Exploiting Explainable Metrics for Augmented SGD

Mahdi S. Hosseini[1*] Mathieu Tuli[2*] Konstantinos N. Plataniotis[2]
[1]University of New Brunswick [2]University of Toronto

Code:https://github.com/mahdihosseini/RMSGD

Abstract

Explaining the generalization characteristics of deep learning is an emerging topic in advanced machine learning. There are several unanswered questions about how learning under stochastic optimization really works and why certain strategies are better than others. In this paper, we address the following question: can we probe intermediate layers of a deep neural network to identify and quantify the learning quality of each layer? With this question in mind, we propose new explainability metrics that measure the redundant information in a network's layers using a low-rank factorization framework and quantify a complexity measure that is highly correlated with the generalization performance of a given optimizer, network, and dataset. We subsequently exploit these metrics to augment the Stochastic Gradient Descent (SGD) optimizer by adaptively adjusting the learning rate in each layer to improve in generalization performance. Our augmented SGD – dubbed RMSGD – introduces minimal computational overhead compared to SOTA methods and outperforms them by exhibiting strong generalization characteristics across application, architecture, and dataset.

1. Introduction

The task of predicting network generalization performance using some measure of complexity based on training data is an emerging topic in the field of machine learning. Development of such "explainability" metrics is vitally important in order to understand and better explain the learning mechanisms involved in training of a given optimizer, network, and dataset. Identifying the causal relationship between some metric and generalization gap (or even testing accuracy directly) in order to select optimal network topologies or tune hyper-parameters is an important problem and actively researched today [10, 22, 23, 29, 36].

While the field of metric development for predicting generalization performance is growing (see related works in section 2), our interest in this work is to exploit such explainability metrics to augment the training of deep neural networks (DNNS). We achieve this by defining new metrics – *stable rank, condition number, and a quality measure* – derived from the intermediate layers of DNNS during training to access the true knowledge in the underlying weights and express how well the network layers are functioning as high-quality autoencoders for knowledge representation. Using these metrics, we exploit their ability to quantify learning in order to augment stochastic gradient descent (SGD) by dynamically adjusting the learning rate.

From a different lens, this work also sheds light on the behaviour of commonly practiced hyper-parameter tuning techniques like learning rate scheduling through decay methods [12, 14, 17, 32, 33, 57] or functional methods [16, 32, 41, 42]. Little is understood about such methods and why they really work and their use becomes more like *alchemy* rather than analytical/empirical reasoning. We highlight how our metrics provide reasonable explanation to these strategies and also how they can be used in a simple optimization framework to augment SGD and gain performance. Our main contributions are as follows:

[C1] We introduce new explainability metrics derived using training data that quantify layer-level learning of neural networks and are robust to the randomness of initialization.

[C2] We use these metrics to explain the training mechanisms of neural networks for various optimizers and datasets, and predict generalization characteristics in deep learning.

[C3] We exploit these metrics to augment SGD and introduce our new RMSGD optimizer, which gains considerable performance improvements at minimal computational cost and generalizes well across experimental configuration.

2. Related Works

A variety of complexity measures have been recently introduced for predicting network generalization such as ℓ_p-norm signatures from network weights in [36], margin distribution by measuring the distance between network training and decision bounds [22], and a gradient signal-to-noise-ratio (GSNR) measure from the evolution of training weights [29]. A more comprehensive analysis of related

*Equally major contribution

measures and exploring their dependencies to a variety of topological structures and datasets can be found in [10, 23]. A few other works also take further steps to train an estimator on pairs of signature and test accuracy using a regression model or fully-connected layers for high accuracy prediction of generalization gap [5, 52] and test accuracy [47]. A variant of such studies also utilizes a neural complexity measure as an additional regularizer to the loss function to accelerate training and attempts at improving the generalization gap [28]. While the above-mentioned metrics show strength in predicting the network generalization, they are either (a) defined as a function of overall network performance that cannot be probed in intermediate layers of deep network for performance measurement; or (b) are sensitive to the noise perturbations of the weight structures which can potentially lead to inconsistent behaviours as well as low correlation accuracy measures. Our proposed methods do not suffer from these drawbacks.

The field of stochastic optimization has grown considerably for training DNNS. Families of stochastic gradient descent (SGD) based optimizers have been introduced in [2, 3, 31, 37, 45, 55]. More advanced methods to increase the generalization of SGD are also studied in [12, 17]. Despite good generalization characteristics of the SGD based optimizers, tuning their associated hyper-parameters (such as learning rate) are the main bottlenecks to their use in practice. A variety of adaptive optimization algorithms are also introduced to leverage an adaptive stochastic minimization framework such as Adam [25], AdaBound [33] and AdamP [17]. While adaptive based optimizers are shown to work well across different applications such as computer vision (CV) and natural language processing, they generalize poorly to test data in CV applications [50]. Improvements are made in [1, 8, 30, 33, 48] to overcome this issue however, they still lack in performance when compared to SGD-based methods. Our proposed work builds on top of SGD and inherits its performance.

