INITIALIZATION AND REGULARIZATION OF FACTORIZED NEURAL LAYERS

Anonymous authors
Paper under double-blind review

ABSTRACT

Factorized layers—operations parameterized by products of two or more matrices—occur in a variety of deep learning contexts, including compressed model training, certain types of knowledge distillation, and multi-head self-attention architectures. We study how to initialize and regularize deep nets containing such layers, examining two simple, understudied schemes, *spectral initialization* and *Frobenius decay*, for improving their performance. The guiding insight is to design optimization routines for these networks that are as close as possible to that of their well-tuned, non-decomposed counterparts; we back this intuition with an analysis of how the initialization and regularization schemes impact training with gradient descent, drawing on modern attempts to understand the interplay of weight-decay and batch-normalization. Empirically, we highlight the benefits of spectral initialization and Frobenius decay across a variety of settings. In model compression, we show that they enable low-rank methods to significantly outperform both unstructured sparsity and tensor methods on the task of training low-memory residual networks; analogs of the schemes also improve the performance of tensor decomposition techniques. For knowledge distillation, Frobenius decay enables a simple, *self-taught* baseline that yields a compact model from over-parameterized training without requiring retraining with or pruning a teacher network. Finally, we show how both schemes applied to multi-head attention lead to improved performance on both translation and unsupervised pre-training.

1 INTRODUCTION

Most neural network layers consist of matrix-parameterized functions followed by simple operations such as activation or normalization. These layers are the main sources of model expressivity, but also the biggest contributors to computation and memory cost; thus modifying these layers to improve computational performance while maintaining performance is highly desirable. We study the approach of *factorizing* layers, i.e. reparameterizing them so that their weights are defined as products of two or more matrices. When these are smaller than the original matrix, the resulting networks are more efficient for both training and inference (Denil et al., 2013; Moczulski et al., 2015; Ioannou et al., 2016; Tai et al., 2016), resulting in *model compression*. On the other hand, if training cost is not a concern, one can increase the width or depth of the factors to over-parameterize models (Guo et al., 2020; Cao et al., 2020), improving learning without increasing inference-time cost. This can be seen as a simple, *self-taught* (i.e. teacher-free) form of knowledge distillation. Factorized layers also arise implicitly, such as in the case of multi-head attention (MHA) (Vaswani et al., 2017).

Despite such appealing properties, networks with factorized neural layers are non-trivial to train from scratch, requiring custom initialization, regularization, and optimization schemes. In this paper we focus on initialization, regularization, and how they interact with gradient-based optimization of factorized layers. We first study *spectral initialization* (SI), which initializes factors using singular value decomposition (SVD) so that their product approximates the target un-factorized matrix. Then, we study *Frobenius decay* (FD), which regularizes the product of matrices in a factorized layer rather than its individual terms. Both are motivated by matching the training regimen of the analogous un-factorized optimization. Note that SI has been previously considered in the context of model compression, albeit usually for factorizing pre-trained models (Nakkiran et al., 2015; Yaguchi et al., 2019; Yang et al., 2020) rather than low-rank initialization for end-to-end training; FD has been used in model compression using an uncompressed teacher (Idelbayev & Carreira-Perpiñán, 2020).

We formalize and study the justifications of SI and FD from both the classical perspective—matching the un-factorized objective and scaling—and in the presence of BatchNorm (Ioffe & Szegedy, 2015), where this does not apply. Extending recent studies of weight-decay (Zhang et al., 2019), we argue that the effective step-size at spectral initialization is controlled by the factorization's Frobenius norm and show convincing evidence that weight-decay penalizes the *nuclear* norm.

We then turn to applications, starting with low-memory training, which is dominated by unstructured sparsity methods—i.e. guessing "lottery tickets" (Frankle & Carbin, 2019)—with a prevailing trend of viewing low-rank methods as uncompetitive for compression (Blalock et al., 2020; Zhang et al., 2020; Idelbayev & Carreira-Perpiñán, 2020; Su et al., 2020), here we show that, without tuning, factorized neural layers outperform all structured sparsity methods on ResNet architectures (He et al., 2016), despite lagging on VGG (Simonyan & Zisserman, 2015). Through ablations, we show that this result is due to using both SI and FD on the factorized layers. We further compare to a recent evaluation of tensor-decomposition approaches for compressed WideResNet training (Zagoruyko & Komodakis, 2016; Gray et al., 2019), showing that (a) low-rank approaches with SI and FD can outperform them and (b) they are themselves helped by tensor-variants of SI and FD.

We also study a fledgling subfield we term *self-taught* knowledge distillation (Arora et al., 2018; Guo et al., 2020; Cao et al., 2020) in which a model's weights are over-parameterized as over-complete factorizations; the factors are then multiplied after training to obtain the compact representation of the same network. We show that FD leads to significant improvements, demonstrating self-taught training of a ResNet56 model that outperforms a regularly-trained ResNet110.

Finally, we study Transformer architectures, starting by showing that FD improves translation performance when applied to MHA. We also show that SI is critical for low-rank training of the model's linear layers. In an application to BERT pre-training (Devlin et al., 2019), we construct a Frobenius-regularized variant—*FLAMBé*—of the LAMB method (You et al., 2020), and show that, much like for transformers, it improves performance both for full-rank and low-rank MHA layers.

To summarize, our main contributions are (1) motivating the study of training factorized layers via both the usual setting (model compression) and recent applications (knowledge distillation, multi-head attention), (2) justifying the use of SI and FD mathematically and experimentally, and (3) demonstrating their effectiveness by providing strong baselines and novel advances in many settings.

1.1 Related Work

We are not the first study at gradient descent on factorized layers; in particular, deep linear nets are well-studied in theory (Saxe et al., 2014; Gunasekar et al., 2019). Apart from Bernacchia et al. (2018) these largely examine existing algorithms, although Arora et al. (2018) do propose what is effectively self-taught knowledge distillation. Rather than the descent method, we focus on the initialization and regularization of factorized layers. For the former, several papers use SI *after* training (Nakkiran et al., 2015; Yaguchi et al., 2019; Yang et al., 2020), while Ioannou et al. (2016) argue for initializing factors as though they were single layers, which we find useful on regular ResNets but inferior to SI on wider networks. Outside deep learning, spectral methods have also been shown to yield better gradient descent initializations for certain matrix and tensor problems (Chi et al., 2019; Cai et al., 2019). For regularization, Gray et al. (2019) suggest *compression-rate scaling*, which scales weight-decay using the reduction in parameter count; this is justified via the standard Bayesian understanding of ℓ_2-regularization (Murphy, 2012). However, we find that FD is superior to any tuning of regular weight-decay, which subsumes this approach. Our own analysis is based on recent work suggesting that the function of weight-decay in modern deep nets is to aid optimization by preventing the effective step-size from becoming too small (Zhang et al., 2019).

2 Preliminaries on Factorized Neural Layers

In the training phase of (self-)supervised ML, we often solve optimization problems of the form $\min_{\theta\in\Theta} \frac{1}{|\mathcal{S}|}\sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathcal{S}} \ell(f_\theta(\boldsymbol{x}),\boldsymbol{y}) + \Omega(\theta)$, where $f_\theta : \mathcal{X} \mapsto \mathcal{Y}$ is a function from input domain $\mathcal{X}$ to output domain $\mathcal{Y}$ parameterized by elements $\theta \in \Theta$, $\ell : \mathcal{Y}\times\mathcal{Y} \mapsto \mathbb{R}$ is a scalar-valued loss function, $\Omega : \Theta \mapsto \mathbb{R}$ is a scalar-valued regularizer, and $\mathcal{S} \subset \mathcal{X}\times\mathcal{Y}$ is a finite set of (self-)supervised training examples. We study the setting where f_θ is a neural network, an L-layer function whose parameters θ consist of L matrices $\boldsymbol{W}_i \in \mathbb{R}^{m_i\times n_i}$ and whose output $f_\theta(\boldsymbol{x})$ given input $\boldsymbol{x}$ is defined recursively using L functions g_i via the formula $\boldsymbol{x}_i = g_i(\boldsymbol{W}_i, \boldsymbol{x}_{i-1})$, with $\boldsymbol{x}_0 = \boldsymbol{x}$ and $f_\theta(\boldsymbol{x}) = \boldsymbol{x}_L$.

The standard approach to training f_θ is to specify the regularizer Ω, (randomly) pick an initialization in Θ, and iteratively update the parameters using some first-order algorithm such as SGD to optimize the objective above until some stopping criterion is met. However, in many cases we instead optimize over factorized variants of these networks, in which some or all of the matrices $\boldsymbol{W}_i \in \mathbb{R}^{m_i \times n_i}$ are re-parameterized as a product $\boldsymbol{W}_i = \boldsymbol{U}_i(\prod_{j=1}^{d_i} \boldsymbol{M}_{ij})\boldsymbol{V}_i^T$ for some inner depth $d_i \geqslant 0$ and matrices $\boldsymbol{U}_i \in \mathbb{R}^{m_i \times r_i}$, $\boldsymbol{V}_i \in \mathbb{R}^{n_i \times r_i}$, and $\boldsymbol{M}_{ij} \in \mathbb{R}^{r_i \times r_i}\ \forall\, j$. As discussed in the following examples, this can be done to obtain better generalization, improve optimization, or satisfy practical computational or memory constraints during training or inference. For simplicity, we drop the subscript i whenever re-parameterizing only one layer and only consider the cases when inner depth d is 0 or 1.

