Layer rotation: a surprisingly simple indicator of generalization in deep networks?

Simon Carbonnelle [1] **Christophe De Vleeschouwer** [1]

Abstract

Our work presents empirical evidence that layer rotation, i.e. the evolution across training of the cosine distance between each layer's weight vector and its initialization, constitutes an impressively consistent indicator of generalization performance. Compared to previously studied indicators of generalization, we show that layer rotation has the additional benefit of being easily monitored and controlled, as well as having a network-independent optimum: the training procedures during which all layers' weights reach a cosine distance of 1 from their initialization consistently outperform other configurations -by up to 20% test accuracy. Finally, our results also suggest that the study of layer rotation can provide a unified framework to explain the impact of weight decay and adaptive gradient methods on generalization.

1 Introduction

In order to understand the intriguing generalization properties of deep neural networks highlighted by [23, 34, 16], the identification of numerical indicators of generalization performance that remain applicable across a diverse set of training settings is critical. A well-known and extensively studied example of such indicator is the width of the minima the network has converged to [12, 16].

In this paper, we present empirical evidence supporting the discovery of a novel indicator of generalization: **the evolution across training of the cosine distance between each layer's weight vector and its initialization** (denoted by *layer rotation*). Indeed, we show across a diverse set of experiments (with varying datasets, networks and training procedures), that larger layer rotations (i.e. larger cosine distance between final and initial weights of each layer) consistently translate into better generalization performance. In addition to providing an original perspective on generalization, our experiments suggest that layer rotation also benefits from the following properties compared to alternative indicators of generalization:

- It is easily monitored and, since it only depends on the evolution of the network's weights, can be controlled along the optimization through appropriate weight update adjustments
- It has a network-independent optimum (all layers reaching a cosine distance of 1)
- It provides a unified framework to explain the impact of weight decay and adaptive gradient methods on generalization.

In comparison, other indicators usually provide a metric to optimize (*e.g.* the wider the minimum, the better) but no clear optimum to be reached (what is the optimal width?), nor a precise methodology to tune it (how to converge to a minimum with a specific width?). By disclosing simple guidelines to tune layer rotations and an easy-to-use controlling tool, our work can also help practitioners get the best out of their network with minimal hyper-parameter tuning.

The presentation of our experimental study is structured according to three successive steps:

1. Development of tools to monitor and control layer rotation (Section 2);
2. Systematic study of layer rotation configurations in a controlled setting (Section 3);
3. Study of layer rotation configurations in standard training settings, with a special focus on SGD, weight decay and adaptive gradient methods (Section 4).

Related work is discussed in Supplementary Material.

2 Monitoring and controlling layer rotation

This section describes the tools for monitoring and controlling layer rotation during training, such as its relation with generalization can be studied in Sections 3 and 4.

2.1 Monitoring with layer rotation curves

Layer rotation is defined as the evolution of the cosine distance between each layer's weight vector and its initialization during training. More precisely, let w_l^t be the flattened weight tensor of the l^{th} layer at optimization step

[1]ICTEAM, Université catholique de Louvain, Louvain-La-Neuve, Belgium. <simon.carbonnelle@uclouvain.be>.

t (t_0 corresponding to initialization), then the rotation of layer l at training step t is defined as the cosine distance between $w_l^{t_0}$ and w_l^t. [1] In order to visualize the evolution of layer rotation during training, we record how the cosine distance between each layer's current weight vector and its initialization evolves across training steps. We denote this visualization tool by *layer rotation curves* hereafter.

2.2 Controlling with Layca

The ability to control layer rotations during training would enable a systematic study of its relation with generalization. Therefore, we present Layca (LAYer-level Controlled Amount of weight rotation), an algorithm where the layer-wise learning rates directly determine the amount of rotation performed by each layer's weight vector during each optimization step (the *layer rotation rates*), in a direction specified by an optimizer (SGD being the default choice). Inspired by techniques for optimization on manifolds [1], and on spheres in particular, Layca applies layer-wise orthogonal projection and normalization operations on SGD's updates, as detailed in Algorithm 1 in Supplementary Material. These operations induce the following simple relation between the learning rate $\rho_l(t)$ of layer l at training step t and the angle $\theta_l(t)$ between w_l^t and w_l^{t-1}: $\rho_l(t) = tan(\theta_l(t))$.