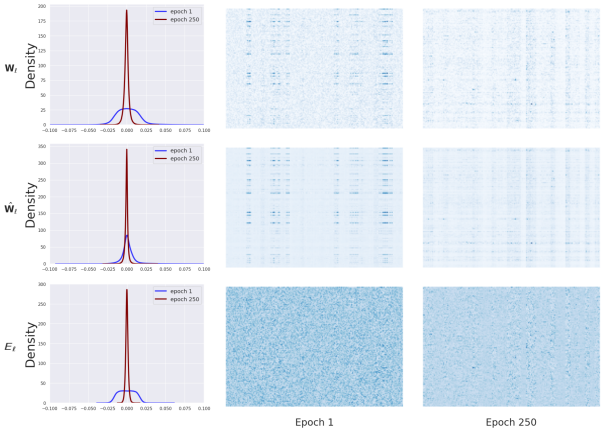
3. On Explainability Metrics

3.1. Low-Rank Factorization of Matrix Weights

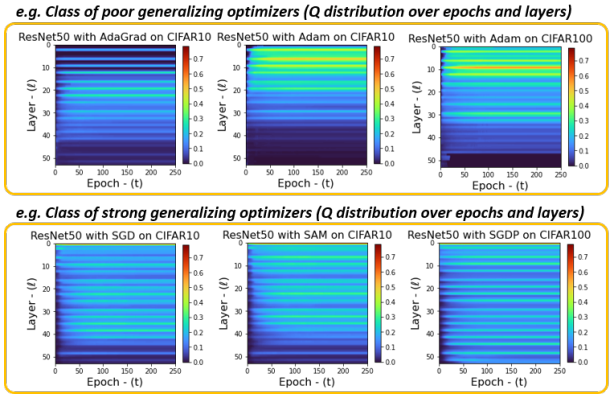
We argue that it is useful to decompose the weight matrices of the network being studied by low-rank factorization. This will allow us to analyze the underlying information that is learned during training. We first note that the weight matrices to be decomposed can be obtained in one of two forms: In the first form, if the weight matrix is part of a convolution layer (i.e. it is 4-dimensional), similar to [27], it can be unfolded as $\mathbf{W}_{4\text{D}} \in \mathbb{R}^{h\times w\times n_i\times n_o} \xrightarrow{\text{unfold}} \mathbf{W} \in \mathbb{R}^{m\times n}$, where w, h are the width and height of the convolution kernel and n_i, n_o correspond to the number of input and output channels, respectively. We unfold the tensor either on mode-3 (input channel) as $\mathbf{W} \in \mathbb{R}^{whn_o\times n_i}$ or mode-4 (output channel) as $\mathbf{W} \in \mathbb{R}^{whn_i\times n_o}$. In the second form, if the layer is a simple linear layer, we can directly utilize the 2D tensor $\mathbf{W} \in \mathbb{R}^{m\times n}$ of linear layers weights (e.g. fully-connected layer, Transformer weights, etc). Note in the case of linear layers, bias would be ignored. In both forms we assume $n \leq m$. We then obtain the low-rank structure by factorizing

$$\mathbf{W}_\ell\texttt{[Weight]} \xrightarrow{\text{fac.}} \widehat{\mathbf{W}}_\ell\texttt{[Low-Rank]} + \mathbf{E}_\ell\texttt{[Noise]} \quad (1)$$

where, $\widehat{\mathbf{W}}_\ell$ is the low-rank matrix containing limited non-zero singular values i.e. $\widehat{\mathbf{W}}_\ell = U\Lambda V^\top$, where $\Lambda = \text{diag}\{\sigma_1, \sigma_2, \ldots \sigma_{n'}\}$ and $n' = \text{rank}\ \widehat{\mathbf{W}}_\ell$. Here, $n' < \min(m, n)$ due to the low-rank property. For our experiments, we employ the Variational Baysian Matrix Factorization (VBMF) method [35] to perform the low-rank factorization. This method provides a global analytical solution and avoids an iterative algorithm by solving a quadratic minimization problem, meaning it is computationally efficient and can be easily applied to multiple layers of arbitrary size with minimal overhead (see Figure 4 for example).



(a) Low-Rank decomposition on layer $\ell = 25$ of ResNet34 applied to CIFAR10



(b) Quality (Q) Measure

Figure 1. (a) Implication of low-rank factorization on weight matrix taken from a particular layer of ResNet34; (b) Quality measure (Q) on ResNet50 trained with different optimizers.