2.1 Fully-Connected Layers

A fully-connected layer takes an n-dimensional input $\boldsymbol{x}_{i-1}$ and outputs an m-dimensional vector $\boldsymbol{x}_i = \sigma(\boldsymbol{W}\boldsymbol{x}_{i-1})$, where $\sigma : \mathbb{R}^m \mapsto \mathbb{R}^m$ is an element-wise activation function. Here, decomposing $\boldsymbol{W} \in \mathbb{R}^{m \times n}$ into the product $\boldsymbol{U}\boldsymbol{V}^T$, where $\boldsymbol{U} \in \mathbb{R}^{m \times r}$, $\boldsymbol{V} \in \mathbb{R}^{n \times r}$, and setting $r \ll \min\{m, n\}$ reduces computation and memory costs from $\mathcal{O}(mn)$ to $\mathcal{O}(mr + nr)$. Low-rank decomposition also decreases the model complexity, which standard learning theory suggests can improve generalization error of the model; indeed, some recent results bound the generalization error of a neural network by the stable rank (and thus also the rank) of its weight matrices (Neyshabur et al., 2018, Theorem 1). We refer to this setting as *model compression*.

Alternatively, by setting $r \geqslant \min\{m, n\}$ and/or including an inner matrix $\boldsymbol{M} \in \mathbb{R}^{r \times r}$, we can attempt to take advantage of improved optimization due to increased width (Du & Hu, 2019) and/or increased depth (Arora et al., 2018). Crucially, this does not increase inference costs because we can recompose the matrix after training and just use the product. As the goal is to obtain a better small model by first training a large one, we refer to this setting as *self-taught knowledge distillation*; of course, unlike regular distillation it is much simpler since there is no student-teacher training stage.

2.2 Convolutional Layers

A 2d convolutional layer takes an $h \times w \times c_{i-1}$-dimensional input $\boldsymbol{x}_{i-1}$ and outputs a $h \times w \times c_i$-dimensional output $\boldsymbol{x}_i$ defined by convolving c_i different $k \times k$ filters over each of c_{i-1} input channels. Often the result is passed through a nonlinearity. 2d convolutional layers are parameterized by $c_i \times c_{i-1} \times k \times k$ tensors and require $O(k^2 c_i c_{i-1})$ memory and compute. A straightforward way of factorizing this tensor without using tensor decomposition is to reshape it into a $c_i k \times c_{i-1} k$ matrix $\boldsymbol{W}$, which can then be decomposed as $\boldsymbol{W} = \boldsymbol{U}\boldsymbol{V}^T$ for $\boldsymbol{U} \in \mathbb{R}^{c_i k \times r}$, $\boldsymbol{V} \in \mathbb{R}^{c_{i-1} k \times r}$ and some rank $r > 0$. As in the fully-connected case, we can either set the rank r to be small in order to reduce the number of parameters or alternatively increase the width (r) or the depth (d) of the factorization to do self-taught knowledge distillation.

Note that in the low-rank case a naive approach does not save computation since we must first multiply $\boldsymbol{U}$ and $\boldsymbol{V}^T$, reshape the product $\boldsymbol{U}\boldsymbol{V}^T$, and then use the resulting tensor in a regular 2d convolution of the original size and complexity. However, as shown by Tai et al. (2016), applying the 2d $k \times k$ convolution with c_{i-1} input channels and c_i output channels obtained by reshaping $\boldsymbol{U}\boldsymbol{V}^T$ is equivalent to a composition of two 1d convolutions: the first defined by $\boldsymbol{V}^T \in \mathbb{R}^{r \times c_{i-1} k}$ consists of r output channels and filters of size k along one input dimension and the second defined by $\boldsymbol{U} \in \mathbb{R}^{c_i k \times r}$ consists of c_i output channels and filters of size k along the other input dimension. Together the two 1d convolutions require $O(kr(c_i + c_{i-1}))$ memory and computation, which is significantly better than the $O(k^2 c_i c_{i-1})$ cost of the unfactorized case if $r \ll k \min\{c_i, c_{i-1}\}$.

2.3 Multi-Head Attention

An MHA layer (Vaswani et al., 2017) with H attention heads and hidden dimension d can be expressed as being parameterized with $4H$ matrices: one each of $\boldsymbol{Q}_h, \boldsymbol{K}_h, \boldsymbol{V}_h, \boldsymbol{O}_h \in \mathbb{R}^{d \times d/H}$ for each head h. Then for a length-T input $\boldsymbol{x} \in \mathbb{R}^{T \times d}$ it outputs

$$\sum_{h=1}^{H} \operatorname{Softmax}\left(\frac{\boldsymbol{x}\boldsymbol{Q}_h\boldsymbol{K}_h^T\boldsymbol{x}^T}{\sqrt{d/H}}\right)\boldsymbol{x}\boldsymbol{V}_h\boldsymbol{O}_h^T \tag{1}$$

MHA combines $2H$ quadratic forms $\boldsymbol{Q}_h\boldsymbol{K}_h^T$, $\boldsymbol{V}_h\boldsymbol{O}_h^T$ of rank $r = d/H$, each a product of matrices, i.e. a factorized layer. We refer to the first form as "Query-Key" and the second as "Output-Value." Note that r can be varied independently of d to change expressivity, memory, and computation.

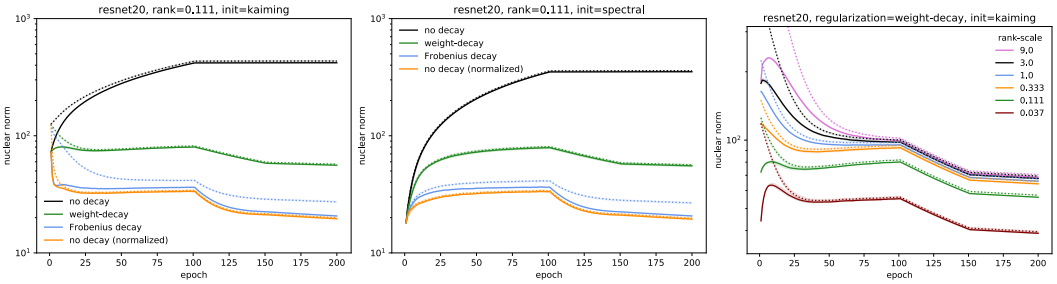

Figure 1: Average nuclear norm across factorized layers of ResNet20 during CIFAR-10 training when initialized regularly (left) and using SI (center); the dotted line is the upper bound on the nuclear norm regularized by weight-decay (2). The right plot track the same quantities for the case of regular weight-decay across different rank-scales. "no decay (normalized)" normalizes matrix factors after each step to have the same norm as "Frobenius decay " (detailed in Section 3.3).

3 Initialization and Regularization

We now define the initialization and regularization schemes we study as natural extensions of techniques for non-factorized models. They thus require no tuning when an existing training implementation of a non-factorized deep net is available. We later discuss how to justify these schemes when layers are normalized, e.g. using BatchNorm (Ioffe & Szegedy, 2015). In all experiments with convolutional models in this and subsequent sections we factorize all layers except the first and last, which are small, and determine layer ranks by multiplying a uniform scaling factor by the product of a layer's output channels and kernel width. This *rank-scale* can be varied to attain the desired number of parameters. Note that our approach may be further improved via a more sophisticated or adaptive rank-assignment scheme (Idelbayev & Carreira-Perpiñán, 2020).

3.1 Spectral Initialization

Initialization is a major focus of deep learning research (He et al., 2015; Mishkin & Matas, 2016; Yang & Schoenholz, 2017). A common approach is to prevent compounding changes in the norms of the intermediate representations across layers caused by repeated multiplication by the weight matrices. The *spectral initialization* scheme for initializing *low-rank* factorized layers attempts to inherit, when possible, this property from an existing initialization by using SVD to ensure that the resulting product matrix is as close as possible to the original parameter:

Definition 3.1. *Let $\boldsymbol{W} \in \mathbb{R}^{m\times n}$ be a parameter of an unfactorized layer. For $r \leqslant \min\{m, n\}$ the* **spectral initialization** *(SI) of the factors $\boldsymbol{U} \in \mathbb{R}^{m\times r}, \boldsymbol{V} \in \mathbb{R}^{n\times r}$ of the corresponding factorized layer sets $\boldsymbol{U} = \tilde{\boldsymbol{U}}\sqrt{\Sigma}$ and $\boldsymbol{V} = \tilde{\boldsymbol{V}}\sqrt{\Sigma}$, for $\tilde{\boldsymbol{U}}, \Sigma, \tilde{\boldsymbol{V}} = \mathrm{SVD}_r(\boldsymbol{W})$ given by the rank-r SVD of $\boldsymbol{W}$.*

SI preserves the largest singular value of $\boldsymbol{W}$, so if the original scheme did not suffer from a compounding increase in representation norm then neither will spectral initialization. On the other hand, while low-rank layers impose a nullspace, SI aligns it with the directions minimized by $\boldsymbol{W}$.