Our controlling tool is based on a strong assumption: that controlling the amount of rotation performed during each individual training step (i.e. the layer rotation rate) enables control of the cumulative amount of rotation performed since the start of training (i.e. layer rotation). This assumption is not trivial since the aggregated rotation is a priori very dependent on the structure of the loss landscape. As will be attested by the inspection of the layer rotation curves, our assumption however appeared to be sufficiently valid, and the control of layer rotation was effective in our experiments.

3 A systematic study of layer rotation configurations with Layca

Section 2 provides tools to monitor and control layer rotation. The purpose of this section is to use these tools to conduct a systematic experimental study of layer rotation configurations. We adopt SGD as default optimizer, but use Layca (cfr. Algorithm 1) to vary the relative rotation rates (faster rotation for first layers, last layers, or no prioritization) and the global rotation rate value (high or low rate, for all layers). The experiments are conducted on five different tasks which vary in network architecture and dataset complexity, and are further described in Table 1.

[1] It is worth noting that our study focuses on weights that multiply the inputs of a layer (*e.g.* kernels of fully connected and convolutional layers).

[2] References: VGG [26], ResNet [10], torch blog [32], Wide ResNet [33], CIFAR-10 [18], Tiny ImageNet [5, 4]. Dropout layers were removed from the torch blog CNN to enable perfect classification on the training set (100% accuracy).

3.1 Layer rotation rate configurations

Layca enables us to specify layer rotation rate configurations by setting the layer-wise learning rates. To explore the large space of possible layer rotation rate configurations, our study restricts itself to two directions of variation. First, we vary the initial global learning rate $\rho(0)$, which affects the layer rotation rate of all the layers. During training, the global learning rate $\rho(t)$ drops following a fixed decay scheme (hence the dependence on t), as is common in the literature (cfr. Supp. Mat. A.6). The second direction of variation tunes the relative rotation rates between different layers. More precisely, we apply static, layer-wise learning rate multipliers that exponentially increase/decrease with layer depth (which is typical of exploding/vanishing gradients). The multipliers are parametrized by the layer index l (in forward pass ordering) and a parameter $\alpha \in [-1, 1]$ such that the learning rate of layer l becomes:

$$\rho_l(t) = \begin{cases} (1-\alpha)^{5\frac{(L-1-l)}{L-1}}\rho(t) & \text{if} \quad \alpha > 0 \\ (1+\alpha)^{5\frac{l}{L-1}}\rho(t) & \text{if} \quad \alpha \leq 0 \end{cases} \tag{1}$$

Values of α close to -1 correspond to faster rotation of first layers, 0 corresponds to uniform rotation rates, and values close to 1 to faster rotation of last layers. Visualization of the layer-wise multipliers for different α values is provided in Supplementary Material.

3.2 Study of the relation between layer rotation and generalization

Figure 1a depicts the layer rotation curves (cfr. Section 2.1) and the corresponding test accuracies obtained with different layer rotation rate configurations. While each configuration solves the classification task on the training data ($\approx 100\%$ training accuracy in all configurations, cfr. Supp. Mat.), we observe huge differences in generalization ability (differences of up to 30% test accuracy). More importantly, these differences in generalization ability seem to be tightly connected to differences in layer rotations. In particular, we extract the following rule of thumb that is applicable across the five considered tasks: the larger the layer rotations, the better the generalization performance. The best performance is consistently obtained when nearly all layers reach the largest possible distance from their initialization: a cosine distance of 1 (cfr. fifth column of Figure 1a).

This observation would have limited value if many configurations (amongst which the best one) lead to cosine distances of 1. However, we notice that most configurations do not. In particular, rotating the layers weights very slightly is sufficient for the network to achieve 100% training accuracy (cfr. third column of Figure 1a)). Moreover, one could imagine training procedures with large layer rotations that do not generalize well, e.g. if large rotations are performed in a

Table 1: Summary of the tasks used for our experiments[2]

Name	Architecture	Dataset
C10-CNN1	VGG-style 25 layers deep CNN	CIFAR-10
C100-resnet	ResNet-32	CIFAR-100
tiny-CNN	VGG-style 11 layers deep CNN	Tiny ImageNet
C10-CNN2	deep CNN from torch blog	CIFAR-10 + data augm.
C100-WRN	Wide ResNet 28-10 with 0.3 dropout	CIFAR-100 + data augm.

random direction. It is indeed necessary that the rotations performed coincide with improvements in the training error. In particular, configurations with too high layer rotation rates can prevent training from happening, thereby escaping the scope of our rule of thumb (cfr. Figure 3).

