposition 3.1 (Every Layer Auto-Tunes Separately)**.** *Consider a concatenation of a linear layer $L(x, W) = x^T W$ followed by a normalization layer N. Then, the derivative wrt. the input remains the same when layer weights are scaled by a factor γ:*

$$\frac{dN}{dx}(x, \gamma W) = \frac{dN}{dx}(x, W). \quad (8)$$

The proof can be found in the Appendix Section A.

3.5. Effective Learning Rates and Their Ratios

To account for scale variance induced by normalization layers, we are interested in the update size of the weight direction $\widehat{W} := \frac{W}{\|W\|_2}$. Similarly to van Laarhoven (2017a), by approximating $\|W(t_{i+1})\|_2 \approx \|W(t_i)\|_2$, we can write:

$$\widehat{W}(t_{i+1}) - \widehat{W}(t_i) \approx \frac{W(t_{i+1}) - W(t_i)}{\|W(t_i)\|_2} \sim \frac{\nabla W(t_i)}{\|W(t_i)\|_2}. \quad (9)$$

It is therefore imperative to consider the ratio from gradient-to-weight norm as measure of change in the layer's weights.

Definition 3.2 (Effective Learning Rate)**.** We define the effective learning rate E of a layer with weight norm σ^2 and base gradient c as:

$$E(t_i) := \mathbb{E}\left(\frac{||\nabla W(t_i)||_F}{||W(t_i)||_F}\right) = \frac{c}{\sigma^2(t_i)}. \quad (10)$$

As all effective learning rates can simply be globally re-scaled by adjusting the learning rate, we are interested in the evolution of layer-wise ratios of effective learning rates.

Definition 3.3 (Effective Learning Rate Ratios)**.** We define the effective learning rate ratio R_{jk} of two layers j and k with weight norms σ_j^2, σ_k^2 and base gradients c_j, c_k at a given time step t_i as:

$$R_{jk}(t_i) := \frac{E_j}{E_k}(t_i) = \frac{c_j \sigma_k^2}{c_k \sigma_j^2}(t_i). \tag{11}$$

3.6. Analysis in the Gradient Flow

In this section, we show that in the gradient flow, weight dynamics have a closed-form solution and all ELR ratios converge smoothly to 1.

Theorem 3.4 (Closed-Form Solution)**.** *In the gradient flow ($\lambda \to 0$), Eq. 5 has the following closed form solution:*

$$\sigma^2(t) = \sqrt{2c^2 t + k_0}. \tag{12}$$

*with $k_0 = 4$, assuming He initialization (He et al., 2015b). We will further call this the **continuous model**.*

Proof. Starting from Eq. 5, we can utilize that $t_{i+1} = t + \lambda^2$ to drop the index and solve for the difference quotient:

$$\frac{\sigma^2(t+\lambda^2) - \sigma^2(t)}{\lambda^2} = \frac{c^2}{\sigma(t)^2}. \tag{13}$$

In the limit $\lambda^2 \to 0$, this yields the gradient flow that can be expressed as a nonlinear first order differential equation :

$$\frac{d\sigma^2}{dt} = \frac{c^2}{\sigma^2} \tag{14}$$

The exact positive solution to the differential equation is given by:

$$\sigma^2(t) = \sqrt{2c^2 t + k_0}. \tag{15}$$

Assuming He initialization, the expected initial squared weight norm is 2 for layer width n. Thus, $2 = \sigma^2(0) = \sqrt{k_0}$ and therefore $k_0 = 4$. □

Theorem 3.5 (Convergence to Fixed Point)**.** *In the gradient flow ($\lambda \to 0$), all effective learning rate ratios eventually converge given enough time, i.e. for any layer pair $j, k \leq L$:*

$$\lim_{i \to \infty} R_{jk}(t_i) = 1. \tag{16}$$

Proof. We consider two arbitrary layers j and k with respective weight norms σ_j^2, $\sigma_k^2 > 0$ and base gradients c_j, $c_k > 0$. Using the formulae for gradient norm (Eq. 14) and weight norm (Eq. 12) in the continuous model, we write:

$$\frac{E_j}{E_k}(t) = \frac{c_j}{\sigma_j^2(t)} \cdot \frac{\sigma_k^2(t)}{c_k} = \frac{c_j \sqrt{2c_k^2 t + k_0}}{c_k \sqrt{2c_j^2 t + k_0}} \overset{t \to \infty}{\longrightarrow} 1. \tag{17}$$

□

3.7. Analysis for Bigger Learning Rates

In this section, we characterize the evolution of ELR ratios for non-infinitesimal, scheduled learning rates $\lambda(t_i)$, now relying solely on the discrete model. If $\lambda(t_i)$ is constant, we find the asymptotic behavior to be the same as in the gradient flow, where ELRs ratios converge in the time limit. On the contrary to the continous model, ratios can (temporarily) widen when surpassing a certain critical learning rate.

Theorem 3.6 (Convergence to Fixed Point)**.** *In the time limit and for a constant learning rate $\lambda(t_i) = \lambda$, all effective learning rate ratios converge. For any layer pair $j, k \leq L$:*

$$\lim_{i \to \infty} R_{jk}(t_i) = 1. \tag{18}$$

The proof can be found in Appendix Section A. The main idea is that by substituting $x_i := \frac{\sigma_j^2(t_i)}{c_j \lambda}$ and $y_i := \frac{\sigma_k^2(t_i)}{c_k \lambda}$, we can rewrite Eq. 4 for two distinct layers j and k as two sequences obeying the same recurrence relation and consequently bound the expression.

Proposition 3.7 (Ratios Shrink)**.** *Let $j, k \leq L$ be any layer pair.*

1. *If $R_{jk}(t_i) > 1$, the ratio R_{jk} is then strictly lower in the next time step, i.e. $R_{jk}(t_{i+1}) < R_{jk}(t_i)$.*
2. *If $R_{jk}(t_i) < 1$, the ratio R_{jk} is then strictly greater in the next time step, i.e. $R_{jk}(t_{i+1}) > R_{jk}(t_i)$.*

Proof. We start by showing the first proposition. We can reformulate the expression $\frac{E_j}{E_k}(t_{i+1}) < \frac{E_j}{E_k}(t_i)$ using the definition of the effective learning rate as the following equivalent expression:

$$\frac{c_j^2 \sigma_k^4}{\sigma_j^4 c_k^2}(t_{i+1}) < \frac{c_j^2 \sigma_k^4}{\sigma_j^4 c_k^2}(t_i). \tag{19}$$

We simplify this expression and take the square root. Since σ are variances and thus non-negative, the following expression is also equivalent:

$$\frac{\sigma_k^2}{\sigma_j^2}(t_i) - \frac{\sigma_k^2}{\sigma_j^2}(t_{i+1}) > 0. \tag{20}$$

We expand the second term using Eq. 5:

$$\frac{\sigma_k^2}{\sigma_j^2}(t_{i+1}) = \frac{\sigma_k^2 + \frac{c_k^2 \lambda^2}{\sigma_k^2}}{\sigma_j^2 + \frac{c_j^2 \lambda^2}{\sigma_j^2}}(t_i) = \frac{\sigma_j^2 \sigma_k^4 + \sigma_j^2 c_k^2 \lambda^2}{\sigma_j^4 \sigma_k^2 + \sigma_k^2 c_j^2 \lambda^2}(t_i). \tag{21}$$

Substituting this term on the left hand side of Eq. 20 and combining the terms yields:

$$\frac{\sigma_k^2}{\sigma_j^2}(t_i) - \frac{\sigma_j^2 \sigma_k^4 + \sigma_j^2 c_k^2 \lambda^2}{\sigma_j^4 \sigma_k^2 + \sigma_k^2 c_j^2 \lambda^2}(t_i) = \frac{\sigma_k^4 c_j^2 \lambda^2 - \sigma_j^4 c_k^2 \lambda^2}{\sigma_j^2 (\sigma_j^4 \sigma_k^2 + \sigma_k^2 c_j^2 \lambda^2)}(t_i). \tag{22}$$

Since the denominator is strictly positive. Eq. 20 is therefore equivalent to:

$$\sigma_k^4 c_j^2 \lambda^2(t_i) > \sigma_j^4 c_k^2 \lambda^2(t_i), \tag{23}$$

which is in turn equivalent to $\frac{E_j}{E_k}(t_i) > 1$ by definition. The second proposition can be shown analogously. □

A consequence of this proposition is that a given ratio R_{jk} diminishes during every step, except for when it flips, i.e. $R_{jk}(t_i) > 1$ and $R_{jk}(t_{i+1}) < 1$. In Appendix Section A, we show that when training with constant learning rates, this can only happen during the first step. Now, we would like to find the precise learning rate where the ratio flips.

Definition 3.8 (Flipping Ratio)**.** We define the "flipping ratio" κ_{jk} of two layers j and k at a given time step t_i as:

$$\kappa_{jk}(t_i) \coloneqq \frac{\sigma_j \sigma_k}{\sqrt{c_j c_k}}(t_i) = \sqrt{\frac{1}{E_j E_k}}(t_i). \tag{24}$$

Proposition 3.9 (Flipping Conditions)**.** *Let $j, k \leq L$ be any layer pair with w.l.o.g. be $R_{jk}(t_i) > 1$.*

1. *The effective learning rate ratio does not flip between time steps t_i and t_{i+1} i.e. $R_{jk}(t_{i+1}) > 1$ if and only if $\lambda(t_i) < \kappa_{jk}(t_i)$.*
2. *The effective learning rate ratio does flip between time steps t_i and t_{i+1} i.e. $R_{jk}(t_{i+1}) < 1$ if and only if $\lambda(t_i) > \kappa_{jk}(t_i)$.*
3. *The ratio $\frac{E_j}{E_k}$ has reached a stationary point at a given time step t_i, i.e. $R_{jk}(t_j) = R_{jk}(t_{j+1})$ for all $j \geq i$ if and only if $\lambda(t_i) = \kappa_{jk}(t_i)$.*

Proof. We start by showing the first proposition. Using the definition of R_{jk} and Eq. 21, we can write:

$$R_{jk}(t_{i+1}) = \frac{c_j \sigma_k^2}{c_k \sigma_j^2}(t_{i+1}) = \frac{\sigma_j^2 \sigma_k^4 c_j + \sigma_j^2 c_j c_k^2 \lambda^2}{\sigma_j^4 \sigma_k^2 c_k + \sigma_k^2 c_j^2 c_k \lambda^2}(t_i). \tag{25}$$

Thus, the condition $R_{jk}(t_{i+1}) > 1$ is equivalent to:

$$\left(\sigma_j^2 \sigma_k^4 c_j + \sigma_j^2 c_j c_k^2 \lambda^2\right)(t_i) > \left(\sigma_j^4 \sigma_k^2 c_k + \sigma_k^2 c_j^2 c_k \lambda^2\right)(t_i) \tag{26}$$

$$\Leftrightarrow \left(\sigma_j^2 c_j c_k^2 \lambda^2 - \sigma_k^2 c_j^2 c_k \lambda^2\right)(t_i) > \left(\sigma_j^4 \sigma_k^2 c_k - \sigma_j^2 \sigma_k^4 c_j\right)(t_i) \tag{27}$$

$$\Leftrightarrow \lambda^2 (\sigma_j^2 c_j c_k^2 - \sigma_k^2 c_j^2 c_k)(t_i) > \sigma_j^4 \sigma_k^2 c_k - \sigma_j^2 \sigma_k^4 c_j(t_i) \tag{28}$$

$$\Leftrightarrow c_j c_k \lambda^2 (\sigma_j^2 c_k - \sigma_k^2 c_j)(t_i) > \sigma_j^2 \sigma_k^2 (\sigma_j^2 c_k - \sigma_k^2 c_j)(t_i) \tag{29}$$

Since we assumed $R_{jk}(t_i) = \frac{E_j}{E_k}(t_i) = \frac{\sigma_k^2 c_j}{\sigma_j^2 c_k}(t_i) > 1$, it follows that $\left(\sigma_j^2 c_k - \sigma_k^2 c_j\right)(t_i) < 0$ and thus we invert the sign of the inequality when dividing by this quantity and we obtain the following equivalent condition:

$$\lambda^2(t_i) < \frac{\sigma_j^2 \sigma_k^2}{c_j c_k}(t_i). \tag{30}$$

All quantities are non-negative, therefore taking the square root preserves equivalence and we obtain the sought condition. The other propositions can be shown analogously. □

Since we are interested in reducing the highest overall ratio $R_{h\ell}(t_i)$ where ℓ, h are the layers with the lowest respective highest effective learning rate, we call $\kappa_{\ell h}(t_i)$ the *critrical learning rate*. When using higher learning rates than this value, $E_h(t_i)$ flips below $E_\ell(t_i)$ during the next step, thus (for high λ considerably) increasing total ELR spread. In the following we will come to understand that in practice, a more conservative choice is advisable; for this reason, we propose the *subcritical*, but still provably fast warm-up scheme below.