3.2 Frobenius Decay

Weight-decay is a common regularizer for deep nets, often implemented explicitly by adding $\Omega(\theta) = \frac{\lambda}{2}\sum_{\boldsymbol{W}\in\theta}\|\boldsymbol{W}\|_F^2$ to the objective for some $\lambda \geqslant 0$. Classically, it is thought to improve generalization by constraining model capacity. When training factorized layers parameterized by $\boldsymbol{U}(\prod_{j=1}^{d}\boldsymbol{M}_j)\boldsymbol{V}^T$, the easiest approach to implement is to replace each $\frac{\lambda}{2}\|\boldsymbol{W}\|_F^2$ term in $\Omega(\theta)$ by $\frac{\lambda}{2}\left(\|\boldsymbol{U}\|_F^2 + \|\boldsymbol{V}\|_F^2 + \sum_{j=1}^{d}\|\boldsymbol{M}_j\|_F^2\right)$. However, this yields a very different optimization problem: for example, if we consider the case of $d = 0$ then this regularizer is in fact an upper bound on the nuclear norm of the recomposed matrix $\boldsymbol{U}\boldsymbol{V}^T$ (Srebro & Shraibman, 2005, Lemma 1):

$$\frac{\lambda}{2}\left(\|\boldsymbol{U}\|_F^2 + \|\boldsymbol{V}\|_F^2\right) \geqslant \min_{\tilde{\boldsymbol{U}}\tilde{\boldsymbol{V}}^T = \boldsymbol{U}\boldsymbol{V}^T} \frac{\lambda}{2}\left(\|\tilde{\boldsymbol{U}}\|_F^2 + \|\tilde{\boldsymbol{V}}\|_F^2\right) = \lambda\|\boldsymbol{U}\boldsymbol{V}^T\|_* \tag{2}$$

In fact, Figure 1 shows that for weight-decay this upper bound is tight throughout the training of factorized ResNet20 across a variety of ranks and initializations, suggesting that the naive approach is indeed regularizing the nuclear norm rather than the Frobenius norm.

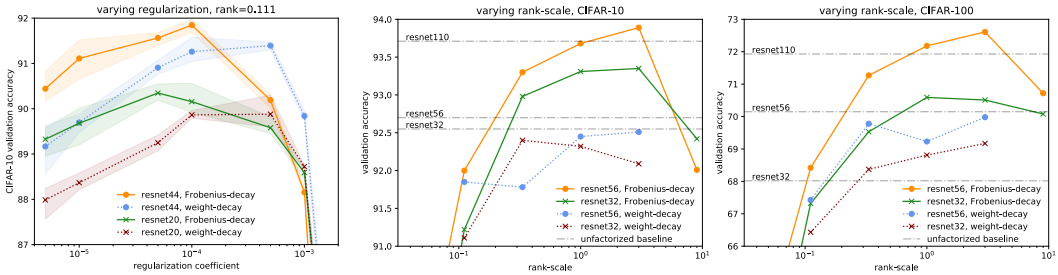

Figure 2: Comparison of weight-decay and FD at different regularization levels (left) and different rank-scales (center and right) when training factorized ResNets on CIFAR.

Since compression already constrains capacity, in the low-rank case one might favor just reducing regularization, e.g. multiplying λ by the compression rate (Gray et al., 2019). However, Figure 2 shows that this can lead to *worse* performance, and the approach still penalizes the nuclear norm. *Frobenius decay* avoids this issue by simply penalizing the squared norm of the entire factorization:

Definition 3.2. *For $\lambda \geqslant 0$ let $\frac{\lambda}{2}\|\boldsymbol{W}\|_F^2$ be the contribution of an unfactorized layer parameterized by $\boldsymbol{W} \in \mathbb{R}^{m\times n}$ to the penalty term. Then the* **Frobenius decay** *(FD) penalty on matrices $\boldsymbol{U} \in \mathbb{R}^{m\times r}$, $\boldsymbol{V} \in \mathbb{R}^{n\times r}$, and $\boldsymbol{M}_j \in \mathbb{R}^{r\times r}$ of the corresponding factorized layer is $\frac{\lambda}{2}\left\|\boldsymbol{U}(\prod_{j=1}^{d}\boldsymbol{M}_j)\boldsymbol{V}^T\right\|_F^2$.*

By substituting the factorization into the original regularized empirical risk minimization, FD makes the minimal possible change to the objective. In particular, any rank-r optimum of the non-factorized problem will also minimize the factorized one. Notably, the squared Frobenius norm is a *stronger* penalty than the nuclear norm implicitly regularized by weight-decay, yet it still yields better models.

3.3 Initialization and Regularization in the Presence of Normalization

The use of spectral initialization and Frobenius decay is largely motivated by the norms of the recomposed matrices: SI prevents them increasing feature vector norms across layers, while FD constrains model capacity via parameter norms. However, normalization layers like BatchNorm (Ioffe & Szegedy, 2015) and others (Ba et al., 2016) largely negate the forward-pass and model-capacity effects of the norms of the weights parameterizing the layers they follow. Thus for most modern models we need a different explanation for the effectiveness of SI and FD.

Despite the fact that most layers' weight-norms do not affect inference or capacity, regular weight-decay remains useful for optimizing deep nets. Recently, Zhang et al. (2019) extended the analysis of Hoffer et al. (2018) to argue that the *effective learning rate* of the weight *direction* $\hat{\boldsymbol{W}}$ is roughly $\eta/\|\boldsymbol{W}\|_F^2$, where η is the SGD step-size. Thus by preventing the norm of $\boldsymbol{W}$ from growing too large weight-decay maintains a large-enough effective learning rate during training. We draw on this analysis to explore the effect of SI and FD on factorized models. For simplicity, we define a normalized layer to be one that does not depend on the scale of its parameter. While often not precisely true due to stability offset terms, it otherwise roughly holds for normalized linear layers, convolutional layers in ResNets, and the Output-Value quadratic form in Transformers if the residual connection is added after rather than before normalization.

Definition 3.3. *A* normalized layer *$g(\boldsymbol{W}, \boldsymbol{x})$ parameterized by $\boldsymbol{W} \in \mathbb{R}^{m\times n}$ is one that satisfies $g(\boldsymbol{W}, \boldsymbol{x}) = g(\rho\boldsymbol{W}, \boldsymbol{x})$ for all $\boldsymbol{W} \in \mathbb{R}^{m\times n}$ and all positive scalars ρ.*

Because the output does not depend on the magnitude of $\boldsymbol{U}\boldsymbol{V}^T$, what matters is the direction of the composed matrix. During an SGD step this direction is updated as follows (proof in Appendix A):

Claim 3.1. *Let g be a normalized layer of a differentiable model $f_\theta : \mathcal{X} \mapsto \mathcal{Y}$ parameterized by the product $\boldsymbol{U}\boldsymbol{V}^T$ of matrices $\boldsymbol{U} \in \mathbb{R}^{m\times r}, \boldsymbol{V}^T \in \mathbb{R}^{n\times r}$, $r \geqslant 1$. Suppose at step t its parameters are updated by SGD with sufficiently small step-size $\eta > 0$ via the rules $\boldsymbol{U}_{t+1} \leftarrow \boldsymbol{U}_t - \eta\nabla_{\boldsymbol{U}_t}$ and $\boldsymbol{V}_{t+1} \leftarrow \boldsymbol{V}_t - \eta\nabla_{\boldsymbol{V}_t}$, where for parameter $\boldsymbol{P}$ we let $\nabla_{\boldsymbol{P}} = \frac{1}{|\mathcal{B}|}\sum_{(x,y)\in\mathcal{B}}\nabla_{\boldsymbol{P}}\ell(f_{\theta_t}(x), y)$ be the gradient at θ_t w.r.t. $\boldsymbol{P}$ using batch $\mathcal{B} \subset (\mathcal{X}, \mathcal{Y})$. Then defining $\hat{\nabla}_t = \nabla_{\hat{\boldsymbol{W}}_t}\boldsymbol{V}_t\boldsymbol{V}_t^T + \boldsymbol{U}_t\boldsymbol{U}_t^T\nabla_{\hat{\boldsymbol{W}}_t}$ we have that the vectorized direction $\hat{\boldsymbol{w}}_t = \mathrm{vec}(\hat{\boldsymbol{W}}_t)$ of $\boldsymbol{W}_t = \boldsymbol{U}_t\boldsymbol{V}_t^T$ has an update satisfying*

$$\hat{\boldsymbol{w}}_{t+1} \leftarrow \hat{\boldsymbol{w}}_t - \frac{\eta}{\|\boldsymbol{W}_t\|_F^2}\left(\boldsymbol{I}_{mn} - \hat{\boldsymbol{w}}_t\hat{\boldsymbol{w}}_t^T\right)\mathrm{vec}(\hat{\nabla}_t) + \mathcal{O}(\eta^2) \tag{3}$$

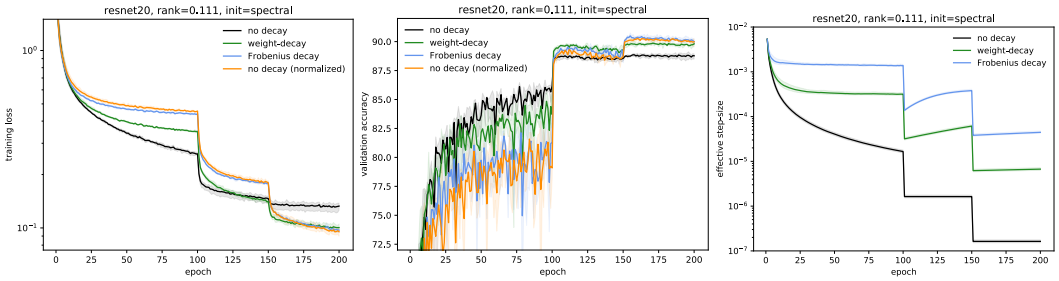

Figure 3: Traces depicting training of low-rank ResNet-20 with different decay settings; "no decay (normalized)" normalizes matrix factors after each step to have the same norm as "Frobenius decay." The effective step size (right) is the average over convolution layers i of $\eta/\|\boldsymbol{U}_i\boldsymbol{V}_i^T\|_F^2$.