4 A study of layer rotation in standard training settings

Section 3 uses Layca to study the relation between layer rotations and generalization in a controlled setting. This section investigates the layer rotation configurations that naturally emerge when using SGD, weight decay or adaptive gradient methods for training. First of all, these experiments will provide supplementary evidence for the rule of thumb proposed in Section 3. Second, we'll see that studying training methods from the perspective of layer rotation can provide useful insights to explain their behaviour.

The experiments are performed on the five tasks of Table 1. The learning rate parameter is tuned independently for each training setting through grid search over 10 logarithmically spaced values ($3^{-7}, 3^{-6}, ..., 3^{2}$), except for C10-CNN2 and C100-WRN where learning rates are taken from their original implementations when using SGD + weight decay, and from [29] when using adaptive gradient methods for training. The test accuracies obtained in standard settings are compared to the best results obtained with Layca, as provided in the 5th column of Figure 1a.

4.1 Analysis of SGD and weight decay

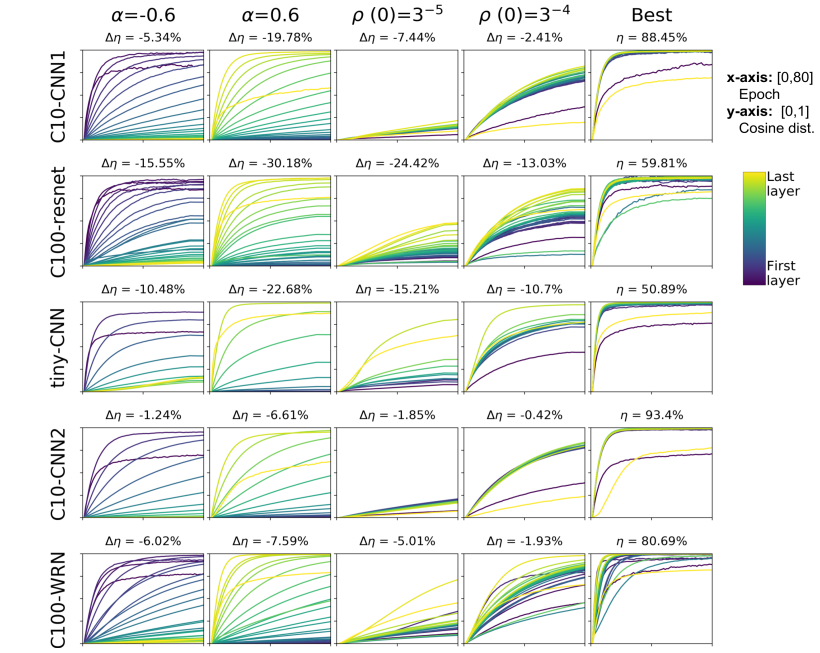
Figure 1b (1^{st} line) depicts the layer rotation curves and the corresponding test accuracies generated by SGD for each of the five tasks. We observe that the curves are far from the ideal scenario disclosed in Section 3, where the majority of the layers' weights reached a cosine distance of 1 from their initialization. Moreover, in accordance with our rules of thumb, SGD reaches a considerably lower test performance than Layca. Extensive tuning of the learning rate did not help SGD to solve its two systematic problems: 1) layer rotations are not uniform and 2) the layers' weights stop rotating before reaching a cosine distance of 1.

Several papers have recently shown that, in batch normalized networks, the regularization effect of weight decay was caused by an increase of the effective learning rate [28, 13, 35]. More generally, reducing the norm of weights increases the amount of rotation induced by a given training step. It is thus interesting to see how weight decay affects layer rotations, and if its impact on generalization is coherent with our rule of thumb. Figure 1b (2^{nd} line) displays, for the 5 tasks, the layer rotation curves generated by SGD when combined with weight decay (in this case, equivalent to L_2-regularization). We observe that SGD's problems are solved: all layers' weights are rotated synchronously and reach a cosine distance of 1 from their initialization. Moreover the observations confirm our rule of thumb: the resulting test performances are on par with the ones obtained with Layca.