The use of VBMF low-rank factorization allows our analysis to be robust to the randomness introduced by different initialization methods employed in training. This factorization is essential, and using techniques such as SVD directly would be heavily influenced by the presence of noise and prevent proper analysis. In the context of our application to augmenting SGD, performance would be seriously degraded.

Initially, the low-rank component of the weight matrix has an empty structure (i.e. $\widehat{\mathbf{W}}_\ell = \oslash$) as the randomness of the initialized weights is fully captured in the noise perturbing component E. As training progresses, the low-rank component becomes non-empty and start learning to develop meaningful mapping structure. Figure 1a demonstrates an intuitive example of this development from a ResNet34 layer. Notice how the low-rank structure is preserved during epoch training while the noise fades away. This highlights how training reduces the perturbing noise within the layers of a neural network and strengthens the useful information embedded in the low-rank structure. We state that this leads to a stabilized encoding layer.

3.2. Probing Metrics

Given this notion of low-rank factorization, we wish to now quantify how well a network layer encodes and propagates information. To this aim, we borrow two metrics from linear algebra matrix analysis: *stable rank* [4, 38] and *condition number* [19]. We apply these concepts to the low-rank factorized weight matrices discussed earlier.

The stable rank is the norm energy of the singular values of a given matrix. We propose a modified definition of the stable rank on the low-rank structure $\widehat{\mathbf{W}}_\ell \in \mathbb{R}^{m \times n}$ as

$$s(\widehat{\mathbf{W}}_\ell) = \frac{1}{n} \frac{\| \widehat{\mathbf{W}}_\ell \|_*}{\| \widehat{\mathbf{W}}_\ell \|_2} = \frac{1}{n\sigma_1^2(\widehat{\mathbf{W}}_\ell)} \sum_{i=1}^{n'} \sigma_i^2(\widehat{\mathbf{W}}_\ell), \quad (2)$$

where, $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_{n'}$ are the low-rank singular values in descending order and $\| \cdot \|_*$ stands for nuclear norm (also known as the Schatten norm). This metric encodes the significance of low-rank span in the output (feature) mapping space. A higher measure indicates a better encoder and stronger carriage of information through the layer's weight matrix. Note that we normalize the stable rank by the smaller input dimension n of the input matrix (since we assume $n \leq m$) to bound $s(\widehat{\mathbf{W}}_\ell) \in [0, 1]$.

The condition number is also defined as a relative ratio of the highest and lowest singular values. We modify this definition on the low-rank structure $\widehat{\mathbf{W}}_\ell \in \mathbb{R}^{m \times n}$ as

$$\kappa(\widehat{\mathbf{W}}_\ell) = 1 - \sigma_{n'}(\widehat{\mathbf{W}}_\ell)/\sigma_1(\widehat{\mathbf{W}}_\ell). \quad (3)$$

Note that $\kappa(\widehat{\mathbf{W}}_\ell) \in [0, 1]$. This metric indicates the numerical sensitivity of the weight matrix's mapping with respect to input noise perturbations. Lower condition indicates higher robustness to noise and better input-output mapping. A concept example is shown in Figure 3 for the evolution of these metrics across training epochs of a layer in ResNet18. Note also that our metric development does not consider components such as skip connections. We argue intuitively that the influence of components such as skip connections gets captured in the weight matrices of nearby layers as it learns, and by thus analyzing the matrices themselves is a sufficient task, which we show holds in our experiments.

3.3. On the Meaning of Probing Metrics

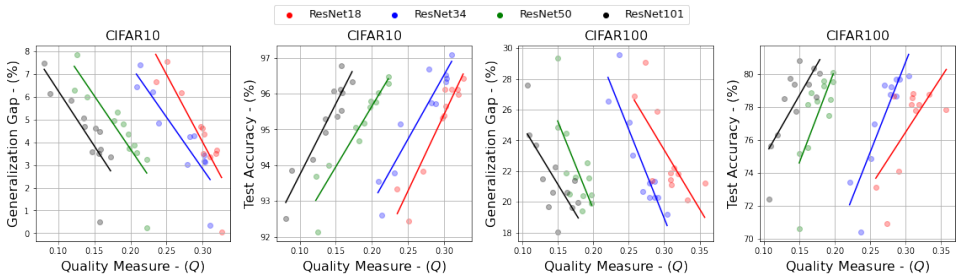
Given the definitions of stable rank in Equation 2 and condition number in Equation 3, we argue that a stable rank of 1 and condition number of 0 indicate a perfectly learned network. Specifically, for the stable-rank, higher value $s(\widehat{\mathbf{W}}_\ell) \to 1$ indicates that most singular values are non-zero (i.e. $\sigma_i^2(\widehat{\mathbf{W}}_\ell) > 0 \forall i \in [1, \ldots, n']$ where $n' \to n$. This creates a subspace spanned by a set of independent vectors corresponding to the non-zero singular values mentioned above. In other words, $s(\widehat{\mathbf{W}}_\ell) \to 1$ corresponds to a many-to-many mapping but not a many-to-low (i.e. rank-deficient) mapping. Also, note that the stable rank is measured on the low-rank and not the raw measure of the weights. So the higher value indicates that the *learned* weight matrix contains more non-empty structure which can be interpreted as a sign of meaningful learning.