Note that (1) we ignore decay because $\lambda = \mathcal{O}(\eta)$ so any resulting term is $\mathcal{O}(\eta^2)$ and (2) the update rule is almost the same as that obtained for the unfactorized case by Zhang et al. (2019), except they have $\hat{\nabla}_t$ as the true gradient of the direction. Thus, apart from a rank-one correction, $\hat{\boldsymbol{W}}_t$ is approximately updated with step-size $\eta/\|\boldsymbol{W}_t\|_F^2$ multiplying a linear transformation of its gradient. To understand the nature of this transformation, note that at spectral initialization we have that $\boldsymbol{V}_0\boldsymbol{V}_0^T = \boldsymbol{U}_0\boldsymbol{U}_0^T = \Sigma_r$ are diagonal matrices of singular values of the full-rank initialization $\boldsymbol{W}$; furthermore, if $\boldsymbol{W}$ is a Gaussian ensemble with scale $1/\sqrt{n}$, which roughly aligns with common initialization schemes (He et al., 2015), then its singular values are roughly distributed around 1 and supported on $[0, 2]$ (Bai & Yin, 1993). Since $\hat{\nabla}_t = \nabla_{\hat{\boldsymbol{W}}_t}\boldsymbol{V}_t\boldsymbol{V}_t^T + \boldsymbol{U}_t\boldsymbol{U}_t^T\nabla_{\hat{\boldsymbol{W}}_t}$, this suggests that, at spectral initialization, an effective learning rate of $\eta/\|\boldsymbol{W}_0\|_F^2$ is a reasonable approximation for the factorized update. This points to the role of SI being to initialize the factorization at an appropriate scale and perhaps also to make the first update more aligned with the gradient w.r.t. $\hat{\boldsymbol{W}}_0$.

As in the unfactorized case, our analysis suggests that, the main role of decay may be to maintain a large effective learning rate $\eta/\|\boldsymbol{W}\|_F^2$; furthermore, FD may be more effective than regular decay because it provides stronger regularization and directly penalizes the quantity of interest. We support this hypothesis using experiments analogous to Zhang et al. (2019) by comparing training a low-rank ResNet-20 with FD to training it with no decay but at each step normalizing all BatchNormed layers to have the same Frobenius norm as the FD run. In Figure 3 we see that the latter scheme closely tracks the former in terms of both the training loss and the test accuracy. Figure 3 also shows that FD maintains a higher effective step-size than regular weight-decay throughout training.

4 Compressed Model Training: Low-Rank, Sparse, and Tensorial

We first study SI and FD for training factorized models with low-rank convolutional layers, comparing against the dominant approaches to low-memory training: sparse layers and tensor decomposition. For direct comparison we evaluate models with near-identical parameter counts; note that by normalizing by memory we *disadvantage* the low-rank approach, which often has a comparative advantage for speed. All models are trained for 200 epochs with the same optimizer settings as for the unfactorized models; the weight-decay coefficient is left unchanged when replacing by FD.

Low-Rank and Sparse Training: We train modified ResNet32 and VGG19 used in the lottery-ticket-guessing literature (Wang et al., 2020; Su et al., 2020). Motivated by Frankle & Carbin (2019), such methods fix a sparsity pattern at initialization and train only the unpruned weights. While they achieve high parameter savings, their computational advantages over full models are less clear, as software and accelerators often do not efficiently implement arbitrary sparsity (Paszke et al., 2019; NVIDIA, 2020). In contrast, we do see acceleration via low-rank convolutions, almost halving the time of ResNet's forward and backward pass at the highest compression. For completeness, we also show methods that vary the sparsity pattern (dynamic), prune trained models (pruning), and prune trained models and retrain (lottery); note that the latter two require training an uncompressed model.

In Table 1 we see that the low-rank approach, with SI & FD, dominates at the higher memory settings of ResNet across all three datasets considered, often outperforming even approaches that train an uncompressed model first. It is also close to the best compressed training approach in the lowest memory setting for CIFAR-100 (Krizhevksy, 2009) and Tiny-ImageNet (Deng et al., 2009).

Table 1: Comparison of low-rank and sparse training in a common evaluation setting for "ticket-guessing" (Wang et al., 2020; Su et al., 2020). Best results overall are *italicized*; best results from full low-memory training are **bolded**. For complete results and deviations see Tables 4 and 5.

	VGG19^{NFC} (full model)*	**CIFAR-10** (93.75±0.48)			**CIFAR-100** (73.42±0.17)			**Tiny-ImageNet** (62.19±0.74)		
Regime	Compressed (from ≈20M)	0.1	0.05	0.02	0.1	0.05	0.02	0.1	0.05	0.02
Regular	Pruning*	93.83	93.69	93.49	73.83	73.07	*71.69*	61.21	60.49	55.55
Training	Lottery† (w. hybrid/rewind)	*94.14*	*93.99*	*93.52*	*74.06*	*73.10*	71.61	61.19	*60.57*	56.18
	Dynamic Sparsity‡	93.75	**93.86**	**93.13**	72.36	**71.98**	**70.70**	***62.49***	**59.81**	***58.36***
Low	Fixed Sparsity$^\#$	**93.77**	93.43	92.45	**72.84**	71.83	68.98	61.02	59.50	57.28
Memory	Low-Rank*	90.71	88.08	80.81	62.32	54.97	42.80	48.50	46.98	36.57
Training	Low-Rank (SI)*	90.99	88.61	82.53	63.77	58.51	45.02	49.87	46.73	37.25
	Low-Rank (FD)*	91.57	88.16	82.81	64.07	59.34	45.73	53.93	47.95	35.63
	Low-Rank (SI & FD)*	91.58	88.98	83.27	65.61	60.1	46.43	53.53	47.60	35.68
	ResNet32^{x2} (full model)*	**CIFAR-10** (94.49±0.29)			**CIFAR-100** (75.41±1.26)			**Tiny-ImageNet** (63.02±0.94)		
Regime	Compressed (from ≈7.5M)	0.1	0.05	0.02	0.1	0.05	0.02	0.15	0.1	0.05
Regular	Pruning*	94.21	*93.29*	90.31	72.34	68.73	60.65	58.86	57.62	51.70
Training	Lottery† (w. hybrid/rewind)	94.14	*93.02*	*90.85*	72.41	69.28	*63.44*	60.31	57.77	51.21
	Dynamic Sparsity‡	92.97	91.61	88.46	69.66	68.20	**62.25**	57.08	57.19	***56.18***
Low	Fixed Sparsity$^\#$	92.97	91.60	**89.10**	69.70	66.82	60.11	57.25	55.53	51.41
Memory	Low-Rank*	93.59	92.45	87.95	72.71	67.86	61.08	60.82	58.72	55.39
Training	Low-Rank (SI)*	92.52	91.72	87.36	71.71	67.41	60.79	59.53	57.60	55.00
	Low-Rank (FD)*	92.92	89.44	83.05	71.85	66.89	55.31	59.45	57.24	54.15
	Low-Rank (SI & FD)*	***94.34***	**92.90**	87.97	***74.41***	***70.22***	61.40	***62.24***	***60.25***	55.97

* Best of (LeCun et al., 1990; Zeng & Urtasun, 2019), obtained from Wang et al. (2020).
† Best of (Frankle & Carbin, 2019; Renda et al., 2020; Su et al., 2020), obtained from Wang et al. (2020) and Su et al. (2020).
‡ Best of (Mostafa & Wang, 2019; Mocanu et al., 2018; Bellec et al., 2018), obtained from Wang et al. (2020).
$^\#$ Best of (Lee et al., 2018; Wang et al., 2020; Su et al., 2020), obtained from Wang et al. (2020) and Su et al. (2020).
* Our method or our reproduction; results averaged over three random trials.

Table 2: Comparison of low-rank and tensor-decomposition training of WideResNet28-10 on CIFAR-10 (mean of 3 trials) with different regularization and initialization. Best errors **bolded**.

compression	decomposition	WD*	WD* & SI	CRS†	CRS† & SI	FD	FD & SI
none (≈36.5M param.)		3.21±0.22					
0.01667	Low-Rank	7.62±0.08	7.72±0.11	8.63±0.87	8.83±0.17	7.23±0.32	7.26±0.45
(≈0.61M param.)	Tensor-Train	8.55±4.22	7.72±0.46	7.01±0.35	6.52±0.17	7.26±0.73	**6.22±0.50**
0.06667	Low-Rank	5.31±0.06	5.49±0.42	5.86±0.22	5.53±0.72	3.89±0.17	**3.86±0.22**
(≈2.4M param.)	Tensor-Train	7.33±0.27	6.49±0.30	5.20±0.37	4.79±0.15	5.02±0.15	5.10±0.35

* Regular weight-decay with $\lambda = 0.005$.
† Regular weight-decay with coefficient scaled by the compression rate (Gray et al., 2019).

On the other hand, the low-rank approach is substantially worse for VGG; nevertheless, ResNet is both smaller and more accurate, so if the goal is an accurate compressed model learned from scratch then one should prefer the low-rank approach. Our results demonstrate a strong, simple baseline not frequently compared to in the low-memory training (Frankle & Carbin, 2019; Wang et al., 2020; Su et al., 2020). In fact, since it preserves the top singular components of the original weights, SI can itself be considered a type of (spectral) magnitude pruning. Finally, Table 1 highlights the complementary nature of SI and FD, which outperform regular low-rank training on both models, although interestingly they consistently *decrease* ResNet performance when used separately.

