Average Path Length: Sparsification of Nonlinearities Creates Surprisingly Shallow Networks

Christian H.X. Ali Mehmeti-Göpel
Jan Disselhoff

Institute of Computer Science
Johannes-Gutenberg University Mainz
Staudingerweg 9, 55122 Mainz, Germany
chalameh@uni-mainz.de, jadissel@uni-mainz.de

Abstract

We perform an empirical study of the behaviour of deep networks when pushing its activation functions to become fully linear in some of its feature channels through a sparsity prior on the overall number of nonlinear units in the network. To measure the "depth" of the resulting partially linearized network, we compute the average number of active nonlinearities encountered along a path in the network graph. In experiments on CNNs with sparsified PReLU s on typical image classification tasks, we make several observations: Under sparsity pressure, 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. We consistently observe a slow decay of performance with depth until the onset of a rapid collapse in accuracy, allowing for surprisingly shallow networks at moderate losses in accuracy that outperform base-line networks of similar depth, even after increasing width to a comparable number of parameters. In terms of training, we observe a nonlinear advantage: Reducing nonlinearity after training leads to a better performance than before, in line with previous findings in linearized training, but with a gap depending on task difficulty that vanishes for easy problems.

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, Sutskever, and Hinton 2012) only used moderate depth, improvements such as batch normalization (Ioffe and Szegedy 2015), Santurkar et al. 2018, Arora, Li, and Lyu (2019) or residual connections (He et al. 2015a) 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 networks depth (along with its width) is known to upper bound the complexity of the function that it can represent (Bartlett et al. 2019) and therefore upper bounds the networks expressivity. Indeed, deeper networks are found to enhance performance (Tan and 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, Li, and Liang 2019), more computationally expensive to train or infer and suffer from numerous stability issues such as vanishing (Hochreiter 1991), exploding (Zhang, Dauphin, and Ma 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, Gabriel, and Hongler 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 very simple setup that associates every nonlinear unit with a cost at channel granularity, which can be raised continuously, while simultaneously trying to maintain performance. This is done by replacing each ReLU layer with channel-wise PReLU activations (He et al. 2015b) and regularizing 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. Observing the partially linearized network can tell us how many nonlinear units are needed by a given network and also where they are needed. We measure the average amount of nonlinear units a given input traverses until it is classified disregarding subsequent linear mappings along computational paths, as these do not increase expressivity in a nonlinear sense.

Using tool, we make a series of experiments on common convolutional network architectures on standard computer vision tasks by partially linearizing networks after regular training, make the following observations:

- The amount of nonlinear units in the network can be reduced considerably before suffering larger performance losses and where the transition occurs depends on the difficulty of the task to learn.
- The average amount of nonlinear units encountered along a path in the computation graph ("effective depth") of
the resulting networks is very low before performance collapses.

- When applying the same regularization strength to networks of varying width and depth, the effective width and depth is similar for all resulting partially linearized networks for a fixed task.
- A network partially linearized at a later stage of training outperforms the same network partially linearized earlier. The biggest difference is notable in the early training phase.

These findings indicate, consistent with earlier hypothesis (Frankle and Carbin 2019) that there is a core nonlinear structure in networks that forms during the first epochs of training whose shape depends on the task to solve and is largely independent of the shape of initial network chose.

Related Work

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 and 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, coming to a similar finding of a problem-difficulty-dependent minimal set, embedded in a much larger and deeper network, when considering nested nonlinear computations. Wen et al. (2016) find sparse subnetworks using different types of weight-regularization on a ResNet; in particular the authors find a shallower subnetwork that can beat the performance of an even deeper network. Our method, aiming at removing nonlinear feature channels from the network, is also tightly related to the classical dropout (Nesterov 1983) method as well as its modern successors such as ShakeDrop (Yamada et al. 2019).

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 wrt. 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, Gabriel, and Hongler 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 and Izmailov 2020). Maybe unsurprisingly, linearized training hurts performance in practice (Fort et al. 2020) and theory. Roberts, Yaida, and Hanin (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 loss in performance when linearizing early in training. Within the NTK framework, the impact of ReLUs can be captured by path kernels (Lakshminarayanan and Singh 2020), the learning of which improves results and generalizes when retraining. 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 approach is different, using (channel-wise) regularization as an analytical tool to understand the emergent structure within a complex nonlinear network better.

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 \( L_{0.5} = \sum \frac{1}{5} | \alpha_i |^{0.5} \) to the regular training loss scaled with a regularization weight \( \omega \), where \( \alpha_i \) is the variable slope of the i-th PReLU. Since this loss is discontinuous at 1, we completely remove a PReLU unit if their slope gets close enough to one. We call such a unit inactive, while all other units are active.