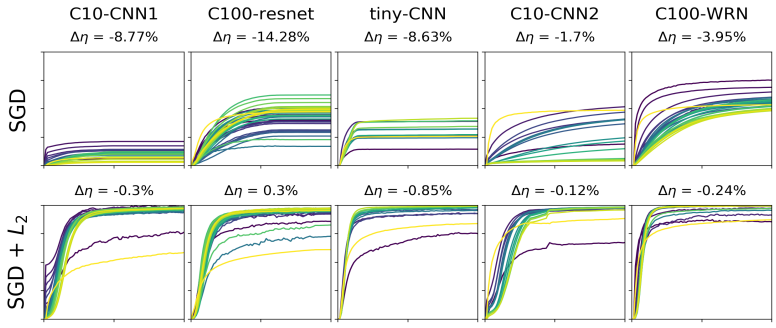
4.2 Analysis of adaptive gradient methods

The recent years have seen the rise of adaptive gradient methods in the context of machine learning (*e.g.* RMSProp [27], Adagrad [6], Adam [17]). Initially introduced for improving training speed, [29] observed that these methods also had a considerable impact on generalization. Since these methods affect the rate at which individual parameters change, they might also influence layer rotations. We will thus verify if their influence on generalization is coherent with our rule of thumb.

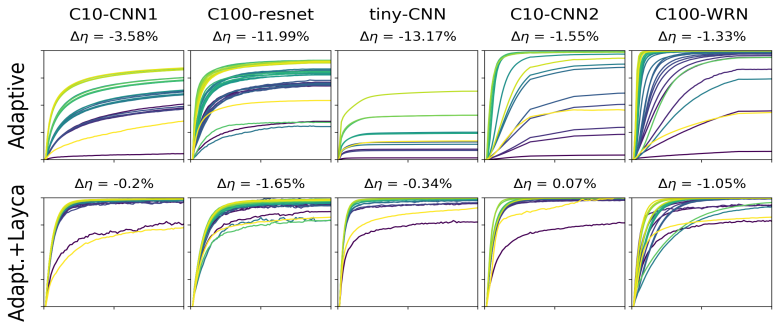
Figure 1c (1^{st} line) provides the layer rotation curves and test accuracies obtained when using adaptive gradient methods to train the 5 tasks described in Table 1. We observe an overall worse generalization ability compared to Layca's optimal configuration and small and/or non-uniform layer rotations. We also observe that the layer rotations of adaptive gradient methods are considerably different from the ones induced by SGD (cfr. Figure 1b). For example, adaptive gradient methods seem to induce larger rotations of the last layers' weights, while SGD usually favors rotation of the first layers' weights. Could these differences explain the impact of parameter-level adaptivity on generalization in deep learning? In Figure 1c (2^{nd} line), we show that when Layca is used on top of adaptive methods (to control layer rotation), adaptive methods can reach test accuracies on par with SGD + weight decay. Our observations thus offer a novel perspective on adaptive gradient methods' poor generalization properties, and provide supplementary evidence for our rule of thumb.



(a)



(b)



(c)

Figure 1: Analysis of the layer rotation curves and test accuracies (η) induced by different training settings on the five tasks of Table 1. $\Delta\eta$ is computed with respect to Layca's best configuration (last column of (a)). Colour code and axes are provided in the upper right. Training accuracies are provided in Supplementary Material ($\approx 100\%$ in all configurations). **Overall, the visualizations unveil large differences in generalization ability across configurations which seem to follow a simple yet consistent rule of thumb: the larger the layer rotation for each layer, the better the generalization performance.** **(a)** Layca training with layer-wise learning rates parametrized by α and $\rho(0)$ (cfr. Section 3.1). **(b)** SGD training without (1^{st} line) or with (2^{nd} line) weight decay, all with gridsearch-optimized learning rates. **(c)** Training with adaptive gradient methods (RMSProp, Adam, Adagrad, RMSProp+L_2 and Adam+L_2 respectively for each task/column) without (1^{st} line) and with (2^{nd} line) control of layer rotation by Layca, all with gridsearch-optimized learning rates.

ACKNOWLEDGEMENTS

Thanks to the anonymous NeurIPS and ICLR reviewers for their helpful feedback on previous versions of this paper. Special thanks to the reddit r/MachineLearning community for enabling outsiders to stay up to date with the last discoveries and discussions of our fast moving field.

Simon and Christophe are Research Fellows of the Fonds de la Recherche Scientifique – FNRS, which provided funding for this work.

References

[1] P.-A. Absil, R. Mahony, and R. Sepulchre. Optimization On Manifolds : Methods And Applications. In *Recent Advances in Optimization and its Applications in Engineering*, pages 125—-144. Springer, 2010.