Matrix and Tensor Decomposition: We next compare against tensor decompositions, another common approach to small model training. A recent evaluation of tensors and other related approaches by Gray et al. (2019) found that Tensor-Train decomposition (Oseledets, 2011) obtained the best memory-accuracy trade-off on WideResNet; we thus compare directly to this approach. Note that, while we normalize according to memory, Tensor-Train must be expanded to the full tensor prior to convolution and thus increases the required compute, unlike low-rank factorization.

In Table 2 we show that at 6.7% of the original parameters the low-rank approach with FD and SI significantly outperforms Tensor-Train. Tensor-Train excels at the highly compressed 1.7% setting but is greatly improved by leveraging tensor analogs of SI—decomposing a random initialization rather than randomly initializing tensor cores directly—and FD—penalizing the squared Frobenius norm of the full tensor. We also compare to CRS (Gray et al., 2019), which scales regular weight-decay by the compression rate. It is roughly as beneficial as FD across different evaluations of Tensor-Train, but FD is significantly better for low-rank factorized neural layers.

Table 3: Self-taught ResNet performance (median of 3 trials). Best at each depth is **bolded**; cases where we match the next deeper network with around the same *training* memory are underlined.

		CIFAR-10						CIFAR-100			
	factorization setting	base	full		deep		wide	base	full	deep	wide
depth	decay ($\lambda = 0.0001$)	WD	WD	FD	WD	FD	FD	WD	FD	FD	FD
110	test accuracy (%)	93.71						71.93			
	training/test param. (M)	1.73						1.73			
56	test accuracy (%)	92.7	92.45	93.68	91.65	93.58	**93.89**	70.15	72.18	72.09	**72.61**
	training param. (M)	0.85	1.72	1.72	2.59	2.59	5.16	0.86	1.73	2.60	5.17
	test param. (M)	0.85	0.85	0.85	0.85	0.85	0.85	0.86	0.86	0.86	0.86
32	test accuracy (%)	92.55	92.32	93.31	91.11	92.99	**93.35**	68.02	**70.59**	70.29	70.51
	training param. (M)	0.46	0.95	0.95	1.43	1.43	2.84	0.47	0.95	1.44	2.84
	test param. (M)	0.46	0.46	0.46	0.46	0.46	0.46	0.47	0.47	0.47	0.47

5 Self-Taught Knowledge Distillation

In contrast to compressed model training, in *knowledge distillation* we do have the capacity to train a large model, but still want to deploy a small one. We consider what we call *self-taught* knowledge distillation, in which a network is over-parameterized in such a way that an equivalent small model can be directly recovered. Previously, this has been studied for small models (Arora et al., 2018; Guo et al., 2020) or using convolution-specific methods (Cao et al., 2020). We take a straightforward factorization approach in which we decompose weights $\boldsymbol{W} \in \mathbb{R}^{m \times n}$ to be products of 2 or 3 matrices while *increasing* the parameter count, either via depth or width. We study three cases: the *full* (rank) setting where $\boldsymbol{W} = \boldsymbol{U}\boldsymbol{V}^T$ for $\boldsymbol{U} \in \mathbb{R}^{m \times m}, \boldsymbol{V} \in \mathbb{R}^{n \times m}$, the *deep* setting where $\boldsymbol{W} = \boldsymbol{U}\boldsymbol{M}\boldsymbol{V}^T$ for $\boldsymbol{U}, \boldsymbol{M} \in \mathbb{R}^{m \times m}$, $\boldsymbol{V} \in \mathbb{R}^{n \times m}$, and the *wide* setting where $\boldsymbol{W} = \boldsymbol{U}\boldsymbol{V}^T$ for $\boldsymbol{U} \in \mathbb{R}^{m \times 3m}$, $\boldsymbol{V} \in \mathbb{R}^{n \times 3m}$. As before, we factorize all but the first and last layer of ResNet and train the self-taught networks using the same routine as for the base model, except when replacing weight-decay by FD. We do not study SI here, simply using the default initialization for $\boldsymbol{U}, \boldsymbol{V}^T$ and $\boldsymbol{M} \sim \mathcal{N}(\boldsymbol{I}_m, 0.01^2)$. Note that the approach does seem sensitive to initialization, so we report the median of three trials.

Table 3 shows the strength of this approach: on both CIFAR datasets we are able to train a ResNet32 that beats ResNet56 and a ResNet56 that beats ResNet110, with both cases needing only half the parameters at inference. Often we can even do so while using about as many parameters at training time as the larger model. The experiments also show that FD is critical: without it, self-taught distillation performs worse than the baseline model, and especially poorly in the "deep" setting. A detailed visualization of this result, comparing the CIFAR performance of factorized ResNets across a range of rank-scale settings covering both the current high-rank (distillation) case and the previous low-rank (compression) case, can be found in Figure 2.

6 Multi-Head Attention as Factorized Quadratic Forms

Our final setting is multi-head attention (Transformer) architectures (Vaswani et al., 2017). As discussed in Section 2, the MHA component is already a factorized layer, consisting of an aggregation over the output of two quadratic forms per head: the "Query-Key" (QK) forms passed into the softmax and the "Output-Value" (OV) forms that multiply the resulting attention scores (c.f. Equation 1). Transformers also contain large linear layers, which can also be factorized for efficiency.

Improving Transformer Training and Compression: We start with the original Transformer architecture on the IWSLT-14 translation task (Cettolo et al., 2014) but use an SGD-based training routine as a baseline (Gehring et al., 2017). As there is no weight-decay by default, we first tune both this and FD on the non-factorized model; here FD is applied to the *implicitly* factorized MHA layers. In Figure 4 we show that this alone yields an improvement: whereas the effect of weight-decay is either negligible or negative, tuning FD does improve the BLEU score. Furthermore, we see that tuning just the OV form in MHA is more robust at higher regularization levels than tuning both OV and QK. We conclude by examining both SI and FD when reducing the number of parameters by (1) factorizing all linear and embedding layers and (2) scaling down the embedding dimension in MHA. In Figure 4 we see that the benefit of Frobenius decay disappears when compressing; on the other hand, SI provides a strong boost under both types of decay, and is in fact necessary for FD to work at all. Note that the major effect here is for the factorized linear layers—we found that SI has minimal effect when applied to MHA, likely because those initializations have already been tuned.

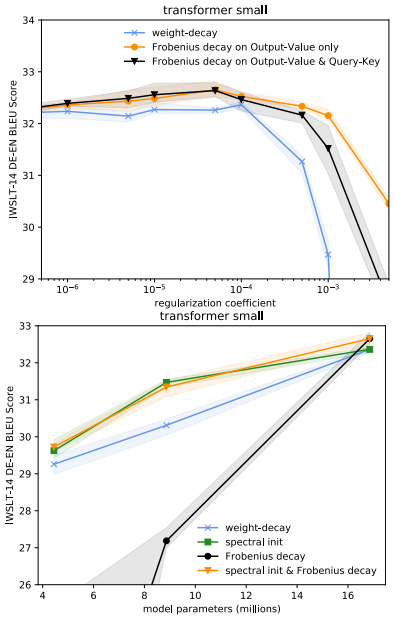

Figure 4: Transformer performance on IWSLT-14 as a function of regularization (top) and compression (bottom).

Figure 5: Comparison of BERT unsupervised pretraining using LAMB (You et al., 2020) and our Frobenius decay scheme FLAMBé. Evaluation is conducted by fine-tuning the final model on the SQuAD question-answering task (Rajpurkar et al., 2016). Guided by IWSLT results, FLAMBé is applied only to the Output-Value form in MHA; regular weight-decay is used on all other parameters. Spectral initialization of MHA is used for FLAMBé as well, but we find the effect minimal. Regularization coefficients for both methods were obtained by tuning on the uncompressed model, targeting the unsupervised loss.

model	optimizer	compression rate (MHA-only)	param. count (M)	SQuAD F1	SQuAD EM
BERT base	LAMB	1.0	110	88.55	81.06
	LAMB (SI & FD)	1.0	110	88.16	80.86
	FLAMBé (SI & FD)	1.0	110	88.88	81.54
	LAMB	0.5	95.7	87.35	79.83
	FLAMBé (SI & FD)	0.5	95.7	87.69	80.42
BERT large	LAMB	1.0	340	91.41	84.99
	FLAMBé (SI & FD)	0.5	295.8	90.55	83.82

FLAMBé for Unsupervised BERT Pre-Training: Lastly, we examine BERT (Devlin et al., 2019), a large transformer trained on a massive unsupervised text corpus and evaluated on downstream language tasks. The state-of-the-art training approach is via the LAMB optimizer (You et al., 2020) using weight-decay based on the AdamW algorithm of Loshchilov & Hutter (2019), in which $\lambda\eta$ times each parameter is subtracted from itself; this equivalent to ℓ_2-regularization for SGD but not for adaptive methods. We can define a similar Frobenius alternative by subtracting $\lambda\eta$ times the Frobenius gradients $\boldsymbol{U}\boldsymbol{V}^T\boldsymbol{V} = \nabla_{\boldsymbol{U}}\frac{1}{2}\|\boldsymbol{U}\boldsymbol{V}^T\|_F^2$ and $\boldsymbol{V}\boldsymbol{U}^T U = \nabla_{\boldsymbol{V}}\frac{1}{2}\|\boldsymbol{U}\boldsymbol{V}^T\|_F^2$ from $\boldsymbol{U}$ and $\boldsymbol{V}$, respectively; when used with the LAMB optimizer we call this method FLAMBé. We see in Table 5 that FLAMBé outperforms the simple FD modification of LAMB and, as with IWSLT, leads to an improvement in downstream task performance without changing model size. For BERT, however, applying FD via FLAMBé also leads to better downstream performance when scaling down the MHA embedding dimension by half. Besides achieving a better compressed model, the success of FLAMBé also shows the potential of new types of decay schemes for adaptive methods.