For the condition number, we note that this metric also defines the numerical sensitivity of the inverse matrix $\widehat{\mathbf{W}}_\ell$ toward minor input pertubations. Note that the error residual reconstruction under the linear system $y = \widehat{\mathbf{W}}_\ell x$ will be bounded by $\| x - \hat{x} \| / \| x \| < c\sigma_1(\widehat{\mathbf{W}}_\ell)/\sigma_{n'}(\widehat{\mathbf{W}}_\ell)$, where x and $\hat{x}$ are the underlying and recovered signals, respectively, and c is some constant measure from the linear system. We defer to [19] for more information. Thus, the mapping condition has a direct impact on the residual reconstruction. This is specifically important during DNN training, where gradients are back-propagated and the matrix weights are involved in adjoint form for parameters updates. If the condition number is low (i.e. $\kappa(\widehat{\mathbf{W}}_\ell) \to 0$) then the noise perturbations from gradients will be accumulated during the iterative training phase and accordingly it yields a poor learned weight matrix for space mapping (i.e. the encoding).

Given this understanding, we the wish to incorporate both metrics into a single probing measure to quantify quality of learning. We tackle this in the following section.

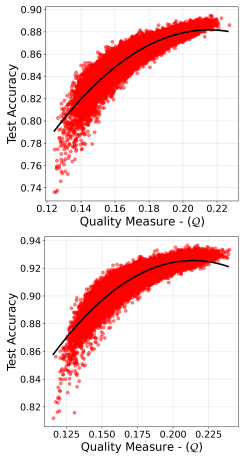
3.4. Quality Measure on Learned Network

We now aim to combine stable rank and condition number to develop a new quality measure that can help us quantify the quality of a learned network and describe its generalization characteristics. We know from subsection 3.3 that a stable rank of 1 and condition number of 0 would indicate a



Network	CIFAR10				CIFAR100			
	PLCC (%)		ROCC (%)		PLCC (%)		ROCC (%)	
	Gen. Gap	Test Acc.	Gen. Gap	Test Acc.	Gen. Gap	Test Acc.	Gen. Gap	Test Acc.
ResNet18	88.18	88.18	74.54	74.54	69.09	47.27	52.73	34.55
ResNet34	83.63	81.82	70.90	67.27	91.82	90.00	78.18	74.54
ResNet50	97.90	97.90	90.90	90.90	72.03	79.02	57.57	66.66
ResNet101	88.81	88.81	78.79	78.79	67.13	74.13	51.52	57.58

(a) Q Metric on ResNets



(b) Q Metric on NATS Benchmark

Figure 2. Our proposed Q measure against generalization gap and test accuracy. (a) Applied on numerous optimizers and hyper-parameter settings (see Table 1a) (dots represent a different experimental setup) on ResNets, with associated PLCC and ROCC correlations for our Q measure and generalization gap (Gen. Gap) and test accuracy (Test Acc.). We consider Q at the last epoch of training. (b) Applied on the NATS benchmark [7] on CIFAR10, where the top figure visualizes epoch 12 and the bottom visualizes epoch 90 of training.

perfectly learned network. We propose the following quality measure to capture these properties:

$$q(\widehat{\mathbf{W}}_\ell) = \arctan s(\widehat{\mathbf{W}}_\ell)/\kappa(\widehat{\mathbf{W}}_\ell), \text{ where } \ell \in [L] \quad (4)$$

where $\arctan(\cdot)$ is the element-wise arctangent, and q is bound by $[0, \frac{\pi}{2}]$. Further, q is maximized when $s \to 1$ and $\kappa \to 0$. Figure 1b visualizes the evolution of this quality measure on different conv layers of ResNet34 trained by different optimizers. The figure highlights the behaviour of this metric in response to poor and strong generalization performances. For instance, the Adam optimizer is known to provide poor generalization performance in CV applications, while the family of SGD-based optimizers are known to yield better performance [12, 17, 50]. We argue that this lack of performance from adaptive optimizers is realized by the low quality measure on latter layers of the ResNet34 model; this is visualized in Figure 1b by the darkness in the bottom of the figure. In contrast, the well performing SGD-based optimizers result in higher quality measure more consistently throughout the network, indicating better learned weights at all stages in the network.