[2] Peter L Bartlett. The Sample Complexity of Pattern Classification with Neural Networks : The Size of the Weights is More Important than the Size of the Network. *IEEE transactions on Information Theory*, 44(2):525–536, 1998.

[3] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[4] Stanford CS231N. Tiny ImageNet Visual Recognition Challenge. *https://tiny-imagenet.herokuapp.com/*, 2016.

[5] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR*, pages 248–255, 2009.

[6] John Duchi, Elad Hazan, and Yoram Singer. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[7] Stanislav Fort, Paweł Krzysztof, and Nowak Srini. Stiffness: A New Perspective on Generalization in Neural Networks. *arXiv 1901.09491*, 2019.

[8] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*, pages 249–256, 2010.

[9] Ian J Goodfellow, Oriol Vinyals, and Andrew M Saxe. Qualitatively characterizing neural network optimization problems. In *ICLR*, 2015.

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *CVPR*, pages 770–778, 2016.

[11] Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *IJUFKS*, 6(2):1–10, 1998.

[12] Sepp Hochreiter and Jurgen Schmidhuber. Flat minima. *Neural Computation*, 9(1):1–42, 1997.

[13] Elad Hoffer, Ron Banner, Itay Golan, and Daniel Soudry. Norm matters: efficient and accurate normalization schemes in deep networks. *arXiv:1803.01814*, 2018.

[14] Elad Hoffer, Itay Hubara, and Daniel Soudry. Train longer , generalize better : closing the generalization gap in large batch training of neural networks. In *NIPS*, pages 1729—-1739, 2017.

[15] Stanisław Jastrzębski, Zachary Kenton, Nicolas Ballas, Asja Fischer, Yoshua Bengio, and Amos Storkey. On the relation between the sharpest directions of DNN loss and the SGD step length. In *ICLR*, 2019.

[16] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima. In *ICLR*, 2017.

[17] Diederik P Kingma and Jimmy Lei Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

[18] Alex Krizhevsky and Geoffrey Hinton. Learning Multiple Layers of Features from Tiny Images. Technical report, University of Toronto, 2009.

[19] Tengyuan Liang, Tomaso Poggio, Alexander Rakhlin, and James Stokes. Fisher-Rao Metric, Geometry, and Complexity of Neural Networks. In *AISTATS*, volume 89, 2019.

[20] Ari Morcos, Maithra Raghu, and Samy Bengio. Insights on representational similarity in neural networks with canonical correlation. In *NIPS*, 2018.

[21] Ari S Morcos, David G T Barrett, Neil C Rabinowitz, and Matthew Botvinick. On the importance of single directions for generalization. In *ICLR*, 2018.

[22] Behnam Neyshabur, Srinadh Bhojanapalli, David Mcallester, and Nathan Srebro. Exploring Generalization in Deep Learning. In *NIPS*, 2017.

[23] Behnam Neyshabur, Ryota Tomioka, and Nathan Srebro. In search of the real inductive bias: On the role of implicit regularization in deep learning. In *ICLR*, 2015.

[24] Behnam Neyshabur, Ryota Tomioka, and Nathan Srebro. Norm-Based Capacity Control in Neural Networks. In *Conference on Learning Theory*, pages 1376–1401, 2015.

[25] Roman Novak, Yasaman Bahri, Daniel A Abolafia, Jeffrey Pennington, and Jascha Sohl-dickstein. Sensitivity and generalization in neural networks: an empirical study. In *ICLR*, 2018.

[26] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv:1409.1556*, 2014.

[27] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning, 2012.

[28] Twan van Laarhoven. L2 Regularization versus Batch and Weight Normalization. *arXiv:1706.05350*, 2017.

[29] Ashia C Wilson, Rebecca Roelofs, Mitchell Stern, Nathan Srebro, and Benjamin Recht. The Marginal Value of Adaptive Gradient Methods in Machine Learning. In *NIPS*, pages 4151–4161, 2017.

[30] Chen Xing, Devansh Arpit, Christos Tsirigotis, and Yoshua Bengio. A Walk with SGD. *arXiv:1802.08770*, 2018.

[31] Huan Xu and Shie Mannor. Robustness and generalization. *Machine learning*, 86(3):391–423, 2012.