Corollary 3.10 (Subcritical Warm-Up)**.** *Given a network with $L > 0$ layers, if we schedule the learning rate as $\lambda(t_i) = \kappa_{hh'}(t_i)$, where we chose $h, h' \leq L$ at each step to be the two layers with the highest effective learning rates, then no ratio R_{jk} for any $j, k \leq L$ ever flips between a time step and the next and all pairs of effective learning rate ratios R_{jk} for any $j, k \leq L$ converge to 1 in L steps.*

Proof. Let h, h' be the two layers with the highest effective learning rates at time step t_i. By Proposition 3.9, if we chose $\lambda(t_i) = \kappa_{hh'}(t_i)$, we have $R_{hh'} = 1$ at the next time step t_{i+1} and for all further time steps. Since h, h' are the two layers with the highest effective learning rate at time step t_i and we chose $\lambda(t_i)$ to be equal to their flipping ratio $\kappa_{hh'}(t_i)$, we have:

$$\lambda(t_i) = \kappa_{hh'}(t_i) \leq \kappa_{jk}(t_i) \tag{31}$$

for all other layers $j, k \leq L$. Therefore, by Lemma 3.9, no ratio R_{jk} will ever flip between a time step and the next for any $j, k \leq L$. If we repeat this process L times, all pairwise learning rate ratios converge to 1. □

3.8. Simulating Warm-Up Schedulers and Criticality

In Figure 1, in order to visualize the concept of criticality, we simulated the evolution of effective learning rates and weight norms for popular learning rate schedulers with our discrete model (ref. Eq. 5), assuming initially exponentially exploding gradients (ref. Eq. 6). We also indicated $\lambda(t)$ along with the critical learning rate $\kappa_{\ell h}(t)$. As predicted by our analysis in Section 3.7, whenever $\lambda(t) > \kappa_{\ell h}(t)$, we see

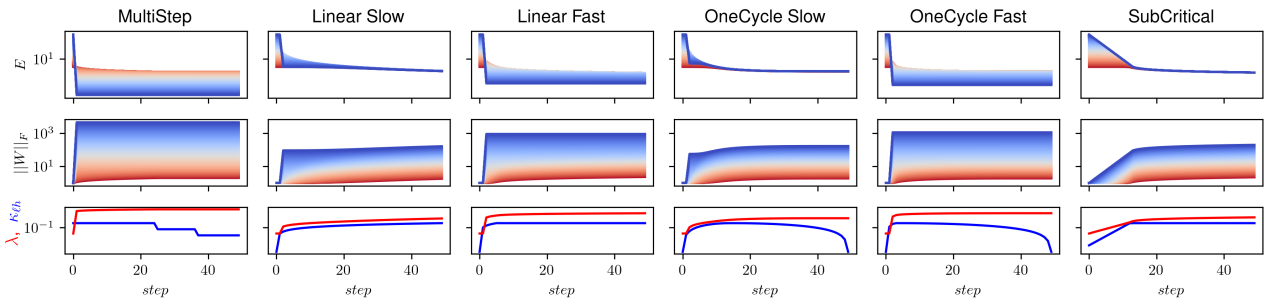


Figure 1. Simulated evolution of layer-wise effective learning rates **(top)**, weight norms **(middle)**, learning rates $\lambda(t)$ and critical LRs $\kappa_{\ell h}(t)$ **(bottom)** for popular learning rate schedulers. All y-axes are in logarithmic scale. Blue color corresponds to the lower layers.

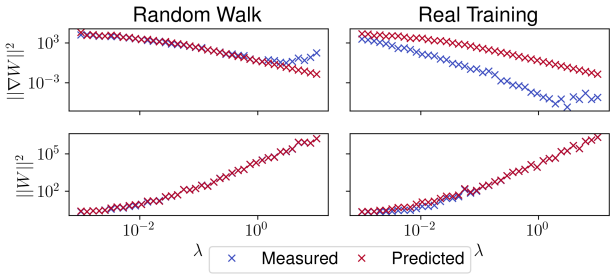


Figure 2. Short term evolution (after 10 steps) of predicted vs. measured weight and gradient norms of the lowest layer of a ResNet56 NoShort trained with random gradients **(left)** and real gradients **(right)** for various λ.

that the highest effective learning rate flips below the lowest, which in turn increases the ELR spread and consequently the convergence time. We conclude that whether a warm-up scheme succeeds in quickly reducing ELR spread highly depends on the chosen hyper-parameters.

4. Experimental Validation

In this experimental Section, we will first check the limitations of the assumption about constant base gradients and validate the predictivity of our model. Then, we will compare the predicted critical learning rate to an empirical value extracted from real training runs. Finally, we confirm that high ELR spreads correlate with network trainability in practice.

4.1. Experimental Setup

4.1.1. Architectures, Datasets and Training Protocols

We chose ResNet v1 (He et al., 2016a) with ("Short") and without ("NoShort") residual connections as examples of standard architectures. We chose a ResNet v1 as opposed to a v2 since in the former, the "correct" placement of normalization layers (ref. Section 3.1) is given without modifying the architecture. Theory predicts that a high number of layers and not using residual connections increase the strength of the observed effect (ref. Section 3.3). We therefore use 56 and 110 layer networks: Without residual connections, the former is deep but still trainable and the latter is mostly un-trainable with basic constant LR training. The final layer of a ResNet v1 is not scale-invariant and we therefore exclude it from our analysis. For computer vision tasks, we work with standard image classification datasets of variable difficulty: CIFAR-10, CIFAR-100 (Krizhevsky, 2009) and ILSVRC 2012 (called ImageNet in the following) (Deng et al., 2009). We use the most basic training setting possible (vanilla SGD) and disable all possible factors that influence weight dynamics: momentum, weight decay, affine BatchNorm parameters and bias on linear layers (for a discussion, please refer to Appendix Section C). We further use different kinds of learning rate scheduling with and without warm-up; further details about the architectures and training process can be found in the Appendix.

4.1.2. Measuring ELR Spread

In our experiments, we need a measure for ELR spreads that is relative to the network's mean ELR.

Definition 4.1 (Relative Logarithmic ELR Spread)**.** We define the Relative Logarithmic ELR Spread as:

$$S_{rel} := \text{std}(\ln(E)), \tag{32}$$

computed across the layers of the network and usually averaged over all channels and the entire training process.

4.1.3. Random Walk

In the past section, we modeled exclusively the dynamics induced by normalization assuming constant base gradients (ref. Eq 4), meaning that the layer-wise expected gradient magnitude excluding normalization effects is constant

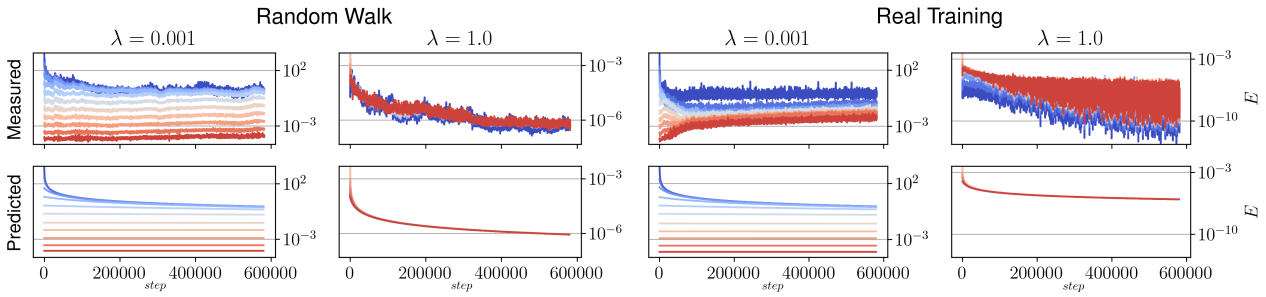


Figure 3. Long term evolution of predicted vs. measured layer-wise effective learning rates for a ResNet56 NoShort trained with random gradients **(left)** and real gradients **(right)**. Blue color corresponds to the lower layers.

over time. This is obviously not strictly true in a practical setting: apart from the obvious factors mentioned above (momentum, affine parameters etc.), the derivative of non-linear layers and the objective function itself changes when varying inputs or weights. During training, gradient norms tend to shrink as the objective function saturates (Lee et al., 2019). To verify that mostly learning effects are responsible for fluctuations in base gradients, we observe how weight dynamics evolve during a random walk.

Definition 4.2 (Random Walk)**.** During each training step, before applying the gradients computed in the backward-pass, we replace every layer's gradient by a random vector of similar norm which is also orthogonal to the layer's weights. Please refer to Algorithm 1 for a formal description.

Algorithm 1 Random Walk

Let e_ℓ denote the number of elements of the weight vector W_ℓ and $\langle \cdot, \cdot \rangle$ the dot product.

for each gradient descent step i **do**
  for each layer ℓ **do**
    Compute $\nabla W_\ell(t_i)$
    $\sigma \leftarrow \sqrt{\|\nabla W_\ell(t_i)\|_2^2 / e_\ell}$
    $R \leftarrow \mathcal{N}(0, \sigma^2)^{e_\ell}$
    $V \leftarrow W(t_i)$
    $\nabla W_\ell(t_i) \leftarrow R - \frac{\langle R, V \rangle}{\langle V, V \rangle} V$ // orthogonalize
  end for
end for

4.2. Model Validation and Limitations

To validate our theory, we measure the initial gradient and weight norms of a network and extrapolate their evolution using our discrete model (Eq. 5). We then compare the predicted weight/gradient norms to the empirically measured values after a given number of steps. We will first see that for a feedforward network with ReLU activations, it is already enough to exclude learning effects (random walk scenario) for our model to be long-term predictive. When including them, as expected, gradients are lower than predicted but the main takeaway qualitatively still holds: ELR spreads diminish over time, given that a certain learning rate is not exceeded.

Short-Term Validation : In Figure 2, we compare the measured weight/gradient norm of the lowest layer of a ResNet56 NoShort after training on Cifar10 for 10 steps to the values predicted using our discrete model on the initial values. In a random walk (left), predictions are quite accurate up to $\lambda \approx 1$ and get slightly inaccurate for higher λ, presumably due to numerical issues. As for real training (right), we see that gradients are notably smaller than expected after 10 steps. For the following, it is crucial to note that the difference between the predicted and measured values is not a constant ratio but instead increases in λ.

Long Term Validation : We conducted a similar experiment for only two different learning rates $\lambda \in \{0.001, 1\}$ over 3000 epochs and visualize the measured/predicted ELR of all layers in Figure 3. We see that in the random walk scenario, our prediction is remarkably accurate. In real training and for $\lambda = 1$, our model predicts this learning rate to be critical, but in reality it is super-critical as the gradients of the lower layers (blue) significantly undershoot with regard to the prediction and their ELR flips below the highest layers (red); further training does not seem to be able to recover the high ELR spread. Since training with any subcritical learning rate reduces ELR spread, we will see that it is sufficient to use a slightly lower λ than the predicted critical value to avoid an increase in ELR spread.