We can define the overall quality measure of the network by aggregating all layers using an L_2-norm

$$Q = 1/\sqrt{L} \parallel \underline{\mathbf{q}} \parallel_2^2, \text{ where } \underline{\mathbf{q}} = [q(\widehat{\mathbf{W}}_1), \cdots, q(\widehat{\mathbf{W}}_L)]. \quad (5)$$

Normalizing by the square root of the number of layers accounts for the summation over layers within the ℓ_2-norm space. We found this normalization method to behave better than a simple mean.

We perform a small study on our quality metric against the popular NATS Benchmark [7] on CIFAR10 as well on a collection of ResNets on CIFAR10 and CIFAR100. We visualize our metric and associated correlation coefficients in Figure 2a & Figure 2b. The NATS Benchmark is a Neural Architecture Search benchmark that provides model checkpoints at epoch 12 and epoch 90 in training, and whose goal is to predict performance/characterize generalization. These model checkpoints are of the 32,768 different topological models generated in the benchmark, and each involve a range of model sizes/complexities. As for the ResNets study, we employed a similar technique as Google's generalization prediction using margin distributions [22] whereby we trained various ResNets with various optimizers and computed our measure for each. The plots and table highlight how this metric is a strong indicator for both generalization performance as well as test performance, and most networks have very strong correlation scores.

Finally, we highlight that this explainability metric is (a) derived using only training data, meaning it can be used to predict the network's generalization to unseen data; (b) can quantify the contribution and strength of each layer in the network, not just the layer as a whole as in previous methods [22, 29, 36]; and (c) is unaffected by random initialization due to low-rank factorization. Note that these points also apply inherently to stable rank and condition number. This can potentially shed light on new layer-based optimization methods, one of which we explore in section 4.

4. Augmenting SGD by Probing Metrics

Our goal in this section is to exploit the probing metrics developed in subsection 3.2 and develop an augmented SGD optimization algorithm that improves model performance.

4.1. New Updating Mechanism on Vanilla SGD

Recall the SGD training objective to minimize an associated loss function given a training dataset $f(\mathbf{W}_\ell; (X)^{\texttt{train}})$ [2, 3, 31, 37, 55]. The update rule is then given by

$$\mathbf{W}_\ell^k \longleftarrow \mathbf{W}_\ell^{k-1} - \eta_k \overline{\nabla f}_k(\mathbf{W}_\ell^{k-1}) \tag{6}$$

for $k \in \{(t-1)K+1, \cdots, tK\}$ where t and K correspond to epoch number and number of mini-batches, respectively, $\overline{\nabla f}_k(\mathbf{W}_\ell^{k-1}) = 1/|\Omega_k| \sum_{i \in \Omega_k} \nabla f_i(\mathbf{W}_\ell^{k-1})$ is the average stochastic gradients on kth mini-batch that are randomly selected from a batch of n-samples $\Omega_k \subset \{1, \cdots, n\}$, and η_k defines the step-size taken toward the opposite direction of average gradients (i.e. the learning rate).

We aim to update the learning rate independently for each network layer after every epoch and therefore the step-size will be a function of epoch index and layer i.e. $\eta_k \equiv \eta_\ell(t)$. We now setup our problem by accumulating all observed gradients throughout K mini-batch updates in one epoch as

$$\mathbf{W}_\ell^t = \mathbf{W}_\ell^{t-1} - \eta_\ell(t-1) \overline{\nabla f}_\ell^t, \tag{7}$$

where, $\overline{\nabla f}_\ell^t = \sum_{k=(t-1)K+1}^{tK} \overline{\nabla f}_k(\mathbf{W}_\ell^{k-1})$ corresponds to the total accumulated gradients in one training epoch. The idea here is to select a learning rate $\eta_\ell(t)$ such that the stable rank is increased over each epoch i.e. $\psi = \{\eta_\ell(t) : s(\widehat{\mathbf{W}}_\ell^t) \geq s(\widehat{\mathbf{W}}_\ell^{t-1})\}$ in order to learn better encoding layers.

Theorem 1 *(Increasing Stable Rank for Vanilla SGD). Let the stable rank to be defined by Equation 2. Starting with an initial learning rate $\eta_\ell(0) > 0$ and setting the step-size of vanilla Stochastic Gradient Descent (SGD) proportional to*

$$\eta_\ell(t) \triangleq \zeta \left[s(\widehat{\mathbf{W}}_\ell^t) - s(\widehat{\mathbf{W}}_\ell^{t-1}) \right] \tag{8}$$

will guarantee the monotonic increase of the stable rank for the next epoch update $s(\widehat{\mathbf{W}}_\ell^{t+1}) \geq s(\widehat{\mathbf{W}}_\ell^t)$ for some existing lower bound $\eta(t) \geq \eta_0$ and $\zeta \geq 0$.

The proof of Theorem 1 is provided in the supplementary material (Appendix-A).