7 Conclusion

In this paper we studied the design of training algorithms for deep nets containing factorized layers, demonstrating that two simple specializations of standard initialization and regularization schemes from the unfactorized case lead to strong improvements for model compression, knowledge distillation, and MHA-based architectures. While we largely focused on the case where the unfactorized model uses Gaussian initialization and ℓ_2-regularization, we believe our work provides guidance for the many cases where other schemes are used to enforce alternative priors or to improve optimization. For example, SI as defined can be applied to any other random initialization, while our FD results suggest that regularizers such as penalty terms and DropConnect (Wan et al., 2013) should be applied on the product matrix rather than directly on the individual factors. The success of SI and FD for both low-rank and tensor decomposition also suggests that these schemes may be useful when using other types of factorized neural layers such as ACDC (Moczulski et al., 2015) and K-matrices (Dao et al., 2020), both defined as products of structured matrices.

REFERENCES

Sanjeev Arora, Nadav Cohen, and Elad Hazan. On the optimization of deep networks: Implicit acceleration by overparameterization. In *Proceedings of the 35th International Conference on Machine Learning*, 2018.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. arXiv, 2016.

Z. D. Bai and Y. Q. Yin. Limit of the smallest eigenvalue of a large dimensional sample covariance matrix. *The Annals of Probability*, 21(3):1275–1294, 1993.

Guillaume Bellec, David Kappel, Wolfgang Maass, and Robert Legenstein. Deep rewiring: Training very sparse deep networks. In *Proceedings of the 6th International Conference on Learning Representations*, 2018.

Alberto Bernacchia, Máté Lengyel, and Guillaume Hennequin. Exact natural gradient in deep linear networks and application to the nonlinear case. In *Advances in Neural Information Processing Systems*, 2018.

Davis Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle, and John Guttag. What is the state of neural network pruning? In *Proceedings of Machine Learning and Systems*, 2020.

Changxiao Cai, Gen Li, H. Vincent Poor, and Yuxin Chen. Nonconvex low-rank tensor completion from noisy data. In *Advances in Neural Information Processing Systems*, 2019.

Jinming Cao, Yangyan Li, Mingchao Sun, Ying Chen, Dani Lischinski, Daniel Cohen-Or, Baoquan Chen, and Changhe Tu. DO-Conv: Depthwise over-parameterized convolutional layer. arXiv, 2020.

Mauro Cettolo, Jan Niehues, Sebastian Stüker, Luisa Bentivogli, and Marcello Federico. Report on the 11th IWSLT evaluation campaign. In *Proceedings of the International Workshop on Spoken Language Translation*, 2014.

Yuejie Chi, Yue M. Lu, and Yuxin Chen. Nonconvex optimization meets low-rank matrix factorization. *IEEE Transactions on Signal Processing*, 67:5239–5269, 2019.

Tri Dao, Nimit Sohoni, Albert Gu, Matthew Eichhorn, Amit Blonder, Megan Leszczynski, Atri Rudra, and Christopher Ré. Kaleidoscope: An efficient, learnable representation for all structured linear maps. In *Proceedings of the 8th International Conference on Learning Representations*, 2020.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2009.

Misha Denil, Babak Shakibi, Laurent Dinh, Marc'Aurelio Ranzato, and Nando de Freitas. Predicting parameters in deep learning. In *Advances in Neural Information Processing Systems*, 2013.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2019.

Simon S. Du and Wei Hu. Width provably matters in optimization for deep linear neural networks. In *Proceedings of the 36th International Conference on Machine Learning*, 2019.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural. In *Proceedings of the 7th International Conference on Learning Representations*, 2019.

Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin. Convolutional sequence to sequence learning. In *Proceedings of the 35th International Conference on Machine Learning*, 2017.

Gavin Gray, Elliot J. Crowley, and Amos Storkey. Separable layers enable structured efficient linear substitutions. arXiv, 2019.

Suriya Gunasekar, Jason Lee, Daniel Soudry, and Nathan Srebro. Implicit bias of gradient descent on linear convolutional networks. In *Advances in Neural Information Processing Systems*, 2019.

Shuxuan Guo, Jose M. Alvarez, and Mathieu Salzmann. ExpandNets: Linear over-parameterization to train compact convolutional networks. arXiv, 2020.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE International Conference on Computer Vision*, 2015.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

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*, 2018.

Yerlan Idelbayev and Miguel A. Carreira-Perpiñán. Low-rank compression of neural nets: Learning the rank of each layer. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2020.

Yani Ioannou, Duncan Robertson, Jamie Shotton, Roberto Cipolla, and Antonio Criminisi. Training CNNs with low-rank filters for efficient image classification. In *Proceedings of the 4th International Conference on Learning Representations*, 2016.

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*, 2015.

Alex Krizhevksy. Learning multiple layers of features from tiny images. Technical report, 2009.

Yann LeCun, John S. Decker, and Sara A. Solla. Optimal brain damage. In *Advances in Neural Information Processing Systems*, 1990.

Namhoon Lee, Thalaiyasingam Ajanthan, and Philip H. S. Torr. SNIP: Single-shot network pruning based on connection sensitivity. In *Proceedings of the 6th International Conference on Learning Representations*, 2018.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *Proceedings of the 7th International Conference on Learning Representations*, 2019.

Dmytro Mishkin and Jiri Matas. All you need is a good init. In *Proceedings of the 4th International Conference on Learning Representations*, 2016.

Decebal Constantin Mocanu, Elena Mocanu, Peter Stone, Phuong H. Nguyen, Madeleine Gibescu, and Antonio Liotta. Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. *Nature Communications*, 9(1), 2018.

Marcin Moczulski, Misha Denil, Jeremy Appleyard, and Nando de Freitas. ACDC: A structured efficient linear layer. arXiv, 2015.

Hesham Mostafa and Xin Wang. Parameter efficient training of deep convolutional neural networks by dynamic sparse reparameterization. In *Proceedings of the 36th International Conference on Machine Learning*, 2019.

Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. MIT Press, 2012.

Preetum Nakkiran, Raziel Alvarez, Rohit Prabhavalkar, and Carolina Parada. Compressing deep neural networks using a rank-constrained topology. In *Proceedings of the 16th Annual Conference of the International Speech Communication Association*, 2015.

Behnam Neyshabur, Srinadh Bhojanapalli, and Nathan Srebro. A PAC-Bayesian approach to spectrally-normalized margin bounds for neural networks. In *Proceedings of the 6th International Conference on Learning Representations*, 2018.

NVIDIA. NVIDIA Ampere GA102 GPU Architecture, 2020. Whitepaper.

Ivan V. Oseledets. Tensor-train decomposition. *SIAM Journal on Scientific Computing*, 33(5):2295–2317, 2011.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, 2019.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2016.

Alex Renda, Jonathan Frankle, and Michael Carbin. Comparing rewinding and fine-tuning in neural network pruning. In *Proceedings of the 8th International Conference on Learning Representations*, 2020.

Andrew M. Saxe, James L. McClelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. In *Proceedings of the 2nd International Conference on Learning Representations*, 2014.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *Proceedings of the 3rd International Conference on Learning Representations*, 2015.

Nathan Srebro and Adi Shraibman. Rank, trace-norm and max-norm. In *Proceedings of the International Conference on Computational Learning Theory*, 2005.

Jingtong Su, Yihang Chen, Tianle Cai, Tianhao Wu, Ruiqi Gao, Liwei Wang, and Jason D. Lee. Sanity-checking pruning methods: Random tickets can win the jackpot. arXiv, 2020.

Cheng Tai, Tong Xiao, Yi Zhang, Xiaogang Wang, and Weinan E. Convolutional neural networks with low-rank regularization. In *Proceedings of the 4th International Conference on Learning Representations*, 2016.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, 2017.

Li Wan, Matthew Zeiler, Sixin Zhang, Yann LeCun, and Rob Fergus. Regularization of neural networks using DropConnect. In *Proceedings of the 30th International Conference on Machine Learning*, 2013.

Chaoqi Wang, Roger Grosse, Sanja Fidler, and Guodong Zhang. EigenDamage: Structured pruning in the Kronecker-factored eigenbasis. In *Proceedings of the 36th International Conference on Machine Learning*, 2019.

Chaoqi Wang, Guodong Zhang, and Roger Grosse. Picking winning tickets before training by preserving gradient flow. In *Proceedings of the 8th International Conference on Learning Representations*, 2020.

Atsushi Yaguchi, Taiji Suzuki, Shuhei Nitta, Yukinobu Sakata, and Akiyuki Tanizawa. Decomposable-Net: Scalable low-rank compression for neural networks. arXiv, 2019.

Greg Yang and Samuel S. Schoenholz. Mean field residual networks: On the edge of chaos. In *Advances in Neural Information Processing Systems*, 2017.

Huanrui Yang, Minxue Tang, Wei Wen, Feng Yan, Daniel Hu, Ang Li, Hai Li, and Yiran Chen. Learning low-rank deep neural networks via singular vector orthogonality regularization and singular value sparsification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2020.