Predicting Criticality: In Figure 4, we train a ResNet110 NoShort for a single epoch using various constant learning rates on Cifar10 in a random walk and a real training scenario, tracking the evolution of ELR spreads. We plot ELR spreads at initialization and after one epoch, averaging measurements over 10 runs for each datapoint. First, we note that as predicted, up to a certain learning rate, ELR spreads are always lowered by training. Next, we indicate

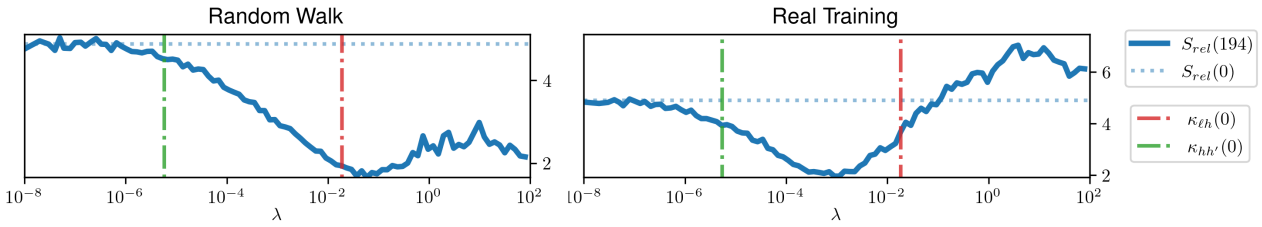


Figure 4. Relative spread after one epoch **(solid blue)**, relative spread at initialization **(dotted blue)** and the critical **(red)** / subcritical **(green)** learning rate at initialization of a ResNet110 NoShort with random gradients (**left**) and real gradients (**right**).

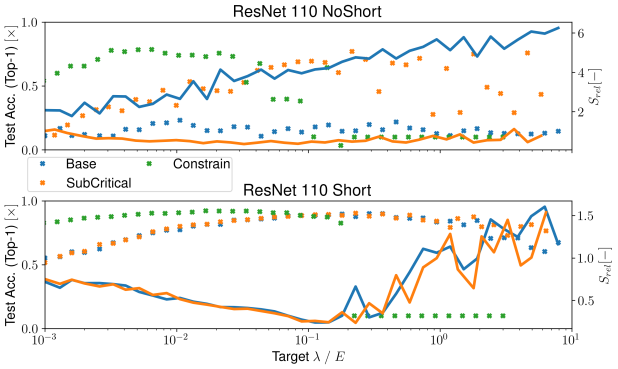


Figure 5. Test accuracies and relative spreads of a ResNet110 (No)Short trained on Cifar10 using regular, warm-up and constrained ELR training protocols for different target (E)LRs.

the predicted (sub)critical learning rate at initialization: as per Proposition A.1, if a learning rate is subcritical in step 0, it should also not increase spreads during later steps. Consequently, we expect the runs with $\lambda \approx \kappa_{\ell h}(0)$ to have the lowest S_{rel} value after training. In Figure 4, we see that this is indeed the case for the random walk. In real training, the qualitative behavior is similar but the curve is shifted to the left as gradients are smaller. We also note that in real training, when using super-critical learning rates ($\lambda > 10^{-3}$), ELRs do not seem to converge anymore, presumably to auto-rate tuning effects becoming too weak compared to fluctuations in base gradient magnitude caused by training.

4.3. ELR Spread and Trainability

In this section, we want to show empirically that networks with high ELR spreads correlate with low trainability and that lowering spreads using various methods can restore trainability. For this, we chose an experimental setting where ELR spreads are large: we train a ResNet110 (No)Short on Cifar10. For all runs, we use a simple multi-step learning rate decay. In Figure 5 (top), we see that for the "NoShort" networks in regular training without warm-up ("base"), spreads (averaged over the training run) are very high and trainability is very low. Using skip connections (bottom), spreads are much lower and the network is able to train.

4.3.1. Subcritical Warm-Up

As we have seen in Figure 4 (right), because of learning effects present in real training, the more conservative choice of using the subcritical learning rate for warm-up seems like a more sensible value to avoid overshooting in practice but still guarantees fast convergence in theory (ref. Corollary 3.10). Further, since we are using BatchNorm, we obtain channel-wise ELR values and use the maximum of these values as our layer-wise value.

4.3.2. Constraining Layer-Wise ELRs

Another possibility of controlling ELR spread is scaling each layer's gradients before each step so that layer-wise effective learning rates are constrained to be constant:

$$\nabla W \leftarrow \nabla W \cdot \frac{E_{goal}}{E + \epsilon}, \tag{33}$$

for a given constant goal effective learning rate E_{goal} and a small ϵ we chose as $\epsilon = 10^{-5}$. A similar mechanic was used in the popular LARS optimizer (You et al., 2017). To prevent increasing weight norms W from overflowing, we additionally divide all layer weights by the maximum layer weight $\widehat{W} = max(||W||_F)$ over all layers before every step. This should not change the network function since normalization layers are scale invariant and gradients are normalized. Alternatively, one could re-scale the gradients by $\frac{1}{\sqrt{1+\lambda^2}}$ after every optimization step, as described by Bernstein et al. (2020).

4.3.3. Evaluation

In Figure 5 (left), we see that both techniques lower ELR spread across layers which correlates with the previously untrainable ResNet110 NoShort becoming trainable, despite its initially exponentially exploding gradients. Although not a proof of a general causal connection between ELR spread and trainability, the fact that an untrainable network

becomes trainable with the intervention made yields some compelling evidence supporting such a hypothesis. For the network with skip-connections (right), we can observe the same effects but less pronounced, which is expected since the initial spread is linear and not exponential in the number of layers as shown in Section 3.3. In Appendix Section B, we repeat this experiment on the Cifar100 dataset and draw similar conclusions.

Finally, we train a ResNet101 (No)Short on ImageNet for 50 epochs using three different warm-up schedulers: OneCycle (Smith and Topin, 2017), sub-critical warm-up and no warm-up; we use the exact same cool-down phase (cosine) for all schedulers. We use default hyper-parameters for the OneCycle scheduler that work well in training a ResNet101 Short in a short amount of epochs on this dataset. In Table 1, we see that indeed warm-up lowers S_{rel} and correlates with increased performance, but the hyper-parameters used for OneCycle that work well with the residual network still result in significant spreads for the non-residual network with higher initial spreads. This confirms that warm-up should be scheduled as a function of current ELRs. Although subcritical warm-up uses very few warm-up steps, it yields comparable or better results than our preset OneCycle warm-up. Using the ELR-constrain method to prescribe a global ELR similar to the OneCycle run, we see that we are able to train the network without residual connections; using warm-up additionally decreases performance, showing that warm-up presents no benefits in a setting with no ELR spread.

Table 1. Test accuracies and relative spread of a ResNet101 (No)Short trained on ImageNet using different types of warm-up / ELR-constrain; *RES* indicates residual connections and *CTN* whether the ELR-constrain method was used.

RES	CTN	W. TYPE	W. STEPS	ACC.	S_{rel}
NO	NO	NONE	0	08.50	3.96
NO	NO	ONECYCLE	64060	22.82	1.96
NO	NO	SUBCRITICAL	9	41.83	0.70
NO	YES	NONE	0	47.99	-
NO	YES	ONECYCLE	64060	45.61	-
YES	NO	NONE	0	72.85	0.29
YES	NO	ONECYCLE	64060	72.83	0.31
YES	NO	SUBCRITICAL	3	73.06	0.27

5. Discussion and Future Work

In past work, high spreads in effective learning rates have been conjectured to negatively affect trainability, but to our best knowledge, no formal model exists that describes their time-based evolution in early training phases for scheduled learning rates. Under the assumption of constant gradient magnitudes beyond normalization effects, we derived a simple model from first principles that describes the evolution of expected weight/gradient norms and consequently effective learning rates during training. Under our model's assumption, we were able to prove that when training long enough using *any* constant learning rate, all ratios of layer-wise effective learning rates eventually converge to the same value. Problems can still arise in the first step(s) if the learning rate $\lambda(t_i)$ is bigger than the critical value $\kappa_{\ell h}(t_i)$ (which depends on current weight/gradient norms), momentarily increasing the disparity between layer-wise effective learning rates. We consider this theoretical model of normalization-induced dynamic effects to be our main contribution.

In a series of empirical experiments, we have shown that although we exclusively model norm-induced dynamics (scale-invariant linear layers) and assume that the expected gradient norm of other layers (objective function, nonlinear layers) does not change over time, our main takeaway still holds when training a deep convolutional ReLU network on real data: training reduces effective learning rate spread up to a certain *critical learning rate*. By using live gradient values at each step and using a slightly more conservative learning rate choice than predicted, we were able to design a hyper-parameter-free warm-up scheduler that is able to quickly reduce effective learning rate spreads in practice. In an (extreme) setting with exponentially exploding initial gradients, we show that reducing ELR spreads using warm-up or by normalizing gradients to prescribe a constant effective learning rate correlates with the network's trainability being restored.

Our analysis applies to all *normalized* networks, i.e. architectures where the network function is invariant wrt. scaling in weight matrices, which is usually the case in most normalized feedforward architectures. Unfortunately, unlike most other traditional MLPs/CNNs/ResNets, the weight matrices of attention blocks are not scale-invariant and thus the inverse scaling property (Eq. 1) and orthogonality (Eq. 2), which our model relies on, are violated. Moreover, modifying the architecture (i.e. adding additional normalization layers) would fundamentally impact its way of working (e.g. attention cannot be unlearnt anymore). Preliminary results show that for architectures containing higher degree nonlinearities (e.g. Transformer models), base gradients can vary much more compared to simple feedforward ReLU networks, therefore limiting the applicability of our model as is. If the order of the fluctuations of the base gradient exceeds that of the auto-rate tuning effects, the effect vanishes. We could envision extending our model to include an error analysis for non-constant base gradients, estimating when this is the case.

Acknowledgments

The authors acknowledge funding from the Emergent AI Center funded by the Carl-Zeiss-Stiftung. The authors would like to thank Daniel Franzen and Jan Disselhoff for their helpful discussions. The authors would also like to express their gratitude to the HPC working group of the Johannes-Gutenberg University Mainz for sharing their compute power in times of need.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

References

Christian H. X. Ali Mehmeti-Göpel, David Hartmann, and Michael Wand. Ringing relus: Harmonic distortion analysis of nonlinear feedforward networks. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL https://openreview.net/forum?id=TaYhv-q1Xit.

Sanjeev Arora, Zhiyuan Li, and Kaifeng Lyu. Theoretical analysis of auto rate-tuning by batch normalization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL https://openreview.net/forum?id=rkxQ-nA9FX.

Muhammad Awais, Md. Tauhid Bin Iqbal, and Sung-Ho Bae. Revisiting internal covariate shift for batch normalization. *IEEE Trans. Neural Networks Learn. Syst.*, 32(11):5082–5092, 2021. doi: 10.1109/TNNLS.2020.3026784. URL https://doi.org/10.1109/TNNLS.2020.3026784.

Jeremy Bernstein, Arash Vahdat, Yisong Yue, and Ming-Yu Liu. On the distance between two neural networks and the stability of learning. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/f4b31bee138ff5f7b84ce1575a738f95-Abstract.html.

Johan Bjorck, Carla P. Gomes, Bart Selman, and Kilian Q. Weinberger. Understanding batch normalization. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 7705–7716, 2018. URL https://proceedings.neurips.cc/paper/2018/hash/36072923bfc3cf47745d704feb489480-Abstract.html.

Andy Brock, Soham De, Samuel L. Smith, and Karen Simonyan. High-performance large-scale image recognition without normalization. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 1059–1071. PMLR, 2021. URL http://proceedings.mlr.press/v139/brock21a.html.