With the start of a positive initial learning rate $\eta_\ell(0) > 0$ and following the update rule in Equation 8, Theorem 1 guarantees the increase of the stable rank for the next epoch update for Vanilla SGD. Accordingly, learning rates using Equation 8 will remain positive over consecutive epochs. For proof of demonstration, refer to Figure 3. We note in Figure 3 that the SGD type training degrades in stable rank over time, as the network learns more, while in contrast our method RMSGD doesn't. The idea is to have a high stable rank after training has completed, which we can see that SGD's algorithm does not exhibit stable behaviour. Further, SGD has higher stable rank, but worse (i.e. higher) condition number. In contrast, RMSGD has a good stable rank, as well as a much better condition number (i.e. lower).

4.2. RMSGD: Augmented SGD Algorithm

The step-size defined in subsection 4.1 for Vanilla SGD is measured only for two consecutive epochs. Due to the stochastic nature of gradients, the in-practice step-size can fluctuate. To reduce step-size oscillations, we employ a momentum algorithm for the historical accumulation of step-sizes and stable ranks over epoch updates. We augment the update rule for SGD by performing the following:

(i) revise learning rate by average momentum:
$\eta_\ell(t) \leftarrow \beta \eta_\ell(t-1) + \zeta[s(\widehat{\mathbf{W}}_\ell^t) - s(\widehat{\mathbf{W}}_\ell^{t-1})]$,
(ii) apply gradients through an average momentum
$\boldsymbol{v}_\ell^k \leftarrow \alpha \boldsymbol{v}_\ell^{k-1} - \eta_\ell(t) \boldsymbol{g}_\ell^k$,
(iii) update network weights:
$\boldsymbol{w}_\ell^k \leftarrow \boldsymbol{w}_\ell^{k-1} + v_\ell^k$,

where, k, t and ℓ correspond to the current mini-batch, current epoch, and network layer indexes, respectively. $\boldsymbol{v}$ is the velocity term, and $\boldsymbol{w}$ are the learnable parameters. Notice there are two associated momentum parameters: (1) SGD momentum fixed at $\alpha = 0.9$; and (2) learning momentum fixed at $\beta = 0.98$. We pick $\zeta = 1$ and learning momentum $\beta < 1$ to remain within the convergence bound of the unit circle associated with the update rule (i). Setting β trades-off between faster convergence and increased stable rank, which eventually leads to higher performance. An ablative study of this parameter is shown in Figure 3. We provide further study of this parameter in Appendix-B. We dub this optimizer **R**ank **M**omentum **SGD**: **RMSGD**[1], and show pseudo-code in Algorithm 1. The condition number plays no role in this algorithm, but we introduced it previously as an additional metric to evaluate/quantify learning. We show here that just one of our metrics is sufficient to develop our algorithm, and future work might include combining both.

Each layer in the network is assigned an index $\{\ell\}_{\ell=1}^L$ where all learnable parameters (e.g. conv, linear transform, biases, batch-norms, etc) are called using this index. The goal in RMSGD is first to callback the matrix weights of each layer, second to compute the stable rank $s(\widehat{\mathbf{W}}_\ell^t)$ of each layer using Equation 2, third to compute the difference gain using Equation 8, and finally accumulate this difference value in a decaying momentum fashion. This is performed to compute the learning rate of each layer independently for the current epoch t to augment the SGD optimization framework. We note here that the initial stable rank is zero $s(\widehat{\mathbf{W}}_\ell^0) = 0$ for all layers due to random initialization of weights. Figure 3 demonstrates an evolution example of the learning rate adapted by RMSGD. Interestingly, the behavior of learning rate coincides with scheduling techniques using cyclical learning in [41, 42] and warm-up/cosine decay in [16]. We believe our method explains the intuition behind

[1] Code from https://github.com/mahdihosseini/RMSGD

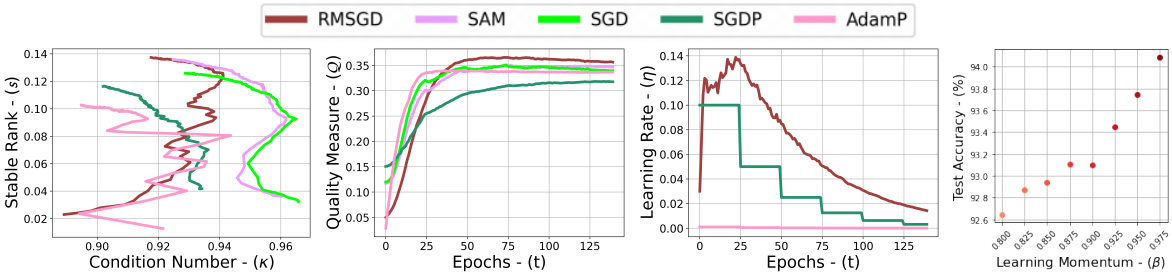


Figure 3. Comparison of (left) explainability metrics, (middle-left) Quality Measure, and (middle-right) learning rates for optimizers on ResNet50 applied to CIFAR10 over 140 epochs. (right) Effect of varying the learning momentum factor β on testing accuracy for ResNet34 on CIFAR10.