Yang You, Jing Li, Sashank 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 *Proceedings of the 8th International Conference on Learning Representations*, 2020.

Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In *Proceedings of the British Machine Vision Conference*, 2016.

Wenyuan Zeng and Raquel Urtasun. MLPrune: Multi-layer pruning for automated neural network compression. OpenReview, 2019.

Guodong Zhang, Chaoqi Wang, Bowen Xu, and Roger Grosse. Three mechanisms of weight decay regularization. In *Proceedings of the 7th International Conference on Learning Representations*, 2019.

Yuge Zhang, Zejun Lin, Junyang Jiang, Quanlu Zhang, Yujing Wang, Hui Xue, Chen Zhang, and Yaming Yang. Deeper insights into weight sharing in neural architecture search. arXiv, 2020.

A Proof of Claim 3.1

Proof. Let $\rho_t = \|\boldsymbol{W}_t\|_F$ be the Frobenius norm of the composed matrix at time t. Applying the update rules for $\boldsymbol{U}_t$ and $\boldsymbol{V}_t$ and using the fact that $\rho_t \nabla_{\boldsymbol{W}_t} = \nabla_{\hat{\boldsymbol{W}}_t}$ yields

$$\begin{aligned} \boldsymbol{W}_{t+1} = (\boldsymbol{U}_t - \eta \nabla_{\boldsymbol{U}_t})(\boldsymbol{V}_t^T - \eta \nabla_{\boldsymbol{V}_t^T}) &= (\boldsymbol{U}_t - \eta \nabla_{\boldsymbol{W}_t} \boldsymbol{V}_t)(\boldsymbol{V}_t^T - \eta \boldsymbol{U}_t^T \nabla_{\boldsymbol{W}_t^T}) \\ &= \boldsymbol{W}_t - \frac{\eta}{\rho_t} \hat{\nabla}_t + \eta^2 \nabla_{\boldsymbol{W}_t} \boldsymbol{W}_t^T \nabla_{\boldsymbol{W}_t} \end{aligned} \tag{4}$$

Taking the squared norm of both sides yields $\rho_{t+1}^2 = \rho_t^2 - 2\eta \operatorname{Tr}(\hat{\boldsymbol{W}}_t^T \hat{\nabla}_t) + \mathcal{O}(\eta^2)$; we can then take the square root of both sides and use a Tayor expansion to obtain

$$\rho_{t+1} = \rho_t \sqrt{1 - \frac{2\eta}{\rho_t^2} \operatorname{Tr}(\hat{\boldsymbol{W}}_t^T \hat{\nabla}_t) + \mathcal{O}(\eta^2)} = \rho_t - \frac{\eta}{\rho_t} \operatorname{Tr}(\hat{\boldsymbol{W}}_t^T \hat{\nabla}_t) + \mathcal{O}(\eta^2) \tag{5}$$

Then, starting from Equation 4 divided by ρ_{t+1}, substituting Equation 5, and applying a Taylor expansion yields

$$\begin{aligned} \hat{\boldsymbol{W}}_{t+1} &= \frac{\rho_t}{\rho_{t+1}} \hat{\boldsymbol{W}}_t - \frac{\eta}{\rho_t \rho_{t+1}} \hat{\nabla}_t + \mathcal{O}(\eta^2) \\ &= \frac{1}{1 - \frac{\eta}{\rho_t^2} \operatorname{Tr}(\hat{\boldsymbol{W}}_t^T \hat{\nabla}_t)} \hat{\boldsymbol{W}}_t - \frac{\eta}{\rho_t^2 - \eta \operatorname{Tr}(\hat{\boldsymbol{W}}_t^T \hat{\nabla}_t)} \hat{\nabla}_t + \mathcal{O}(\eta^2) \\ &= \left(1 + \frac{\eta}{\rho_t^2} \operatorname{Tr}(\hat{\boldsymbol{W}}_t^T \hat{\nabla}_t)\right) \hat{\boldsymbol{W}}_t - \frac{\eta}{\rho_t^2} \hat{\nabla}_t + \mathcal{O}(\eta^2) \\ &= \hat{\boldsymbol{W}}_t - \frac{\eta}{\rho_t^2} (\hat{\nabla}_t - (\hat{\boldsymbol{w}}_t^T \operatorname{vec}(\hat{\nabla}_t)) \hat{\boldsymbol{W}}_t) + \mathcal{O}(\eta^2) \end{aligned} \tag{6}$$

Vectorizing and observing that $(\hat{\boldsymbol{w}}_t^T \operatorname{vec}(\hat{\nabla}_t)) \hat{\boldsymbol{w}}_t = \hat{\boldsymbol{w}}_t \hat{\boldsymbol{w}}_t^T \operatorname{vec}(\hat{\nabla}_t)$ yields the result. □

B Experimental Details for Training Convolutional Networks

For experiments with regular ResNets on CIFAR we used code provided here: `https://github.com/akamaster/pytorch_resnet_cifar10`. All hyperparameter settings are the same, except initialization and regularization as appropriate.

For comparisons with sparse model training of ResNet32$^{\text{x2}}$ and VGG19^{NFC} we use code by Wang et al. (2019, `https://github.com/alecwangcq/EigenDamage-Pytorch`), which is closely related to that of the lottery-ticket-guessing paper by Wang et al. (2020). All hyperparameter settings are the same, except (1) initialization and regularization as appropriate and (2) for Tiny-ImageNet we only train for 200 epochs instead of 300.

For comparisons with tensor decomposition training of WideResNet we use code by Gray et al. (2019, `https://github.com/BayesWatch/deficient-efficient`). All hyperparameter settings are the same, except initialization and regularization as appropriate.

C EXPERIMENTAL DETAILS FOR TRAINING TRANSFORMER MODELS

For experiments with Transformer models on machine translation we used code provided here: `https://github.com/StillKeepTry/Transformer-PyTorch`. All hyperparameter settings are the same, except initialization and regularization as appropriate.

For comparisons with LAMB optimization of BERT, we use an implementation provided by NVIDIA: `https://github.com/NVIDIA/DeepLearningExamples/tree/master/PyTorch/LanguageModeling/BERT`. All hyperparameter settings are the same, except initialization and regularization as appropriate. For fine-tuning on SQuAD we apply the same optimization routine to all pre-trained models.

Table 4: Pruning, sparse training, and low-rank training for VGG-19^{NFC}, i.e. the model of Simonyan & Zisserman (2015) with no fully-connected layers, as in Wang et al. (2020).

CIFAR-10	uncompressed	0.1	0.05	0.02
baseline*	94.23			
baseline†	93.70			
baseline (reproduced)	93.75±0.48			
OBD* LeCun et al. (1990)		93.74	93.58	93.49
MLPrune* (Zeng & Urtasun, 2019)		93.83	93.69	93.49
LT*, original initialization		93.51	92.92	92.34
LT*, reset to epoch 5		93.82	93.61	93.09
LR Rewinding† (Renda et al., 2020)		94.14±0.17	93.99±0.15	10.00±0.00
Hybrid Tickets† (Su et al., 2020)		94.00±0.12	93.83±0.10	93.52±0.28
DSR* (Mostafa & Wang, 2019)		93.75	93.86	93.13
SET* (Mocanu et al., 2018)		92.46	91.73	89.18
Deep-R* (Bellec et al., 2018)		90.81	89.59	86.77
SNIP* (Lee et al., 2018)		93.63±0.06	93.43±0.20	92.05±0.28
GraSP* (Wang et al., 2020)		92.59±0.10	91.01±0.21	87.51±0.31
Random Tickets† (Su et al., 2020)		93.77±0.10	93.42±0.22	92.45±0.22
Low-Rank		90.71±0.42	88.08±0.14	80.81±0.85
Low-Rank (SI)		90.99±0.34	88.61±0.95	82.53±0.14
Low-Rank (FD)		91.57±0.26	88.16±0.50	82.81±0.37
Low-Rank (SI & FD)		91.58±0.27	88.98±0.18	83.27±0.56
CIFAR-100	uncompressed	0.1	0.05	0.02
baseline*	74.16			
baseline†	72.60			
baseline (reproduced)	73.42±0.17			
OBD* LeCun et al. (1990)		73.83	71.98	67.79
MLPrune* (Zeng & Urtasun, 2019)		73.79	73.07	71.69
LT*, original initialization		72.78	71.44	68.95
LT*, reset to epoch 5		74.06	72.87	70.55
LR Rewinding† (Renda et al., 2020)		73.73±0.18	72.39±0.40	1.00±0.00
Hybrid Tickets† (Su et al., 2020)		73.53±0.20	73.10±0.11	71.61±0.46
DSR* (Mostafa & Wang, 2019)		72.31	71.98	70.70
SET* (Mocanu et al., 2018)		72.36	69.81	65.94
Deep-R* (Bellec et al., 2018)		66.83	63.46	59.58
SNIP* (Lee et al., 2018)		72.84±0.22	71.83±0.23	58.46±1.10
GraSP* (Wang et al., 2020)		71.95±0.18	71.23±0.12	68.90±0.47
Random Tickets† (Su et al., 2020)		72.55±0.14	71.37±0.09	68.98±0.34
Low-Rank		62.32±0.26	54.97±5.28	42.80±3.14
Low-Rank (SI)		63.77±0.93	58.51±1.90	45.02±3.24
Low-Rank (FD)		64.07±1.58	59.34±2.97	45.73±2.62
Low-Rank (SI & FD)		65.61±1.48	60.1±1.29	46.43±1.26
Tiny-ImageNet	uncompressed	0.1	0.05	0.02
baseline*	61.38			
baseline†	61.57			
baseline (reproduced)	62.19±0.74			
OBD* LeCun et al. (1990)		61.21	60.49	54.98
MLPrune* (Zeng & Urtasun, 2019)		60.23	59.23	55.55
LT*, original initialization		60.32	59.48	55.12
LT*, reset to epoch 5		61.19	60.57	56.18
DSR* (Mostafa & Wang, 2019)		62.43	59.81	58.36
SET* (Mocanu et al., 2018)		62.49	59.42	56.22
Deep-R* (Bellec et al., 2018)		55.64	52.93	49.32
SNIP* (Lee et al., 2018)		61.02±0.41	59.27±0.39	48.95±1.73
GraSP* (Wang et al., 2020)		60.76±0.23	59.50±0.33	57.28±0.34
Random Tickets† (Su et al., 2020)		60.94±0.38	59.48±0.43	55.87±0.16
Low-Rank		48.50±3.43	46.98±1.68	36.57±1.03
Low-Rank (SI)		49.87±1.39	46.73±2.83	37.25±0.19
Low-Rank (FD)		53.93±0.66	47.95±1.13	35.63±0.71
Low-Rank (SI & FD)		53.53±0.45	47.60±0.51	35.68±0.64