[32] Sergey Zagoruyko. 92.45% on CIFAR-10 in Torch. *http://torch.ch/blog/2015/07/30/cifar.html*, 2015.

[33] Sergey Zagoruyko and Nikos Komodakis. Wide Residual Networks. In *BMVC*, 2016.

[34] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires re-thinking generalization. In *ICLR*, 2017.

[35] Guodong Zhang, Chaoqi Wang, Bowen Xu, and Roger Grosse. Three mechanisms of weight decay regularization. In *ICLR*, 2019.

A Supplementary Material

A.1 Code

Code to reproduce all the training procedures and figures of this paper is available at `https://github.com/anonymousDLresearch/Layer-rotation` or at `https://github.com/Simoncarbo`.

A.2 Related work

After the intriguing generalization properties of deep neural networks were highlighted by [23, 34, 16], several works have tried to identify aspects of training which could predict generalization performance in a consistent and general way.

A first line of work tries to identify the characteristics of *trained models* that correlate with good generalization properties. Different complexity metrics have been proposed: norm-based metrics are studied in [2, 24, 19], sensitivity-based metrics in [31, 25] and sharpness-based metrics in [12, 16]. These three approaches are compared and further studied in [22]. Other works proposed complexity metrics that explicitly use the decomposition of deep neural networks in layers. Such decomposition is also key to our work and has already been very successful when analysing the training difficulties of deep neural networks [3, 11, 8]. In [20], the similarity of hidden representations resulting from training with different initializations is used as an indicator of generalization. In [21], sensitivity to perturbation of hidden representations is studied.

While the works described above help us understand the characteristics of models that generalize well, they don't explicitly disclose how *the training procedure* itself leads to such characteristics. A second line of work studies indicators of generalization that characterize the whole training trajectory instead of solely focusing on its endpoint. The sharpness metric is revisited in [15] and [7] analyses stiffness. While the presence of noise is believed to affect generalization, the mechanisms at play are poorly understood [9, 30] and a clear metric to quantify its influence on generalization is still lacking. [14] studies the evolution of the euclidean distance between the model's weight vector and its initialization in the context of large batch training. This metric is the most similar to layer rotation, and also has the particularity of being both easy to monitor and to control. Our work differs by the used distance metric (layer-level cosine distance instead of model-level euclidean distance) and by performing a more extensive study that extends the context of large-batch training.

A.3 Pseudocode for the Layca algorithm

Cfr. Algorithm 1.

Algorithm 1 Layca, an algorithm that enables control over the amount of weight rotation per step for each layer through its learning rate parameter (cfr. Section 2.2).

Require: o, an optimizer (SGD is the default choice)
Require: T, the number of training steps
L is the number of layers in the network
for l=0 **to** L-1 **do**
  Require: $\rho_l(t)$, a layer's learning rate schedule
  Require: w_0^l, the initial multiplicative weights of layer l
end for
$t \leftarrow 0$
while $t < T$ **do**
  $s_t^0, ..., s_t^{L-1} = \text{getStep}(o, w_t^0, ..., w_t^{L-1})$ (get the updates of the selected optimizer)
  for l=0 **to** L-1 **do**
    $s_t^l \leftarrow s_t^l - \frac{(s_t^l \cdot w_t^l) w_t^l}{w_t^l \cdot w_t^l}$ (project step on space orthogonal to w_t^l)
    $s_t^l \leftarrow \frac{s_t^l \|w_t^l\|_2}{\|s_t^l\|_2}$ (rotation-based normalization)
    $w_{t+1}^l \leftarrow w_t^l + \rho_l(t) s_t^l$ (perform update)
    $w_{t+1}^l \leftarrow w_{t+1}^l \frac{\|w_0^l\|_2}{\|w_{t+1}^l\|_2}$ (project weights back on sphere)
  end for
  $t \leftarrow t + 1$
end while

A.4 Visualizing the α parameter.

The α parameter is used in Section 3 to characterize the layer prioritization schemes used during training. While the specific parametrization is provided in Equation 1, Figure 2 provides a graphical illustration of it.

A.5 Test accuracies for supplementary α and $\rho(0)$ configurations

Cfr. Figure 3.