Soham De and Sam Smith. Batch normalization biases residual blocks towards the identity function in deep networks. *Advances in Neural Information Processing Systems*, 33:19964–19975, 2020.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2009), 20-25 June 2009, Miami, Florida, USA*, pages 248–255. IEEE Computer Society, 2009. doi: 10.1109/CVPR.2009.5206848. URL https://doi.org/10.1109/CVPR.2009.5206848.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*, pages 1026–1034. IEEE Computer Society, 2015a. doi: 10.1109/ICCV.2015.123. URL https://doi.org/10.1109/ICCV.2015.123.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*, pages 1026–1034. IEEE Computer Society, 2015b. doi: 10.1109/ICCV.2015.123. URL https://doi.org/10.1109/ICCV.2015.123.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016*

IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016, pages 770–778. IEEE Computer Society, 2016a. doi: 10.1109/CVPR.2016.90. URL https://doi.org/10.1109/CVPR.2016.90.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part IV*, volume 9908 of *Lecture Notes in Computer Science*, pages 630–645. Springer, 2016b. doi: 10.1007/978-3-319-46493-0_38.

Alex Henry, Prudhvi Raj Dachapally, Shubham Shantaram Pawar, and Yuxuan Chen. Query-key normalization for transformers. In Trevor Cohn, Yulan He, and Yang Liu, editors, *Findings of the Association for Computational Linguistics: EMNLP 2020, Online Event, 16-20 November 2020*, volume EMNLP 2020 of *Findings of ACL*, pages 4246–4253. Association for Computational Linguistics, 2020. doi: 10.18653/v1/2020.findings-emnlp.379. URL https://doi.org/10.18653/v1/2020.findings-emnlp.379.

Elad Hoffer, Ron Banner, Itay Golan, and Daniel Soudry. Norm matters: efficient and accurate normalization schemes in deep networks. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 2164–2174, 2018. URL https://proceedings.neurips.cc/paper/2018/hash/a0160709701140704575d499c997b6ca-Abstract.html.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Francis R. Bach and David M. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 448–456. JMLR.org, 2015.

Alex Krizhevsky. Learning multiple layers of features from tiny images. pages 32–33, 2009. URL https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In Peter L. Bartlett, Fernando C. N. Pereira, Christopher J. C. Burges, Léon Bottou, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States*, pages 1106–1114, 2012. URL https://proceedings.neurips.cc/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html.

Jaehoon Lee, Lechao Xiao, Samuel S. Schoenholz, Yasaman Bahri, Roman Novak, Jascha Sohl-Dickstein, and Jeffrey Pennington. Wide neural networks of any depth evolve as linear models under gradient descent. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 8570–8581, 2019. URL https://proceedings.neurips.cc/paper/2019/hash/0d1a9651497a38d8b1c3871c84528bd4-Abstract.html.

Zhiyuan Li and Sanjeev Arora. An exponential learning rate schedule for deep learning. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL https://openreview.net/forum?id=rJg8TeSFDH.

Kyle Luther. Why batch norm causes exploding gradients. Blog post, 2020. URL https://kyleluther.github.io/2020/02/18/batchnorm-exploding-gradients.html.

OpenAI. GPT-4 technical report. *CoRR*, abs/2303.08774, 2023. doi: 10.48550/arXiv.2303.08774. URL https://doi.org/10.48550/arXiv.2303.08774.

Jeffrey Pennington, Samuel Schoenholz, and Surya Ganguli. Resurrecting the sigmoid in deep learning through dynamical isometry: theory and practice. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/d9fc0cdb67638d50f411432d0d41d0ba-Paper.pdf.

B.T. Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational*

Mathematics and Mathematical Physics, 4(5):1–17, 1964. ISSN 0041-5553. doi: https://doi.org/10.1016/0041-5553(64)90137-5. URL https://www.sciencedirect.com/science/article/pii/0041555364901375.

PyTorch. BatchNorm2d; PyTorch 2.1 documentation — pytorch.org. https://pytorch.org/docs/stable/generated/torch.nn.BatchNorm2d.html. [Accessed 23-11-2023].

Tim Salimans and Diederik P. Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In Daniel D. Lee, Masashi Sugiyama, Ulrike von Luxburg, Isabelle Guyon, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, page 901, 2016. URL https://proceedings.neurips.cc/paper/2016/hash/ed265bc903a5a097f61d3ec064d96d2e-Abstract.html.

Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. How does batch normalization help optimization? In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 2488–2498, 2018. URL https://proceedings.neurips.cc/paper/2018/hash/905056c1ac1dad141560467e0a99e1cf-Abstract.html.

Andrew M Saxe, James L McClelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv preprint arXiv:1312.6120*, 2013.

Sam Shleifer, Jason Weston, and Myle Ott. Normformer: Improved transformer pretraining with extra normalization. *CoRR*, abs/2110.09456, 2021. URL https://arxiv.org/abs/2110.09456.

Leslie N. Smith and Nicholay Topin. Super-convergence: Very fast training of residual networks using large learning rates. *CoRR*, abs/1708.07120, 2017. URL http://arxiv.org/abs/1708.07120.

Ryszard Szwarc. Convergence analysis of a quotient of two sequences $x_{n+1}^2 = x_n^2 + \frac{c}{x_n^2}$. Mathematics Stack Exchange. URL https://math.stackexchange.com/q/4820434. URL:https://math.stackexchange.com/q/4820434 (version: 2023-12-05).

Twan van Laarhoven. L2 regularization versus batch and weight normalization. *CoRR*, abs/1706.05350, 2017a. URL http://arxiv.org/abs/1706.05350.

Twan van Laarhoven. L2 regularization versus batch and weight normalization. *CoRR*, abs/1706.05350, 2017b. URL http://arxiv.org/abs/1706.05350.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5998–6008, 2017. URL https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html.

Andreas Veit, Michael Wilber, and Serge Belongie. Residual networks behave like ensembles of relatively shallow networks. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, page 550–558, Red Hook, NY, USA, 2016. Curran Associates Inc. ISBN 9781510838819.

Ruosi Wan, Zhanxing Zhu, Xiangyu Zhang, and Jian Sun. Spherical motion dynamics: Learning dynamics of normalized neural network using SGD and weight decay. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 6380–6391, 2021. URL https://proceedings.neurips.cc/paper/2021/hash/326a8c055c0d04f5b06544665d8bb3ea-Abstract.html.

Xiaoxia Wu, Rachel Ward, and Léon Bottou. Wngrad: Learn the learning rate in gradient descent. *CoRR*, abs/1803.02865, 2018. URL http://arxiv.org/abs/1803.02865.

Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Chen Xing, Huishuai Zhang, Yanyan Lan, Liwei Wang, and Tie-Yan Liu. On layer normalization in the

transformer architecture. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 10524–10533. PMLR, 2020. URL http://proceedings.mlr.press/v119/xiong20b.html.

Greg Yang, Jeffrey Pennington, Vinay Rao, Jascha Sohl-Dickstein, and Samuel S. Schoenholz. A mean field theory of batch normalization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL https://openreview.net/forum?id=SyMDXnCcF7.

Yang You, Igor Gitman, and Boris Ginsburg. Scaling SGD batch size to 32k for imagenet training. *CoRR*, abs/1708.03888, 2017. URL http://arxiv.org/abs/1708.03888.

Yang You, Jing Li, Sashank J. Reddi, Jonathan Hseu, Sanjiv Kumar, Srinadh Bhojanapalli, Xiaodan Song, James Demmel, Kurt Keutzer, and Cho-Jui Hsieh. Large batch optimization for deep learning: Training BERT in 76 minutes. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL https://openreview.net/forum?id=Syx4wnEtvH.

A. Additional Proofs

This section contains proofs of theorems as well as additional material complementary to the main section.

Proof of Proposition 3.1. We compute the derivative of the output with regard to the input using the chain rule and relate it to the derivative with unscaled inputs:

$$\frac{dN}{dx}(x,\gamma W) = \frac{dN}{dL}(x,\gamma W)\cdot\frac{dL}{dx}(x,\gamma W) \tag{34}$$

$$= \frac{1}{\gamma}\frac{dN}{dL}(x,W)\cdot\gamma\frac{dL}{dW}(x,W) \tag{35}$$

$$= \frac{dN}{dx}(x,W). \tag{36}$$

Where in Eq. 35, we used the inverse scaling property from Eq. 1. □

Proof of Theorem 3.6. We would like to credit the original author of this proof (Szwarc). By setting $x_i \coloneqq \frac{\sigma_j^2(t_i)}{c_j\lambda}$ and $y_i \coloneqq \frac{\sigma_k^2(t_i)}{c_k\lambda}$, we can rewrite Eq. 4 for layers j and k as two sequences obeying the same recurrence relation:

$$x_{i+1} = x_i + \frac{1}{x_i} \tag{37}$$

$$y_{i+1} = y_i + \frac{1}{y_i}. \tag{38}$$

Raising x_i to the second power yields:

$$x_{i+1}^2 = x_i^2 + 2 + \frac{1}{x_i^2}. \tag{39}$$

This allows us to unroll the recursion as follows :

$$x_i^2 = x_1^2 + 2(i-1) + \frac{1}{u_1^2} + \ldots + \frac{1}{u_{i-1}^2}. \tag{40}$$

As $x_j \geq 2(j-1)$, we can write the following inequality:

$$2(i-1) \leq x_i^2 \leq 2(i-1) + x_1^2 + \frac{1}{x_1^2} + \frac{1}{2} + \frac{1}{4} + \ldots + 2(i-2). \tag{41}$$

By the integral test, it is clear that $\sum_{i=1}^{n-1}\frac{1}{k} \leq ln(n)$ and therefore $\sum_{i=1}^{n-1}\frac{1}{2k} \leq \frac{ln(n)}{2} = ln(\sqrt{n})$. Let be $\gamma \coloneqq u_1^2 + \frac{1}{u_1^2} - 2$, we consider the square root of the expression above:

$$\sqrt{2i-2} \leq x_i \leq \sqrt{2i + log(\sqrt{i-1}) + \gamma}. \tag{42}$$

Since γ is a constant and $\lim_{i\to\infty}\frac{log(i)}{i} = 0$, it follows that:

$$\lim_{i\to\infty}\frac{x_i}{\sqrt{2i}} = 1. \tag{43}$$

and analogously

$$\lim_{i\to\infty}\frac{y_i}{\sqrt{2i}} = 1. \tag{44}$$

We therefore obtain:

$$\lim_{i\to\infty}\frac{x_i}{y_i} = \frac{\sigma_j^2 c_k}{\sigma_k^2 c_j}(t_i) = R_{kj}(t_i) = 1, \tag{45}$$

which is in turn also true for the inverse fraction. □

Proposition A.1 (Ratios Flip at Most Once)**.** *Let $j,k \leq L$ be any layer pair with w.l.o.g. $R_{jk}(t_i) > 1$ and assume a constant learning rate $\lambda(t_i) = \lambda$.*

1. *If effective learning rate ratios do not flip between a given time step and the next, they will never flip at a later time step, i.e. if $R_{jk}(t_{i+1}) > 1$ it follows that $R_{jk}(t_{i+j}) > 1$ for all $j \geq 1$.*

2. *If effective learning rate ratios do flip between a given time step and the next, they will never flip again at a later time step,, i.e. if $R_{jk}(t_{i+1}) < 1$ it follows that $R_{jk}(t_{i+j}) < 1$ for all $j \geq 1$.*