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 traditional notion of depth corresponds to the amount of activation layers encountered when following the computation graph of a network from input to output. For networks where nonlinearity is removed in certain channels, and thus the amount of activation layers encountered highly differs according to the path taken through the network, considering the average amount of active units encountered per path might be a more informative choice.

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 denote \( V_0 \subset V \) 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 \( \epsilon \) is the weight of the PReLU.

Let \( P^{(n)} \subset V^n \) be the set of all paths of length \( n \) in \( G \) that originate in \( v_{in} \), i.e. \( p_i \in v_{in} \) 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 \( p \in P^{(n)} \) 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 \( \phi^{(l)}(v) := |\{ p \in P^{(l)} \mid p_{l_1} = v, \phi(p) = l \}, 1 \leq l' \leq l \} \) as the number of paths from \( v_{in} \) to \( v \) of effective length \( l \). We finally define the average path length (APL) of the network as \( APL(G) := \sum_{v \in V} \phi^{(l)}(v) d(v_{in}, v) \), where \( d \) is the depth of the network. In order to effectively compute the APL of a network, we resort to dynamic programming.

**Proposition 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 \( v \):

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

A proof can be found in the Appendix. In simple terms, we obtain the path histogram of a given node by summing the path histograms of all incoming nodes and shifting it "to the right" if a node contains an active ReLU.

**Implementation details:** This recursion can easily be implemented 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 have to be ignored. We finally just need to choose a constant \( 1/0/1/\ldots \) 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 can be found 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.

**Experiments**

In this section, we apply linearization to trained network architectures on different datasets and observe the resulting structure to see if we can find some regularities in their behavior that are consistent over different datasets and architectures - both by visual inspection and statistical analysis.

As our techniques require networks with ReLU activations, we chose a ResNet (He et al. 2016) and PyramidNet (Han, Kim, and Kim 2017) with ("Short") and without ("NoShort") residual connections as examples of standard architectures with and without residual connections with differently distributed feature channels. For ease of comparison with related work, 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 and Yang 2015) and ILSVRC 2012 (called ImageNet in the following) (Deng et al. 2009). CIFAR-10 is used by default if not further specified for experiments requiring many runs. We trained all networks from scratch except on ImageNet where we use a pre-trained ResNet50 from the Torchvision library.

We chose the most basic setup for regular training that is capable of delivering benchmark results for the chosen architectures: regular training with ReLU units, momentum SGD, a multistep learning-rate scheduler and weight decay. Afterwards, the ReLU units were replaced by regularized PReLU units (initial negative slope 0) and we resumed training in a shorter post-training step. Concerning learning-rate scheduling in the post-training step, we needed 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 reverted to the initial learning rate and use the same multistep scheduling as in the regular training phase adapted to the shorter post-training phase. More details about architectures and training regimes used can be found in the Appendix. 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 in-line with previous works discussing path lengths in ResNets (Veit, Wilber, and Belongie 2016). Results with unnormalized path length yield similar results albeit the absolute numbers are higher as shown in the Appendix.
Visual Inspection of the Remaining Nonlinear Feature Channels

We start with informal observations about how reducing the amount of nonlinear feature channels in a network affects what they react to, in order to gain an intuition about how the resulting networks might behave. We visualized some feature channels from ResNet50 Short trained on ImageNet and partly linearized to different degrees ($\omega \in [0.0005, 0.0025]$) with the Lucent (Kiat 2021) library; the performance and amount of remaining active PReLU units in these networks is shown in Figure 5 and discussed in later sections. We chose some arbitrary feature channels from a deeper layer that are still active in all three networks. We see that in most cases, we can recognize a given feature over the different networks. Generally, the feature visualizations seem subjectively sharper for higher $\omega$, but sometimes mix with others or change drastically. We conclude that our method is not simply selecting feature channels but also changes what input they react to - with an in intensity depending on the regularization weight used.

Performance and Structure of Partially Linearized Networks

We now observe the temporal evolution of the linearization process for two different network architectures in Figure 3. When evaluating the proportion of inactive PReLU units 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 PReLU units 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 gradient effects as a cause. The fact that for the ResNet56 NoShort, many layers are fully linearized without explicit insensitive to do so indicates 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 par-
Figure 4: NAPL and test accuracy for different partially linearized network architectures with $\omega \in [0.0005, 0.005]$. Darker colors represent a higher regularization weight and the disk sizes represent the global proportion of inactive PReLU units. 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. Qualitatively similar plots can be found for the CIFAR-100 dataset in the Appendix. We further verified our claims for a ResNet50 on the ImageNet dataset in Figure 5; note that this network contains a non-ReLU activation layer (maximum pooling) that we included in our calculations.