* Obtained from Wang et al. (2020).
† Obtained from Su et al. (2020).

Table 5: Pruning, sparse training, and low-rank training for ResNet-32^{x2}, i.e. the model of He et al. (2016) with twice the number of filters, as in Wang et al. (2020).

CIFAR-10	uncompressed	0.1	0.05	0.02
baseline*	94.80			
baseline†	94.62			
baseline (reproduced)	94.49±0.29			
OBD* LeCun et al. (1990)		94.17	93.29	90.31
MLPrune* (Zeng & Urtasun, 2019)		94.21	93.02	89.65
LT*, original initialization		92.31	91.06	88.78
LT*, reset to epoch 5		93.97	92.46	89.18
LR Rewinding† (Renda et al., 2020)		94.14±0.10	93.02±0.28	90.83±0.22
Hybrid Tickets† (Su et al., 2020)		93.98±0.15	92.96±0.13	90.85±0.06
DSR* (Mostafa & Wang, 2019)		92.97	91.61	88.46
SET* (Mocanu et al., 2018)		92.30	90.76	88.29
Deep-R* (Bellec et al., 2018)		91.62	89.84	86.45
SNIP* (Lee et al., 2018)		92.59±0.10	91.01±0.21	87.51±0.31
GraSP* (Wang et al., 2020)		92.38±0.21	91.39±0.25	88.81±0.14
Random Tickets† (Su et al., 2020)		92.97±0.05	91.60±0.26	89.10±0.33
Low-Rank		93.59±0.12	92.45±0.69	87.95±0.91
Low-Rank (SI)		92.52±0.66	91.72±0.63	87.36±0.65
Low-Rank (FD)		92.92±0.29	89.44±0.25	83.05±1.02
Low-Rank (SI & FD)		94.34±0.34	92.90±0.43	87.97±0.86
CIFAR-100	uncompressed	0.1	0.05	0.02
baseline*	74.64			
baseline†	74.57			
baseline (reproduced)	75.41±1.26			
OBD* LeCun et al. (1990)		71.96	68.73	60.65
MLPrune* (Zeng & Urtasun, 2019)		72.34	67.58	59.02
LT*, original initialization		68.99	65.02	57.37
LT*, reset to epoch 5		71.43	67.28	58.95
LR Rewinding† (Renda et al., 2020)		72.41±0.49	67.22±3.42	59.22±1.15
Hybrid Tickets† (Su et al., 2020)		71.47±0.26	69.28±0.40	63.44±0.34
DSR* (Mostafa & Wang, 2019)		69.63	68.20	61.24
SET* (Mocanu et al., 2018)		69.66	67.41	62.25
Deep-R* (Bellec et al., 2018)		66.78	63.90	58.47
SNIP* (Lee et al., 2018)		68.89±0.45	65.22±0.69	54.81±1.43
GraSP* (Wang et al., 2020)		69.24±0.24	66.50±0.11	58.43±0.43
Random Tickets† (Su et al., 2020)		69.70±0.48	66.82±0.12	60.11±0.16
Low-Rank		72.71±0.43	67.86±0.91	61.08±0.76
Low-Rank (SI)		71.71±1.46	67.41±1.12	60.79±0.26
Low-Rank (FD)		71.85±0.39	66.89±0.73	55.31±0.61
Low-Rank (SI & FD)		74.41±0.67	70.22±0.64	61.40±0.96
Tiny-ImageNet	uncompressed	0.15	0.1	0.1
baseline*	62.89			
baseline†	62.92			
baseline (reproduced)	63.02±0.94			
OBD* LeCun et al. (1990)		58.55	56.80	51.00
MLPrune* (Zeng & Urtasun, 2019)		58.86	57.62	51.70
LT*, original initialization		56.52	54.27	49.47
LT*, reset to epoch 5		60.31	57.77	51.21
DSR* (Mostafa & Wang, 2019)		57.08	57.19	56.08
SET* (Mocanu et al., 2018)		57.02	56.92	56.18
Deep-R* (Bellec et al., 2018)		53.29	52.62	52.00
SNIP* (Lee et al., 2018)		56.33±0.24	55.43±0.14	49.57±0.44
GraSP* (Wang et al., 2020)		57.25±0.11	55.53±0.11	51.34±0.29
Random Tickets† (Su et al., 2020)		N/A	55.26±0.22	51.41±0.38
Low-Rank		60.82±0.72	58.72±0.53	55.39±1.02
Low-Rank (SI)		59.53±1.41	57.60±0.88	55.00±0.90
Low-Rank (FD)		59.45±0.82	57.24±0.61	54.15±1.22
Low-Rank (SI & FD)		62.24±0.39	60.25±1.02	55.97±0.48

* Obtained from Wang et al. (2020).
† Obtained from Su et al. (2020).

Table 6: Comparison of low-rank and tensor-decomposition training (mean of 3 trials) of WideResNet28-10 on CIFAR-10. The best error for each compression-decomposition setting is **bolded**; in addition, the best error at each compression level is underlined.

compression	decomposition	decay ($\lambda = 0.005$)	initialization default	initialization spectral
none (≈36.5 param.)			**3.21±0.22**	
0.01667 (≈0.61M param.)	Low-Rank	regular	7.62±0.08	7.72±0.11
		CRS*	8.63±0.87	8.83±0.17
		Frobenius	**7.23±0.32**	7.26±0.45
	Tensor-Train	regular	8.55±4.22	7.72±0.46
		CRS*	7.01±0.35	6.52±0.17
		Frobenius	7.26±0.73	**6.22±0.50**
	Tucker	regular	10.67±7.68	9.91±6.71
		CRS*	8.28±0.05	**6.79±0.34**
		Frobenius	8.65±0.27	7.07±0.28
0.06667 (≈2.4M parameters)	Low-Rank	regular	5.31±0.06	5.49±0.42
		CRS*	5.86±0.22	5.53±0.72
		Frobenius	3.89±0.17	**3.86±0.22**
	Tensor-Train	regular	7.33±0.27	6.49±0.30
		CRS*	5.20±0.37	**4.79±0.15**
		Frobenius	5.02±0.15	5.10±0.35
	Tucker	regular	5.64±0.19	5.61±0.46
		CRS*	5.30±0.35	**5.15±0.23**
		Frobenius	5.30±0.12	5.24±0.21

* Regular weight-decay with coefficient scaled by the compression rate (Gray et al., 2019).

Table 7: Self-taught ResNet performance (median of 3 trials). Best at each depth is **bolded**; cases where we match the next deeper network with around the same *training* memory are underlined.

CIFAR-10 depth	factorization setting decay ($\lambda = 0.0001$)	base WD	full WD	full FD	deep WD	deep FD	wide WD	wide FD
110	test accuracy (%)	93.71						
	training param. (M)	1.73						
	test param. (M)	1.73						
56	test accuracy (%)	92.7	92.45	93.68	91.65	93.58	92.51	**93.89**
	training param. (M)	0.85	1.72	1.72	2.59	2.59	5.16	5.16
	test param. (M)	0.85	0.85	0.85	0.85	0.85	0.85	0.85
32	test accuracy (%)	92.55	92.32	93.31	91.11	92.99	92.09	**93.35**
	training param. (M)	0.46	0.95	0.95	1.43	1.43	2.84	2.84
	test param. (M)	0.46	0.46	0.46	0.46	0.46	0.46	0.46

CIFAR-100 depth	factorization setting decay ($\lambda = 0.0001$)	base WD	full WD	full FD	deep WD	deep FD	wide WD	wide FD
110	test accuracy (%)	71.93						
	training param. (M)	1.73						
	test param. (M)	1.73						
56	test accuracy (%)	70.15	69.23	72.18	68.25	72.09	69.98	**72.61**
	training param. (M)	0.86	1.73	1.73	2.60	2.60	5.17	5.17
	test param. (M)	0.86	0.86	0.86	0.86	0.86	0.86	0.86
32	test accuracy (%)	68.02	68.81	**70.59**	67.93	70.29	69.17	70.51
	training param. (M)	0.47	0.95	0.95	1.44	1.44	2.84	2.84
	test param. (M)	0.47	0.47	0.47	0.47	0.47	0.47	0.47