Proof. We start by showing the first statement. Assuming that the effective learning rate ratio does not flip between time steps t_i and t_{i+1}, we know by Lemma 3.9 that $\lambda < \kappa_{jk}(t_i)$. We now just have to show that $\lambda < \kappa_{jk}$ for all successive time steps. Since c_j and c_k are constants and we know by the definition of the discrete process in Eq. 5 that all weight norms $\sigma(t_i)$ are strictly increasing over time, we can write:

$$\lambda < \kappa_{jk}(t_i) = \frac{\sigma_j\sigma_k}{\sqrt{c_j c_k}}(t_i) < \frac{\sigma_j\sigma_k}{\sqrt{c_j c_k}}(t_{i+j}) = \kappa_{jk}(t_{i+j}) \tag{46}$$

for all $j \geq 1$ and thus by Lemma 3.9 the ratio will never flip again.
We now show the second statement. Assuming that the effective learning rate ratio does flip between time steps t_i and t_{i+1}, we know by Lemma 3.9 that $\lambda > \kappa_{jk}(t_i)$. We start by showing that the ratio will not flip for the next time step, which is in turn equivalent to $\lambda < \kappa_{jk}(t_{i+1})$. We can

expand this term as follows:

$$\kappa_{jk}^2(t_{i+1}) = \frac{\sigma_j^2\sigma_k^2}{c_j c_k}(t_{i+1}) \tag{47}$$

$$= \frac{1}{c_j c_k}\left(\sigma_j^2 + \frac{c_j^2\lambda^2}{\sigma_j^2}\right)\left(\sigma_k^2 + \frac{c_k^2\lambda^2}{\sigma_k^2}\right)(t_i) \tag{48}$$

$$= \left(\frac{\sigma_j^2\sigma_k^2}{c_j c_k} + \frac{\sigma_j^2 c_k\lambda^2}{\sigma_k^2 c_j} + \frac{\sigma_k^2 c_j\lambda^2}{\sigma_j^2 c_k} + \frac{c_j c_k\lambda^4}{\sigma_j^2\sigma_k^2}\right)(t_i) \tag{49}$$

$$= \left(\kappa_{jk}^2 + \frac{E_k}{E_j}\lambda^2 + \frac{E_j}{E_k}\lambda^2 + \frac{1}{\kappa_{jk}^2}\lambda^4\right)(t_i) \tag{50}$$

$$> \left(\kappa_{jk}^2 + \frac{E_k}{E_j}\lambda^2 + \frac{E_j}{E_k}\lambda^2 + \frac{1}{\lambda^2}\lambda^4\right)(t_i) \tag{51}$$

$$\geq \lambda^2. \tag{52}$$

We can write Eq. 51 because of the assumption that $\lambda > \kappa_{jk}(t_i)$ and Eq. 52 because all summands are non-negative. We have therefore shown that $\lambda < \kappa_{jk}(t_{i+1})$ and thus the ratio will not flip between time step t_{i+1} and t_{i+2}. By the first proposition shown above, we know it will therefore never flip in future time steps, i.e. $R_{jk}(t_{i+j}) < 1$ for all $j \geq 1$. □

B. Additional Experiments

In Figure 6, we repeated the experiment of Figure 5 on the Cifar100 dataset. Qualitatively, we observe the same effects. The ResNet110 NoShort does not train at all and has high ELR spreads. By using the ELR-constrain or critical-warmup method, we are able to train the network to a significant, but not very good performance. As for the ResNet110 Short, we start to see a difference between runs without warm-up our sub-critical scheduler for high learning rates $\lambda > 10$) where again, a reduced spread results in increased trainability. We conclude that reducing ELR spread correlates with increased trainability, but other factors (e.g. vanishing dimensionalty) explain the gap between short and no-short architectures.

C. Other Factors Influencing Weight Dynamics

As mentioned in the main paper, some techniques commonly used in training influence the evolution of weight dynamics in a way that is not modeled by Eq. 5; in this section we will discuss them.

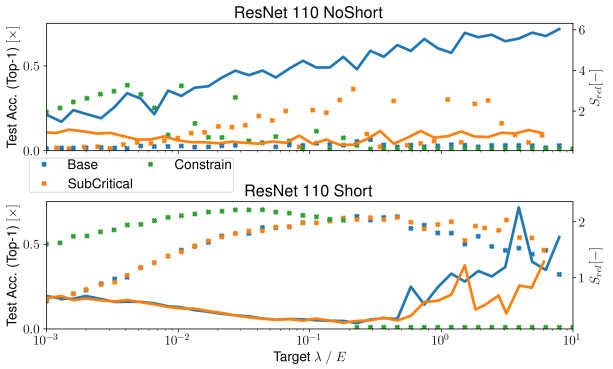


Figure 6. Test accuracies and relative spread of a ResNet110 (No)Short trained on Cifar100 using regular, warm-up and constrained ELR training protocols.

C.1. Weight Decay

The fact that weight decay influences weight dynamics in normalized networks is quite trivial and well-explored in recent literature: (Hoffer et al., 2018) (van Laarhoven, 2017b). In a normalized network, if all weights are reduced by a factor α, this corresponds to an increase of the global learning rate by a factor α, as per Eq 1.

C.2. Momentum

As momentum SGD (Polyak, 1964) modifies each gradient's direction and length before it is applied, it is easy to see that it must influence weight dynamics. It is possible to compute weight dynamics of a network optimized with momentum SGD, but we consider this to be out of scope of this work.

C.3. Affine Normalization Parameters

Normalization layers are usually applied with learnable affine parameters $\gamma \cdot N(x) + \beta$ that are initialized to $\gamma = 1$ and $\beta = 0$ (PyTorch). In the case of a network where normalization layers are followed by ReLUs (this is the case in our experiments), this means that we initialize in the "maximum curvature region" of the nonlinearity but drift away from it during training (Ali Mehmeti-Göpel et al., 2021) leading to gradients dropping further than expected. In Figure 7, we repeated the experiment of Figure 3 using random gradients (a setting that produces a reliable prediction) but add affine BatchNorm parameters to our training protocol. For $\lambda = 0.001$, the prediction is still quite accurate but for $\lambda = 1$, we see that the real gradients are much smaller than predicted.

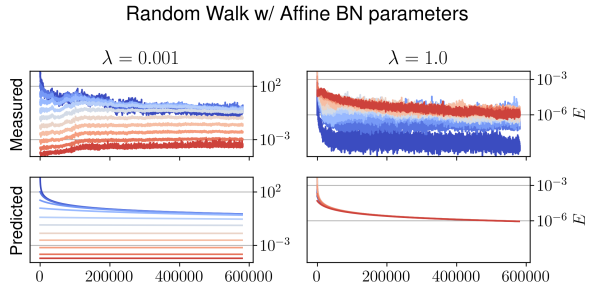


Figure 7. Long term evolution of simulated/real layer-wise effective learning rates for a ResNet56 NoShort trained with random gradients and affine BatchNorm parameters.

D. Architecture and Training Details

As described in the main paper, we used ResNet variants with varying hyper-parameters, with and without skip connections. Architectural details can be found in the tables below.

The experiments in the paper were made on computers running Arch Linux, Python 3.11.5, PyTorch Version 2.1.2+cu121. Various Nvidia GPUS were used ranging from GeForce GTX 1080TI, GeForce GTX 2080Ti RTX 4090.

Table 2. Network architecture and training regime used for the Cifar10/100 task.

ARCHITECTURE	RESNET56/110
BLOCK	BASICBLOCK V1
NUM. BLOCKS	9 9 9 / 18 18 18
NUM. PLANES	16 32 64
SHORTCUT TYPE	A (PADDING)

TRAINING	CIFAR-10 / CIFAR-100
EPOCHS	200
SCHEDULER	MULTISTEP ($\gamma = 0.1$)
MILESTONES	100, 150
LEARNING RATE	VARIABLE
BATCH SIZE	256
OPTIMIZER	SGD
MOMENTUM	0
WEIGHT DECAY	0
AUGMENTATION	RANDOM FLIP
NESTEROV	FALSE

Table 3. Network architecture and training regime used for the ImageNet task.

ARCHITECTURE	RESNET101
BLOCK	BOTTLENECKBLOCK V1
NUM. BLOCKS	3 4 32 3
NUM. PLANES	64 128 256 512
SHORTCUT TYPE	B (1X1-CONV+BN)

TRAINING	IMAGENET
EPOCHS	50
SCHEDULER	ONECYCLE/ NO-WARMUP + COSINE/ SUBCRITICAL + COSINE
MAX. LR	0.4
BATCH SIZE	100
OPTIMIZER	SGD
NESTEROV	FALSE
MOMENTUM	0
WEIGHT DECAY	0
AUGMENTATION	RANDOM FLIP
ONECYCLE ANNEALSTRATEGY	COSINE
ONECYCLE BASEMOMENTUM	0
ONECYCLE CYCLEMOMENTUM	TRUE
ONECYCLE DIVFACTOR	20
ONECYCLE EPOCHSSTART	0.1
ONECYCLE FINALDIVFACTOR	2000
ONECYCLE MAXMOMENTUM	0.0

Bibliography

[1]Rohan Anil, Sebastian Borgeaud, Yonghui Wu, et al. “Gemini: A Family of Highly Capable Multimodal Models”. In: *CoRR* abs/2312.11805 (2023). arXiv: 2312.11805 (cit. on p. 1).

[2]Sanjeev Arora, Zhiyuan Li, and Kaifeng Lyu. “Theoretical Analysis of Auto Rate-Tuning by Batch Normalization”. In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019 (cit. on p. 14).

[3]David Balduzzi, Marcus Frean, Lennox Leary, et al. “The Shattered Gradients Problem: If resnets are the answer, then what is the question?” In: *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*. Ed. by Doina Precup and Yee Whye Teh. Vol. 70. Proceedings of Machine Learning Research. PMLR, 2017, pp. 342–350 (cit. on pp. 25, 26).

[4]David Balduzzi, Marcus Frean, Lennox Leary, et al. “The Shattered Gradients Problem: If resnets are the answer, then what is the question?” In: *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*. Ed. by Doina Precup and Yee Whye Teh. Vol. 70. Proceedings of Machine Learning Research. PMLR, 2017, pp. 342–350 (cit. on pp. 27, 28, 30).

[5]Peter L. Bartlett, Nick Harvey, Christopher Liaw, and Abbas Mehrabian. “Nearly-tight VC-dimension and Pseudodimension Bounds for Piecewise Linear Neural Networks”. In: *J. Mach. Learn. Res.* 20 (2019), 63:1–63:17 (cit. on p. 21).

[6]Minsu Cho, Ameya Joshi, Brandon Reagen, Siddharth Garg, and Chinmay Hegde. “Selective Network Linearization for Efficient Private Inference”. In: *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*. Ed. by Kamalika Chaudhuri, Stefanie Jegelka, Le Song, et al. Vol. 162. Proceedings of Machine Learning Research. PMLR, 2022, pp. 3947–3961 (cit. on p. 24).

[7]Anna Choromanska, Mikael Henaff, Michaël Mathieu, Gérard Ben Arous, and Yann LeCun. “The Loss Surfaces of Multilayer Networks”. In: *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2015, San Diego, California, USA, May 9-12, 2015*. Ed. by Guy Lebanon and S. V. N. Vishwanathan. Vol. 38. JMLR Workshop and Conference Proceedings. JMLR.org, 2015 (cit. on p. 2).

[8]Thomas M. Cover. “Geometrical and Statistical Properties of Systems of Linear Inequalities with Applications in Pattern Recognition”. In: *IEEE Trans. Electron. Comput.* 14.3 (1965), pp. 326–334 (cit. on p. 21).

[9]George Cybenko. “Approximation by superpositions of a sigmoidal function”. In: *Math. Control. Signals Syst.* 2.4 (1989), pp. 303–314 (cit. on p. 21).

[10]Hadi Daneshmand, Jonas Moritz Kohler, Francis R. Bach, Thomas Hofmann, and Aurélien Lucchi. “Batch normalization provably avoids ranks collapse for randomly initialised deep networks”. In: *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*. Ed. by Hugo Larochelle, Marc’Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin. 2020 (cit. on p. 27).

[11]Soham De and Samuel L. Smith. “Batch Normalization Biases Residual Blocks Towards the Identity Function in Deep Networks”. In: *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*. Ed. by Hugo Larochelle, Marc’Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin. 2020 (cit. on p. 27).