Third, we want to investigate whether a partially linearized network of a given average depth can outperform a shallow network of similar (absolute) depth. For this, we indicated the performance of a ResNet6 and a ResNet6 wide with at least as many parameters as the ResNet56 trained for 200 epochs as horizontal lines for the Cifar10 runs. Comparing the performance of the partially linearized ResNet56 to the ResNet6 wide, we note that the performance of the partially linearized deep network is superior to the performance of shallow networks of comparable depth, hinting that linearizing after training can be advantageous. Since NAPL is only an average measure of depth, a network with NAPL $n$ is not strictly comparable to a network with true depth $n$ since for the latter, the number of nonlinear units on a computation path is strictly inferior to $n$; we address this limitation in a later section.

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.

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 PReLU units 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.

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 8, 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). 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 the CIFAR-100 in the Appendix show qualitatively the same behaviour but with different NAPL values.

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 $\omega$ on CIFAR-10, CINIC-10 and CIFAR-100. Looking at Figure 9.
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 $\omega$. 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.

The Nonlinear Advantage

In Figure 4, we observed that a partially linearized deep network can outperform a shallow network of comparable depth: this hints the existence of a "nonlinear advantage" for partially linearized networks in the sense that linearizing at a later stage of training is more advantageous compared to an earlier stage. In this section, we want to make sure that the observed effect is strictly due to the linearization later on in training and cannot only be attributed to architecture differences between the two networks. For this, we compare the performance of networks with the same architecture that are partially linearized at different stages of training.

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 at initialization. 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. This way, we are able to see whether the exact layerwise distribution of active PReLU units is important for the network to train to its full potential. We re-train the network 5 times, using the same number of epochs and schedule as in the original training scheme. Apart from the mask of inactive PReLU units, everything else (network weights, optimizer etc.) is re-initialized and the network is trained from scratch.

- **Exact**: The exact binary masks of inactive PReLU are kept.
- **Layer-Wise Permutation**: The binary masks of inac-
We slightly modified our regularizer to stop when a goal percentage (we chose 80%, a value high enough to impact training performance) of inactive PReLU units over all layers is reached. We see that even when using this method, networks regularized later in training are still significantly more performant than networks partially linearized earlier in training. This effect is particularly pronounced for the harder dataset Tiny ImageNet, indicating a correlation between effect strength and the hardness of the task at hand.

### Discussion and Future Work

In our experiments, we found that when naively penalizing nonlinearity equally for each feature channel in the network, surprisingly shallow structures form for networks without residual connections, indicating that their actual depth isn’t fully utilized. We also found that we can extract a more performant shallow network when partially linearizing after training compared to training a similarly linear network from scratch, further supporting our claim that fewer nonlinearity is needed than previously thought.Reducing combinatorial complexity in this way might help us gain new insights in network optimization processes, given that many theoretical works on network optimization are limited to shallow networks.

We also found strong indications that during training, neural networks form a small “core nonlinear structure” that is hidden within the network, confirming previous observations such as the “lottery ticket hypothesis” and “nonlinear advantage” from another angle. We further found indications that the shape of this structure is strongly dependent on the problem on hand.

Finally, we used a feature visualization library to view the features of linearized networks and interestingly we found that features that can be recognized across networks seem to have subjectively more clearly defined shapes and increased sharpness. The reduced complexity of linearized networks combined with this fact might indicate that our approach can support efforts to reach explainable AI.
References

Allen-Zhu, Z.; Li, Y.; and Liang, Y. 2019. 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, 6155–6166.

Arora, S.; Li, Z.; and Lyu, K. 2019. 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.

Baldazzi, D.; Frean, M.; Leary, L.; Lewis, J. P.; Ma, K. W.; and McWilliams, B. 2017. 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, 342–350. PMLR.

Bartlett, P. L.; Harvey, N.; Liaw, C.; and Mehrabian, A. 2019. Nearly-tight VC-dimension and Pseudodimension Bounds for Piecewise Linear Neural Networks. J. Mach. Learn. Res., 20: 63:1–63:17.

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

Cybenko, G. 1989. Approximation by superpositions of a sigmoidal function. Math. Control. Signals Syst., 2(4): 303–314.

Darlow, L. N.; Crowley, E. J.; Antoniou, A.; and Storkey, A. J. 2018. CINIC-10 is not ImageNet or CIFAR-10. CoRR, abs/1810.03505.

Deng, J.; Dong, W.; Socher, R.; Li, L.; Li, K.; and Fei-Fei, L. 2009. 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, 248–255. IEEE Computer Society.

Dror, A. B.; Zehngut, N.; Raviv, A.; Artyomov, E.; Vitek, R.; and Jevnisek, R. J. 2021. Layer Folding: Neural Network Depth Reduction using Activation Linearization. CoRR, abs/2106.09309.

Fort, S.; Dziugaite, G. K.; Paul, M.; Kharaghani, S.; Roy, D. M.; and Ganguli, S. 2020. 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.

Frankle, J.; and Carbin, M. 2019. 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.