Algorithm 1: Augmented SGD (RMSGD)

Require: batch size $|\Omega_k|$, # epochs T, # layers L, initial step-sizes $\{\eta_\ell(0) = 0.03\}_{\ell=1}^{L}$, initial vectors $\{v_\ell^0, \boldsymbol{w}_\ell^0\}_{\ell=1}^{L}$, SGD momentum $\alpha = 0.9$, learning momentum $\beta = 0.98$, $\zeta = 1$

for $t = 1 : T$ **do**

Stage-I: SGD optimization augmented by adaptive learning rates

- generate K mini-batches: $\{\Omega_k \subset \{1, \cdots, n\}\}_{k=1}^{K}$

for $k = (t-1)K + 1 : tK$ **do**

1. compute gradients: $\boldsymbol{g}_\ell^k \leftarrow 1/|\Omega_k| \sum_{i \in \Omega_k} \nabla f_i(\boldsymbol{w}_\ell^{k-1}) \, ; \ell \in \{1, \cdots, L\}$
2. compute velocity terms: $\boldsymbol{v}_\ell^k \leftarrow \alpha \boldsymbol{v}_\ell^{k-1} - \eta_\ell(t) \boldsymbol{g}_\ell^k$
3. apply updates: $\boldsymbol{w}_\ell^k \leftarrow \boldsymbol{w}_\ell^{k-1} + \boldsymbol{v}_\ell^k$

end for

Stage-II: adaptive computation of learning rates

for $\ell = 1 : L$ **do**

1. Factorize matrix weights $\widehat{\mathbf{W}}_\ell^t$ by Equation 1 using `EVBMF` [35]
2. compute stable rank $s(\widehat{\mathbf{W}}_\ell)$ by Equation 2
3. update LR: $\eta_\ell(t) \leftarrow \beta \eta_\ell(t-1) + \zeta[s(\widehat{\mathbf{W}}_\ell^t) - s(\widehat{\mathbf{W}}_\ell^{t-1})]$

end for

end for

such scheduling technique and why the learning rate is suggested to start from small value and increase with proceeding epochs (i.e. warm-up), and then decay to finalize training.

Proposition 1 *(Convergence Guarantees of RMSGD). Given RMSGD only evolves the learning rate and ensures that $\eta_\ell(t) > 0 \; \forall t \in \{1, \ldots, T\}$, its convergence guarantee follows that of SGD [31, 56].*

We further elaborate on the proof in Appendix-A.

5. Empirical Evaluation

Setup. We evaluate RMSGD against state-of-the-art adaptive and non-adaptive optimizers and conduct different studies including (a) image classification on CIFAR10/CIFAR100 [26], ImageNet [39], as well as on two computational pathology datasets (MHIST [49] and ADP [20]). Hyper-parameters are tuned using ResNet18 on CIFAR10; (b) image classification with Cutout [6]; and (c) batch size robustness. Grid search is used for learning rate tuning. For full details on data augmentation, hyper-parameter tuning, and additional results, see Appendix-B.

Hardware. A single-GPU (RTX2080Ti) was used for each experiment, see Appendix-B for specifics.

Note. For all tables, green shows the best result and orange is within standard deviation from best.

5.1. Image Classification

Note on CIFAR experiments. Since different optimizers exhibit widely different epoch times (see Figure 4), for a fair comparison, we limit training to the total wall clock time consumed by 250 epochs using SGD. In terms of epochs, this amounts to $\sim$ 250 epochs for all optimizers except SAM, which consumes only $\sim 128 - 133$ epochs due to its 2 forward passes and twice as long epoch times.

Computer Vision: CIFAR. We report all results in Table 1a and some in Figure 4, with the other figures in Appendix-B. RMSGD performs consistently optimally, and shows strength with increasing dataset complexity (CIFAR10 $\rightarrow$ CIFAR100) and increasing network complexity (ResNet18 $\rightarrow$ ResNet34 $\rightarrow$ ResNet50 $\rightarrow$ ResNet101). We note that SGDP and SAM do remain within standard deviations of performance in many cases, which highlights the competitiveness of these optimizers. Note that for other adaptive optimizers, the competitiveness is non-existent.

We further highlight the performance to time results shown in Figure 4. Despite SAM's competitive performance, it consumes $\sim 2\times$ as many seconds per epoch to train. We highlight how RMSGD is able to outperform while incurring very low computational overhead. SAM does exhibit better generalization by its lower train-test gap. Note that for scenarios where additional resources or time is available, we also report final performance of all optimizers at epoch-250 in Appendix-B.