[12]Amir Ben Dror, Niv Zehngut, Avraham Raviv, et al. “Layer Folding: Neural Network Depth Reduction using Activation Linearization”. In: *CoRR* abs/2106.09309 (2021). arXiv: 2106.09309 (cit. on pp. 22, 24, 29, 32, 112).

[13]Xavier Glorot and Yoshua Bengio. “Understanding the difficulty of training deep feedforward neural networks”. In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2010, Chia Laguna Resort, Sardinia, Italy, May 13-15, 2010*. Ed. by Yee Whye Teh and D. Mike Titterington. Vol. 9. JMLR Proceedings. JMLR.org, 2010, pp. 249–256 (cit. on p. 26).

[14]Yiwen Guo, Anbang Yao, and Yurong Chen. “Dynamic Network Surgery for Efficient DNNs”. In: *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*. Ed. by Daniel D. Lee, Masashi Sugiyama, Ulrike von Luxburg, Isabelle Guyon, and Roman Garnett. 2016, pp. 1379–1387 (cit. on p. 29).

[15]Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. “Deep Residual Learning for Image Recognition”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*. IEEE Computer Society, 2016, pp. 770–778 (cit. on pp. 6, 29).

[16]Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. “Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification”. In: *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*. IEEE Computer Society, 2015, pp. 1026–1034 (cit. on p. 22).

[17]Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. “Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification”. In: *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*. IEEE Computer Society, 2015, pp. 1026–1034 (cit. on p. 26).

[18]Sepp Hochreiter and Jürgen Schmidhuber. “Flat Minima”. In: *Neural Comput.* 9.1 (1997), pp. 1–42 (cit. on p. 2).

[19]Elad Hoffer, Ron Banner, Itay Golan, and Daniel Soudry. “Norm matters: efficient and accurate normalization schemes in deep networks”. In: *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*. Ed. by Samy Bengio, Hanna M. Wallach, Hugo Larochelle, et al. 2018, pp. 2164–2174 (cit. on p. 13).

[20]Andrew G. Howard, Menglong Zhu, Bo Chen, et al. “MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications”. In: *CoRR* abs/1704.04861 (2017). arXiv: 1704.04861 (cit. on p. 24).

[21]Lei Huang, Dawei Yang, Bo Lang, and Jia Deng. “Decorrelated Batch Normalization”. In: *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*. Computer Vision Foundation / IEEE Computer Society, 2018, pp. 791–800 (cit. on p. 111).

[22]Lei Huang, Yi Zhou, Fan Zhu, Li Liu, and Ling Shao. “Iterative Normalization: Beyond Standardization Towards Efficient Whitening”. In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*. Computer Vision Foundation / IEEE, 2019, pp. 4874–4883 (cit. on p. 111).

[23]Zehao Huang and Naiyan Wang. “Data-Driven Sparse Structure Selection for Deep Neural Networks”. In: *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part XVI*. Ed. by Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss. Vol. 11220. Lecture Notes in Computer Science. Springer, 2018, pp. 317–334 (cit. on p. 29).

[24]Ahmed Imtiaz Humayun, Randall Balestriero, and Richard G. Baraniuk. “Deep Networks Always Grok and Here is Why”. In: *CoRR* abs/2402.15555 (2024). arXiv: 2402.15555 (cit. on p. 24).

[25]Sergey Ioffe and Christian Szegedy. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. In: *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*. Ed. by Francis R. Bach and David M. Blei. Vol. 37. JMLR Workshop and Conference Proceedings. JMLR.org, 2015, pp. 448–456 (cit. on p. 27).

[26]John Jumper, Richard Evans, Alexander Pritzel, et al. “Highly accurate protein structure prediction with AlphaFold”. In: *Nature* 596.7873 (2021), pp. 583–589 (cit. on p. 1).

[27]Patrick Kidger and Terry J. Lyons. “Universal Approximation with Deep Narrow Networks”. In: *Conference on Learning Theory, COLT 2020, 9-12 July 2020, Virtual Event [Graz, Austria]*. Ed. by Jacob D. Abernethy and Shivani Agarwal. Vol. 125. Proceedings of Machine Learning Research. PMLR, 2020, pp. 2306–2327 (cit. on p. 21).

[28]Alex Krizhevsky. “One weird trick for parallelizing convolutional neural networks”. In: *CoRR* abs/1404.5997 (2014). arXiv: 1404.5997 (cit. on p. 18).

[29]Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. “ImageNet Classification with Deep Convolutional Neural Networks”. In: *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States*. Ed. by Peter L. Bartlett, Fernando C. N. Pereira, Christopher J. C. Burges, Léon Bottou, and Kilian Q. Weinberger. 2012, pp. 1106–1114 (cit. on p. 1).

[30]Twan van Laarhoven. “L2 Regularization versus Batch and Weight Normalization”. In: *CoRR* abs/1706.05350 (2017). arXiv: 1706.05350 (cit. on p. 15).

[31]Yann LeCun, Léon Bottou, Genevieve B. Orr, and Klaus-Robert Müller. “Efficient BackProp”. In: *Neural Networks: Tricks of the Trade - Second Edition*. Ed. by Grégoire Montavon, Genevieve B. Orr, and Klaus-Robert Müller. Vol. 7700. Lecture Notes in Computer Science. Springer, 2012, pp. 9–48 (cit. on p. 2).

[32]Jaehoon Lee, Lechao Xiao, Samuel S. Schoenholz, et al. “Wide Neural Networks of Any Depth Evolve as Linear Models Under Gradient Descent”. In: *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*. Ed. by Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, et al. 2019, pp. 8570–8581 (cit. on p. 13).

[33]Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. “Visualizing the Loss Landscape of Neural Nets”. In: *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*. Ed. by Samy Bengio, Hanna M. Wallach, Hugo Larochelle, et al. 2018, pp. 6391–6401 (cit. on pp. 6, 13).

[34]Yuanzhi Li and Yang Yuan. “Convergence Analysis of Two-layer Neural Networks with ReLU Activation”. In: *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*. Ed. by Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, et al. 2017, pp. 597–607 (cit. on p. 2).

[35]Seppo Linnainmaa. “The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors”. PhD thesis. Master’s Thesis (in Finnish), Univ. Helsinki, 1970 (cit. on p. 5).

[36]AI @ Meta Llama Team. “The Llama 3 Herd of Models”. In: (2024) (cit. on p. 29).

[37]Ilya Loshchilov and Frank Hutter. “Decoupled Weight Decay Regularization”. In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019 (cit. on p. 18).

[38]Ekdeep Singh Lubana, Robert P. Dick, and Hidenori Tanaka. “Beyond BatchNorm: Towards a Unified Understanding of Normalization in Deep Learning”. In: *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*. Ed. by Marc’Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan. 2021, pp. 4778–4791 (cit. on p. 25).

[39]Kyle Luther. *Why Batch Norm Causes Exploding Gradients*. Blog post. 2020 (cit. on p. 27).

[40]Christian H. X. Ali Mehmeti-Göpel". “About that loss surface”. MA thesis. Johannes-Gutenberg University Mainz, 2019 (cit. on pp. 14, 31).

[41]Christian H. X. Ali Mehmeti-Göpel and Michael Wand. “Spreads in Effective Learning Rates: The Perils of Batch Normalization During Early Training”. In: *CoRR* abs/2306.00700 (2023). arXiv: 2306.00700 (cit. on p. 13).

[42]Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, et al. “NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis”. In: *Computer Vision - ECCV 2020 - 16th European Conference, Glasgow, UK, August 23-28, 2020, Proceedings, Part I*. Ed. by Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm. Vol. 12346. Lecture Notes in Computer Science. Springer, 2020, pp. 405–421 (cit. on p. 1).

[43]OpenAI. “GPT-4 Technical Report”. In: *CoRR* abs/2303.08774 (2023). arXiv: 2303.08774 (cit. on p. 1).

[44]Ricardo Pacilli. “Studying Performance of partially linearized neural networks”. MA thesis. Johannes-Gutenberg University Mainz, 2023 (cit. on p. 24).

[45]Jeffrey Pennington, Samuel S. Schoenholz, and Surya Ganguli. “Resurrecting the sigmoid in deep learning through dynamical isometry: theory and practice”. In: *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*. Ed. by Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, et al. 2017, pp. 4785–4795 (cit. on p. 26).

[46]Nasim Rahaman, Aristide Baratin, Devansh Arpit, et al. “On the Spectral Bias of Neural Networks”. In: *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, 2019, pp. 5301–5310 (cit. on p. 11).

[47]Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. “How Does Batch Normalization Help Optimization?” In: *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*. Ed. by Samy Bengio, Hanna M. Wallach, Hugo Larochelle, et al. 2018, pp. 2488–2498 (cit. on p. 25).

[48]Andrew M. Saxe, James L. McClelland, and Surya Ganguli. “Exact solutions to the nonlinear dynamics of learning in deep linear neural networks”. In: *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2014 (cit. on p. 25).

[49]Andrew M. Saxe, James L. McClelland, and Surya Ganguli. “Exact solutions to the nonlinear dynamics of learning in deep linear neural networks”. In: *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2014 (cit. on pp. 26, 31).

[50]Jürgen Schmidhuber. “Annotated History of Modern AI and Deep Learning”. In: *CoRR* abs/2212.11279 (2022). arXiv: 2212.11279 (cit. on p. 5).

[51]Samuel S. Schoenholz, Justin Gilmer, Surya Ganguli, and Jascha Sohl-Dickstein. “Deep Information Propagation”. In: *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017 (cit. on p. 25).

[52]Steffen Schotthöfer, Emanuele Zangrando, Jonas Kusch, Gianluca Ceruti, and Francesco Tudisco. “Low-rank lottery tickets: finding efficient low-rank neural networks via matrix differential equations”. In: *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*. Ed. by Sanmi Koyejo, S. Mohamed, A. Agarwal, et al. 2022 (cit. on p. 29).

[53]Ashish Vaswani, Noam Shazeer, Niki Parmar, et al. “Attention is All you Need”. In: *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*. Ed. by Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, et al. 2017, pp. 5998–6008 (cit. on p. 18).

[54]Andreas Veit, Michael J. Wilber, and Serge J. Belongie. “Residual Networks Behave Like Ensembles of Relatively Shallow Networks”. In: *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*. Ed. by Daniel D. Lee, Masashi Sugiyama, Ulrike von Luxburg, Isabelle Guyon, and Roman Garnett. 2016, pp. 550–558 (cit. on pp. 9, 23, 27, 32).

[55]Ruosi Wan, Zhanxing Zhu, Xiangyu Zhang, and Jian Sun. “Spherical Motion Dynamics: Learning Dynamics of Normalized Neural Network using SGD and Weight Decay”. In: *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*. Ed. by Marc’Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan. 2021, pp. 6380–6391 (cit. on pp. 13, 14).

[56]Wikipedia. *Exclusive or — Wikipedia, The Free Encyclopedia*. http://en.wikipedia.org/w/index.php?title=Exclusive%20or&oldid=1213385953. [Online; accessed 18-March-2024]. 2024 (cit. on p. 5).

[57]Wikipedia. *Perceptron — Wikipedia, The Free Encyclopedia*. http://en.wikipedia.org/w/index.php?title=Perceptron&oldid=1213384391. [Online; accessed 18-March-2024]. 2024 (cit. on p. 5).

[58]Xiaoxia Wu, Rachel Ward, and Léon Bottou. “WNGrad: Learn the Learning Rate in Gradient Descent”. In: *CoRR* abs/1803.02865 (2018). arXiv: 1803.02865 (cit. on p. 14).

[59]Greg Yang, Jeffrey Pennington, Vinay Rao, Jascha Sohl-Dickstein, and Samuel S. Schoenholz. “A Mean Field Theory of Batch Normalization”. In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019 (cit. on pp. 13, 14).

[60]Greg Yang, Jeffrey Pennington, Vinay Rao, Jascha Sohl-Dickstein, and Samuel S. Schoenholz. “A Mean Field Theory of Batch Normalization”. In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019 (cit. on pp. 15, 27).