A.6 Learning rate decay schemes

Our work uses standard learning rate decay schemes, as follows:

- C10-CNN1: 100 epochs and a reduction of the learning rate by a factor 5 at epochs 80, 90 and 97
- C100-resnet: 100 epochs and a reduction of the learning rate by a factor 10 at epochs 70, 90 and 97
- tiny-CNN: 80 epochs and a reduction of the learning rate by a factor 5 at epoch 70
- C10-CNN2: 250 epochs and a reduction of the learning rate by a factor 5 at epochs 100, 170, 220
- C100-WRN: 250 epochs and a reduction of the learning rate by a factor 5 at epochs 100, 170, 220

The only exceptions are C10-CNN2 and C100-WRN training with SGD+weight decay and with adaptive methods, where the learning rate decay schemes are the ones used in their original implementation or in [29].

A.7 Training errors associated to the layer rotation curves.

In Figures 1a, 1b and 1c, the test accuracies corresponding to each layer rotation curves visualization are provided. While it is briefly mentioned that training accuracy is close to perfect in most cases, Tables 2, 3 and 4 provide the exact values for completeness.

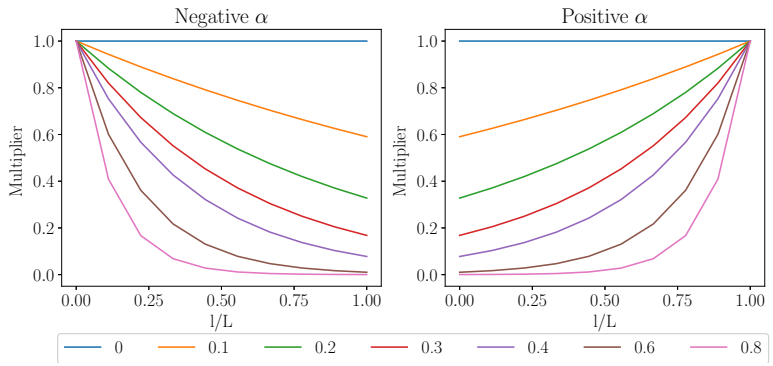


Figure 2: Visualization of the prioritization schemes as parametrized by α (cfr. Section 3). The colours of the lines represent the absolute value of α. Illustration is separated for prioritization of the first layers (negative α values) and of the last layers (positive α values). The layer-wise learning rate multipliers (y-axis) depend on the layer's location in the network (x-axis), which is represented by the layer index l (in forward pass ordering) divided by the number of layers L.

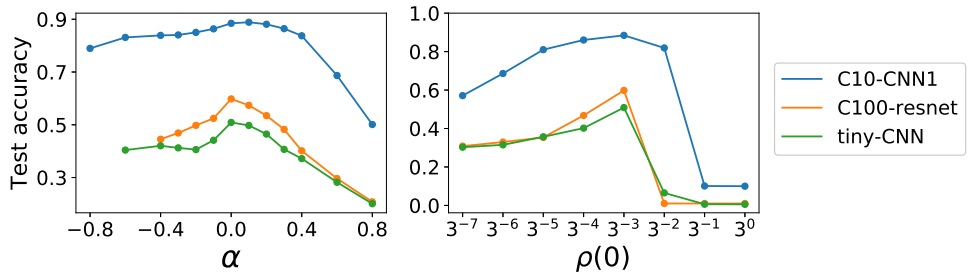


Figure 3: Provides test accuracies for a larger set of α and $\rho(0)$ values than the ones presented in 1a, for the first three tasks of Table 1.

Table 2: Train accuracies associated to Figure 1a

	$\alpha = 0.6$	$\alpha = -0.6$	$\rho(0) = 3^{-5}$	$\rho(0) = 3^{-4}$	Best
C10-CNN1	100%	99.99%	100%	100%	99.99%
C100-resnet	82.09%	99.54%	99.87%	99.99%	99.75%
tiny-CNN	99.98%	99.95%	99.97%	99.97%	98.91%
C10-CNN2	100%	99.94%	99.99%	99.99%	99.97%
C100-WRN	99.88%	99.91%	99.97%	99.99%	99.96%

Table 3: Train accuracies associated to Figure 1b

	C10-CNN1	C100-resnet	tiny-CNN	C10-CNN2	C100-WRN
SGD	100%	100%	100%	100%	100%
SGD + L_2	100%	100%	100%	100%	100%

Table 4: Train accuracies associated to Figure 1c

	C10-CNN1	C100-resnet	tiny-CNN	C10-CNN2	C100-WRN
Adaptive methods	100%	100%	100%	100%	99.9%
Adaptive + Layca	100%	99.7%	99.2%	100%	100%