Han, D.; Kim, J.; and Kim, J. 2017. Deep Pyramidal Residual Networks. In 2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017, 6307–6315. IEEE Computer Society.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2015a. 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, 1026–1034. IEEE Computer Society.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2015b. 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, 1026–1034. IEEE Computer Society.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. 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, 630–645. Springer.

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

Ioffe, S.; and Szegedy, C. 2015. 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, 448–456. JMLR.org.

Jacot, A.; Gabriel, F.; and Hongler, C. 2018. Neural Tangent Kernel: Convergence and Generalization in Neural Networks. volume 31, 8571–8580. Curran Associates, Inc.

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

Krizhevsky, A. 2009. Learning Multiple Layers of Features from Tiny Images. 32–33.

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In Hinton, G. E.; Larochelle, H.; Sejnowski, T. J.;ürmäe, Y.; and colleagues, eds., Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems(NeurIPS) Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States, 1106–1114.

Lakshminarayanan, C.; and Singh, A. V. 2020. 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.

Le, Y.; and Yang, X. S. 2015. Tiny ImageNet Visual Recognition Challenge.

Neill, J. O. 2020. An overview of neural network compression. arXiv preprint arXiv:2006.03669.
Nesterov, Y. 1983. A method for solving the convex programming problem. *Proceedings of the USSR Academy of Sciences*, 269: 543–547.

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

Santurkar, S.; Tsipras, D.; Ilyas, A.; and Madry, A. 2018. 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*, 2488–2498.

Tan, M.; and Le, Q. V. 2019. 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*, 6105–6114. PMLR.

Veit, A.; Wilber, M. J.; and Belongie, S. J. 2016. 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*, 550–558.

Wen, W.; Wu, C.; Wang, Y.; Chen, Y.; and Li, H. 2016. Learning Structured Sparsity in Deep Neural Networks. Wilson, A. G.; and Izmailov, P. 2020. Bayesian deep learning and a probabilistic perspective of generalization. *Advances in neural information processing systems*, 33: 4697–4708.

Yamada, Y.; Iwamura, M.; Akiba, T.; and Kise, K. 2019. Shakedown Regularization for Deep Residual Learning. *IEEE Access*, 7: 186126–186136.

Zhang, H.; Dauphin, Y. N.; and Ma, T. 2019. 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.
Appendix

Histogram Computation

In this section, we provide proof of Proposition 1 in the main paper and discuss further the need for normalization when computing the average path length.

Proof 1 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.

Normalizing the Histogram

Veit, Wilber, and Belongie (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, Wilber, and Belongie (2016).

Plots for APL Instead of NAPL

In this section, we repeat the experiments of Figure 4, 6 and 7 showing APL instead of NAPL. In Figure 16, 14 and 15 we see fundamentally no difference in the results except for all unnormalized path length values being consistently higher than the normalized ones.

Experiments on Cifar100

In Figure 20, 17, 18 and 19 we repeated the experiments of Figure 5, 4, 6 and 7 on CIFAR-100 and see qualitatively the same behavior although all measured NAPL value are consistently higher than on CIFAR-10.

APL of Networks with Constant Width

In Figure 8 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 21, we repeated the same experiment with a simplistic Toy-Net having constant width and no striding after the first layer and see the pattern disappear completely. We conclude that the observed pattern is an artifact of scaling ResBlocks of different width and striding operations in the network.

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 / CIFAR-100 runs. We used shortcut option “A” (padding) for all networks except in Section 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 \( \nu \). For Figure 6 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 Figure 21 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.

(Post-)Training Hyperparameters & 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 22 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 pre-trained models from torchvision model-zoo. For post-training, the hyper-parameters in Figure 23 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.
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: APL and test accuracy for different network architectures with regularization weight $\omega \in [0.0005, 0.005]$.

Figure 17: 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 18: Validation accuracy and NAPL during linearization of ResNets of different depth for $\omega = 0.003$ on CIFAR-100.

Figure 19: Validation accuracy and NAPL during linearization of ResNets56 of different width for $\omega = 0.003$ on CIFAR-100.

Figure 20: Proportion of inactive PReLU's when linearizing a ResNet56 NoShort / Short with $\omega = 0.003$ on CIFAR-100.
Figure 21: Average path length of Toy-Nets of different depth after post-training linearization for $\omega = 0.003$. 
Training CIFAR-10 / CINIC-10 / CIFAR-100

| Parameter      | Value                  |
|---------------|------------------------|
| 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

| Parameter      | Value                  |
|---------------|------------------------|
| 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 22: Details of the training regime.

Post-Training CIFAR-10 / CINIC-10 / CIFAR-100

| Parameter      | Value                  |
|---------------|------------------------|
| 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

| Parameter      | Value                  |
|---------------|------------------------|
| 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 23: Details of the post-training regime.