[61]Greg Yang and Samuel S. Schoenholz. “Mean Field Residual Networks: On the Edge of Chaos”. In: *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*. Ed. by Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, et al. 2017, pp. 7103–7114 (cit. on p. 32).

[62]Yang You, Igor Gitman, and Boris Ginsburg. “Scaling SGD Batch Size to 32K for ImageNet Training”. In: *CoRR* abs/1708.03888 (2017). arXiv: 1708.03888 (cit. on p. 18).

[63]Yang You, Jing Li, Sashank J. Reddi, et al. “Large Batch Optimization for Deep Learning: Training BERT in 76 minutes”. In: *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020 (cit. on p. 18).

[64]Xin Yu, Zhiding Yu, and Srikumar Ramalingam. “Learning Strict Identity Mappings in Deep Residual Networks”. In: *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*. Computer Vision Foundation / IEEE Computer Society, 2018, pp. 4432–4440 (cit. on p. 27).

[65]Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. "Understanding deep learning requires rethinking generalization". In: *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017 (cit. on pp. 2, 21).

Appendix

A

A.1 Absolute Convergence of Power-Law Functions

Empirically, the w-o spectrum of a layer decays similarly to a power-law. Unlike the DC coefficient of any natural data, a power-law function has a singularity at 0 which we want to avoid. We therefore shift the power-law function by a small ϵ to the left to avoid the singularity. Since the original function is real-valued in time space, we assume a symmetrical magnitude spectrum.

Proposition 1. *Let $P(\omega) \in \mathbb{C}$ be a complex-valued function whose magnitude spectrum obeys a shifted power law, i.e. $|P(\omega)| \sim (\omega + \epsilon)^{-\alpha}$ for constant $\alpha \in \mathbb{R}^+$ and $\epsilon \in \mathbb{R}^+$. The function absolutely converges for $\alpha > 1$, i.e.*

$$\int_{\epsilon}^{\infty} |P(\omega)| d\omega < \infty. \tag{A.1}$$

Proof. For any $\epsilon > 0$, we can write

$$\int_{\epsilon}^{\infty} |\omega^{-\alpha}| d\omega = \left[\frac{\omega^{-a+1}}{-a+1} \right]_{\epsilon}^{\infty} \tag{A.2}$$

$$= \lim_{\omega \to \infty} \frac{\omega^{-\alpha+1}}{-\alpha+1} - \frac{\epsilon^{-\alpha+1}}{-\alpha+1} \tag{A.3}$$

For $\alpha > 1$, the first term approaches 0 as $\omega \to \infty$, making the integral finite. Since $\int_{-\infty}^{\infty} |P(\omega)| d\omega = 2 \int_{\epsilon}^{\infty} |\omega^{-\alpha}| d\omega$, the result follows.

□

A.2 Discrete Auto-Rate Tuning Model Including Momentum

A.2.1 Extended Theory

In this section, we extend the discrete model of Chapter 3 to include the effects of momentum. We consider basic training dynamics following the momentum formula; for simplicity of notation, we consider time step $t_k = k\lambda^2$, and denote $g_k := \nabla W(t_k)$, $\widehat{g}_k := \nabla\widehat{W}(t_k)$ and $w_k := W(t_k)$, where ∇W is the momentum buffer and $\nabla\widehat{W}$ is the "real gradient", before applying the momentum formula but including auto-rate tuning effects.

First, similarly to Equation 3.4, the current gradient magnitude before applying momentum should be inversely proportional to the current layer weights:

$$||\widehat{g}_k||_F = \frac{c}{||w_k||_F}. \tag{A.4}$$

Since SGD with momentum is computed as $g_k = \mu g_{k-1} + \widehat{g}_k$, we can write:

$$g_k = \sum_{i=0}^{k} \mu^{k-i}\widehat{g}_i. \tag{A.5}$$

Assuming all $\widehat{g}_i$ are pairwise orthogonal for all $i \in \{0 \dots k\}$, we can use the Pythagorean theorem and obtain:

$$||g_k||_F^2 = \sum_{i=0}^{k} \mu^{2(k-i)}||\widehat{g}_i||_F^2. \tag{A.6}$$

Finally, we unroll the basic SGD update rule in a similar fashion and substitute Equation A.5:

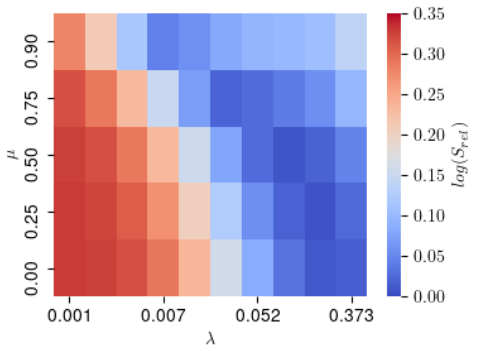


Fig. A.1.: Simulated ELR spread after 50 steps in our discrete model for different values of λ and μ.

$$\begin{aligned} w_k &= w_{k-1} - \lambda g_{k-1} \\ &= w_0 - \lambda \sum_{i=0}^{k-1} g_i \\ &= w_0 - \lambda \sum_{i=0}^{k-1} \sum_{j=0}^{i} \mu^{i-j} \widehat{g}_j \end{aligned} \tag{A.7}$$

By rearranging the terms, we can sum all entries for a fixed j with a geometric sum and obtain:

$$w_k = w_0 - \lambda \sum_{j=0}^{k-1} \widehat{g}_j \cdot \frac{1-\mu^{k-j}}{1-\mu}. \tag{A.8}$$

By additionally assuming that w_0 and $\widehat{g}_i$ are orthogonal for all $i \in \{0 \dots n\}$, we obtain:

$$\begin{aligned} ||w_k||_F^2 &= ||w_0||_F^2 + \lambda^2 \cdot \sum_{j=0}^{k-1} ||\widehat{g}_j||_F^2 \cdot \left(\frac{1-\mu^{k-j}}{1-\mu} \right)^2 \\ &= ||w_0||_F^2 + \lambda^2 \left(||\widehat{g}_{k-1}||_F^2 + \sum_{j=0}^{k-2} \left(\frac{1-\mu^{k-j}}{1-\mu} \right)^2 \cdot ||\widehat{g}_j||_F^2 \right). \end{aligned} \tag{A.9}$$

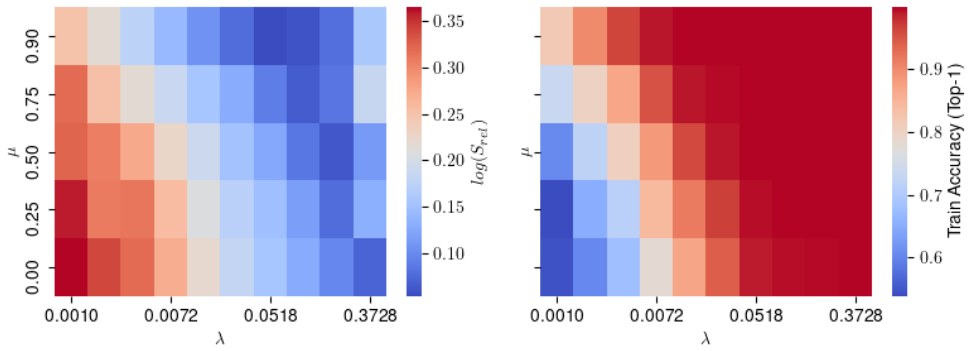


Fig. A.2.: ELR spread averaged over the entire training process (**left**) and training accuracies (**right**) after training a ResNet56 'Short' for 200 Epochs on Cifar10 for different combinations of learning rates λ and momentum parameter values μ.

In order to compute the evolution of the weight and gradient norms, after initialization of the initial values w_0 and $g_0 = \widehat{g_0}$, one should follow the following steps in a loop:

1. Computed the squared gradient norm $||\widehat{g_i}||_F^2$ using Equation A.4.
2. Compute the squared momentum buffer norm $||g_i||_F^2$ using Equation A.6.
3. Compute the squared weight norms $||w_i||_F^2$ using Equation A.9.

In order to obtain model predictions with momentum, we run the simulation outlined above, storing the history of previous weight norms in a table.

Using this model, we computed the ELR spread after 50 steps for a 56-layer network for different values of μ and λ in Figure A.1. We chose 50 steps as it takes a number of steps for momentum to impact ELR spread, especially for small learning rates. We see that overall, using momentum seems to lower the effective learning rate spread. Only for very small values of λ the effect is not clear, a regime where our orthogonality assumptions are unrealistic.

A.2.2 Further Experiments

To confirm our predictions of Figure A.1, we trained a ResNet56 Short on Cifar10 for multiple values of $\lambda \in [0.001, 3]$ and $\mu \in [0, 1]$ and report the measured ELR spread (averaged over the entire training process) and training accuracies in Figure A.2. The exact values for ELR spread cannot be compared

exactly as in Figure A.1 we reported spreads after 50 steps and in Figure A.1 we averaged over the entire training run, but we see that qualitatively our predictions are accurate. For small and medium learning rates, high values for μ result in a smaller ELR spread. Networks with a very high ELR spread (low λ and μ) do indeed seem to train badly and raising μ seems indeed to result in a higher test accuracy for those networks. Overall, the predicted ELR spread seems to qualitatively match reality.

A.3 Evolution of the Output Rank During Training

In Section 7.3 Figure 5, we trained deep feedforward networks (ResNet110 NoShort) for a range of learning rates on Cifar10. In a basic setting, these networks without residual connections will not reach a decent training accuracy for any constant learning rate due to very high spreads in effective learning rates. Using ELR constrain and SubCritical warm-up, we effectively eliminated ELR spread and made these deep feedforward networks trainable once again. In this setting, perhaps surprisingly, we still noticed a significant gap in trainability between residual and feedforward architectures, albeit the increased trainability using residual connections is commonly attributed to reducing exploding gradients, which we already removed in this setup. An explanation for this gap could be vanishing rank in deep feedforward networks (ref. Section 5.1), where the network loses its ability to distinguish different inputs, leading to an uninformative forward pass and reduced trainability. We repeated this experiment whilst monitoring the soft rank of the last activations in the network using the *matrix_rank* function of the PyTorch library, using the default values for tolerance. For that, we flattened the output shape (B, C, H, W) to $(B, C * H * W)$, as we wanted to measure the network's ability to distinguish different inputs. We use the current image batch as input for the calculation: this can entail unwanted effects, since images of the same class can (and should, after some training) have similar representations. This should be addressed in future experiments, for example by using only inputs of different classes for measuring rank and the experiments below should only be regarded as preliminary.

First, we confirm that rank collapse can occur in deep feedforward networks, even when eliminating ELR spread. In Figure A.3, we present the rank measurements of the base and SubCritical warm-up runs for $\lambda = 0.1125$, a learning

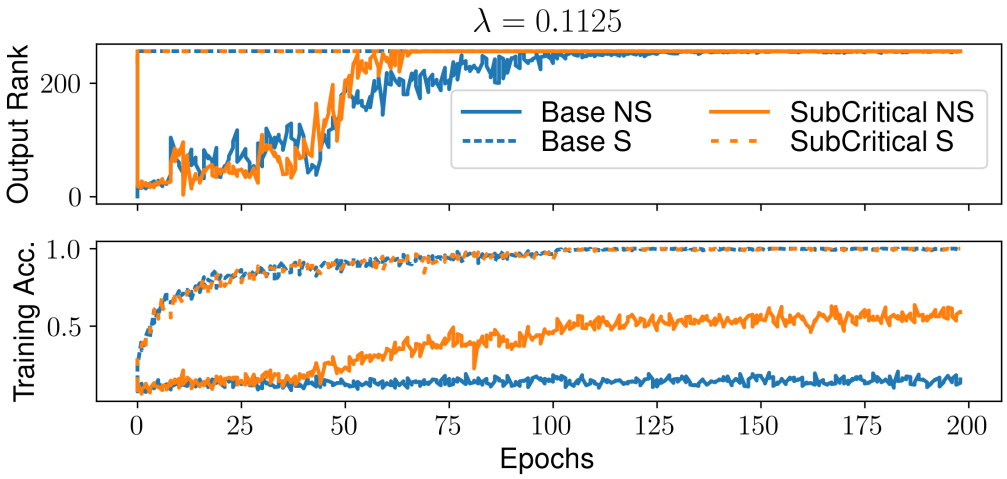


Fig. A.3.: Output rank (**top**) and training performance (**bottom**) of a ResNet 110 (No)Short trained on Cifar10 using different warm-up schedulers: no warm-up (blue) and SubCritical warm-up (orange).

rate where all networks reach at least 50% test performance using warm-up. We observe that all residual networks have full rank during the entire training process. As for the feedforward networks, we measure a full rank at initialization, which then quickly drops to 1 but fully recovers within 50-100 epochs. We clearly see that even for the NoShort network trained with SubCritical warm-up, where high ELR spreads are eliminated by construction, rank collapse could be an issue in early training. We conclude that the use of residual connections, in addition to the well-known effect on gradient disparity, correlates with a high output rank in deep networks, contrary to feedforward architectures.

The training performances between the feedforward and residual networks in Figure A.3 are difficult to compare, as they might train with significantly differing effective learning rates. For this, we repeat the experiment using the ELR constrain method which prescribes a constant layer-wise effective learning rate, ensuring that the runs are comparable in terms of ELR. In Figure A.4, we see that the training performances of the Short and NoShort architectures are now closer together, but we still observe a gap. In this setting, it is difficult to clearly attribute the performance gap to vanishing rank, as the rank vanishes completely but recovers quickly after around 25 epochs of training. To further evaluate whether this performance gap is due to the vanishing rank in the first 25 epochs of training or due to some other beneficial

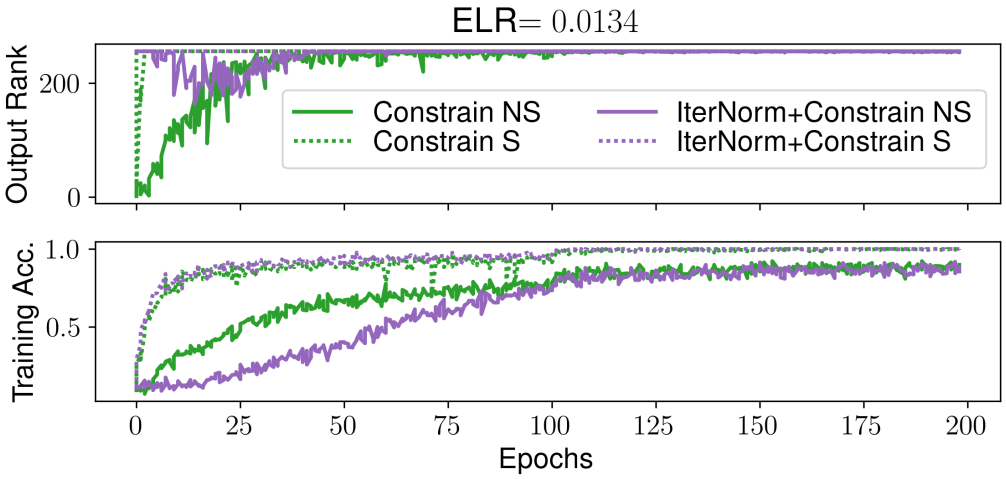


Fig. A.4.: Output rank (**top**) and training performance (**bottom**) of a ResNet 110 (No)Short trained on Cifar10 using ELR Constrain (green) and ELR Constrain + IterNorm (purple).

properties of residual connections, a constructive approach to eliminating rank collapse is needed.

After conducting a short experiment, we find that a normalization layer that simply normalizes the co-variances between channels by computing the SVD of the co-variance matrix (also called *PCA norm* in literature) does not train at all when used in deep feedforward networks. This is in line with literature claiming that this is due to the ambiguity of the SVD with regard to rotation, leading to a different transform at every step purely based on batch statistics [21]. We found a PyTorch implementation of *IterNorm*, a recently proposed efficient approximate whitening layer, which uses Newton's method to efficiently approximate the SVD [22]. In Figure A.4, we employed IterNorm using the pre-set default hyperparameters along with *ELR Constrain* to see if the rank stabilizes and accuracy improves. For the feedforward architecture, we do see an improvement on the rank. The fact that we still observe a drop-off in early training can be explained by the fact that IterNorm only performs an approximate whitening. However, the training speed seems worse than before, as is the test performance (80.6% for IterNorm+Constrain, 81.9% for Constrain). This can be due to specifics in how IterNorm is implemented, hyperparameter choice, or simply an unwanted side-effect of a full whitening normalization layer. More research would be needed to understand whether we can obtain further performance improvements on deep feedforward networks without ELR spread by maintaining a high output rank during training.

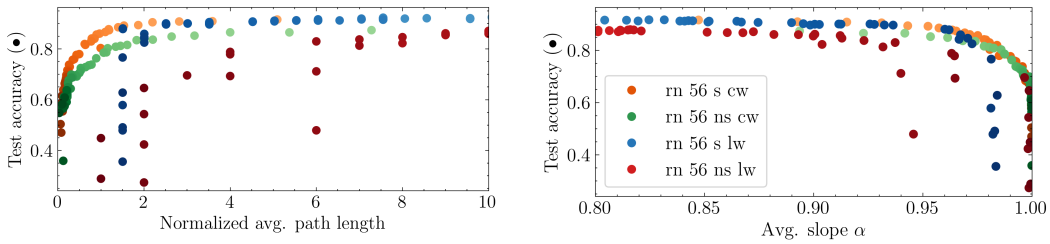


Fig. A.5.: Average path length (**right**) and average slope α (**left**) plotted against the test performance for a ResNet 56 (No)Short after a post-training partial linearization comparing the channel-wise ('cw') and layer-wise ('lw') linearization protocol.

A.4 Layer-Wise Partial Linearization

In this Section, we recreate the experiment from Section 7.2 Figure 5, comparing the test performance of a ResNet56 (No)Short architecture after post-training linearization using two different techniques: channel-wise (as in all experiments of our paper) and layer-wise (similar to Dror et al. [12]) i.e. there is only a single PReLU weight per layer. In Figure A.5 left, inline with the results of Dror et al., we see that for the feedforward architecture using the layer-wise linearization technique, it is not possible to reach a lower depth than 4 without suffering massive accuracy losses. For the same architecture, when using a channel-wise approach, it is possible to reach a much lower average depth. The effect is less pronounced, but similar, for the residual architecture.

It is worth mentioning that the channel-wise architecture has slightly more parameters than the layer-wise approach per construction, but for higher NAPL values, we see that there is not much difference in accuracies. To see if the above result might be a simple side effect of the discretization in measuring NAPL, i.e. only fully linear channels are counted as linear, we plot the average slope α on the right of Figure A.5. This seems not to be the case, as the networks linearized using the layer-wise technique tend to have lower average α values (i.e. less linear) than the networks linearized using the channel-wise method.

List of Figures

Colophon

This thesis was typeset with LaTeX 2_ε. It uses the *Clean Thesis* style developed by Ricardo Langner. The design of the *Clean Thesis* style is inspired by user guide documents from Apple Inc.

Download the *Clean Thesis* style at `http://cleanthesis.der-ric.de/`.

Adaptations to the style of the Institute of Computer Science can be found at `https://gitlab.rlp.net/institut-fur-informatik/cleanthesis-jgu`.

Declaration

I hereby declare that I have written the present thesis independently and without use of other than the indicated means. I also declare that to the best of my knowledge all passages taken from published and unpublished sources have been referenced. The thesis has not been submitted for evaluation to any other examining authority, nor has it been published in any form whatsoever. I duly noted the Regulations for Good Scientific Practice and Dealing with Scientific Misconduct.

The external tools used are Grammarly (https://app.grammarly.com/) to find and correct grammatical mistakes, and ChatGPT (https://openai.com/chatgpt/) version 4 and 4o to rephrase blocks of text of approximately 1-3 sentences.

Mainz, October 18, 2024

Christian H. X. Ali
Mehmeti-Göpel

Christian Ali Mehmeti-Göpel

Rektor-Forestier-Str. 10 – 55122 Mainz – Germany

+49 1732886023 • chalimeh@uni-mainz.de

Nationality

German

Education

École de la Rhonelle — **Artres(France)**
Primary school — *1999-2004*

Collège Carpeaux — **Valenciennes (France)**
Diplôme national des Collèges — *2004–2008*
Middle school

Conservatoire de Valenciennes — **Valenciennes (France)**
Musical Education, grade : good — *2004–2008*
Classical musical education in musical theory, piano and the clarinet.

Lycée de l'Éscaut — **Valenciennes (France)**
Baccalauréat Scientifique, grade : very good — *2008–2011*
High school. Focus on scientific subjects, especially mathematics.

Higher Education

Johannes Gutenberg-Universität — **Mainz (Germany)**
Bachelor of Science in Mathematics, Minor in Philosophy — *2011–2016*

- Concentrated on Numerical Methods, providing a robust foundation in mathematical problem-solving and computational techniques.
- Bachelor's thesis titled "PageRank Applied to Neuroscience and Genetics", exploring innovative applications of the PageRank algorithm beyond its traditional use.
- Philosophy studies centered on Theory of Mind and Mental Representations, providing a unique perspective critical for philosophical considerations in AI system development.

Universitat de Barcelona — **Barcelona (Spain)**
Erasmus Exchange Programme — *2014-2015*

Johannes Gutenberg-Universität — **Mainz (Germany)**
Master of Science in Computer Science, Minor in Mathematics — *2016-2019*

- Delved into Optimization and Coding/Information Theory in Mathematics, and Machine Learning/Data Mining in Computer Science, essential for understanding Deep Learning theory.
- Master's thesis on Deep Learning theory: "What does a good loss surface look like?", an in-depth study about the effects of architecture design on loss surface characteristics in neural network training.

Università degli Studi di Firenze **Florence (Italy)**
Erasmus Exchange Programme *2017-2018*

Johannes Gutenberg-Universität **Mainz (Germany)**
Ph.D. in Computer Science / Deep Learning (Dr. rer. nat.) *2019-2024 (Graduation EOY)*

- Specializing in Deep Learning theory with an emphasis on neural network architecture design, exploring the impact of popular architectural choices such as batch normalization and residual connections on the shape of the loss surface, expressivity and its impact on trainability.
- Dissertation focuses on a mathematical analysis of neural network architectures, seeking to understand and optimize the function-space characteristics of neural networks for enhanced trainability whilst conserving expressivity.

Teaching Experience

2017-2019: Teaching Assistant in Complexity Theory, Efficient Algorithms, and Data Structures

- Assisted in delivering course content and provided tutoring, including exercise correction and conducting tutorials.

2019-2024: Teaching Assistant in Programming (Python), Software Development (C++), Advanced Computer Graphics (C++), Mathematical Modeling, and Deep Learning (PyTorch)

- Created and delivered course material in Python and C++, focusing on both practical and theoretical aspects of software development.
- Guided students through advanced computer graphics and mathematical modeling courses, applying complex concepts in practical scenarios such as creating a path tracer from scratch.

Technical Skills

Programming Languages: Python, C++, Java, R, Matlab

Frameworks and Libraries: PyTorch, Pandas, NumPy, SciPy, Matplotlib

Tools and Technologies: Git, SQL, LaTeX, Markdown, Linux

Language Skills

German: Mother tongue *Complete bilingual education*

French: Mother tongue *Complete bilingual education*

English: Fluent *Everyday use for work*

Italian: Fluent *Completed language courses and lived one year in Italy*

Spanish: Fluent *Completed language courses and lived one year in Spain*

Catalan: Basic *Completed language courses and lived one year in Catalunya*