Computer Vision: CIFAR with Cutout. We also compare RMSGD using cutout, as one may argue that cutout can be used to augment SGD and negate the need for adaptive optimizers. However, we can see from Table 1b that RMSGD with cutout is still able to consistently outperform SAM,

Table 1. Perfomance of various networks and optimizers on CIFAR10 and CIFAR100 (a) without Cutout and (b) with Cutout. Reported using wall clock time of 250 SGD training epochs as the cutoff. Note ResNet$\{18, 34, 50, 101\}$=R$\{18, 34, 50, 101\}$ and ResNeXt=RNeXt.

(a) Without Cutout

	Network	AdaBelief [57]	AdaBound [33]	AdaGrad [9]	Adam [25]	AdamP [17]	SLS [48]	SAM [12]	SGD [14]	SGDP [17]	RMSGD
CIFAR10	R18 [15]	$93.34_{0.10}$	$93.84_{0.09}$	$92.45_{0.24}$	$93.27_{0.10}$	$94.82_{0.10}$	$93.62_{0.10}$	**$95.58_{0.07}$**	$95.32_{0.07}$	**$95.39_{0.16}$**	**$95.66_{0.17}$**
	R34 [15]	$93.55_{0.05}$	$93.79_{0.19}$	$92.59_{0.30}$	$93.47_{0.18}$	$95.14_{0.25}$	$93.45_{0.16}$	**$95.81_{0.16}$**	**$95.56_{0.10}$**	**$95.75_{0.14}$**	**$95.71_{0.07}$**
	R50 [15]	$93.70_{0.18}$	$94.00_{0.15}$	$92.12_{0.23}$	$92.67_{0.12}$	$94.69_{0.10}$	$92.70_{0.18}$	$95.20_{0.18}$	$95.05_{0.28}$	$95.19_{0.15}$	**$95.63_{0.05}$**
	R101 [15]	$93.86_{0.20}$	$94.17_{0.13}$	$92.51_{0.22}$	$93.13_{0.08}$	$94.92_{0.24}$	$64.20_{20.98}$	**$95.40_{0.12}$**	**$95.30_{0.13}$**	**$95.36_{0.04}$**	**$95.53_{0.14}$**
	RNeXt [51]	$93.55_{0.05}$	$92.83_{0.14}$	$91.09_{0.19}$	$91.78_{0.16}$	$93.82_{0.10}$	$93.67_{0.09}$	$94.38_{0.09}$	$94.62_{0.09}$	$94.79_{0.24}$	**$95.49_{0.05}$**
CIFAR100	R18	$73.11_{0.21}$	$74.09_{0.27}$	$70.92_{0.31}$	$72.45_{0.34}$	$76.81_{0.31}$	$73.59_{0.04}$	$77.16_{0.25}$	$77.80_{0.07}$	**$78.13_{0.16}$**	**$78.63_{0.34}$**
	R34	$73.43_{0.14}$	$74.84_{0.18}$	$70.39_{0.57}$	$72.09_{0.50}$	$76.93_{0.40}$	$73.22_{0.11}$	$77.98_{0.39}$	$77.88_{0.39}$	$78.74_{0.12}$	**$79.32_{0.10}$**
	R50	$75.15_{0.45}$	$75.52_{0.37}$	$70.60_{0.91}$	$70.53_{0.36}$	$77.47_{0.16}$	$75.80_{0.23}$	$77.39_{0.66}$	$78.12_{0.42}$	$78.44_{0.24}$	**$79.59_{0.54}$**
	R101	$75.63_{0.10}$	$76.31_{0.41}$	$72.39_{0.84}$	$72.20_{0.68}$	$77.71_{0.16}$	$73.31_{0.84}$	$78.38_{0.48}$	$78.48_{0.45}$	**$78.60_{0.55}$**	**$79.36_{0.26}$**
	RNeXt	$72.64_{0.49}$	$72.97_{0.38}$	$68.83_{0.43}$	$71.54_{0.41}$	$74.54_{0.40}$	$72.35_{0.42}$	$75.83_{0.30}$	$75.36_{0.33}$	**$76.56_{0.33}$**	**$77.14_{0.31}$**

(b) With Cutout

Network	CIFAR10				CIFAR100			
	SAM^C	SGD^C	SGDP^C	RMSGD^C	SAM^C	SGD^C	SGDP^C	RMSGD^C
ResNet18	**$95.96_{0.13}$**	**$96.12_{0.13}$**	**$96.13_{0.13}$**	**$96.13_{0.08}$**	$77.58_{0.11}$	$78.16_{0.21}$	**$78.82_{0.37}$**	**$78.53_{0.22}$**
ResNet34	**$96.64_{0.09}$**	**$96.53_{0.13}$**	**$96.70_{0.10}$**	$96.42_{0.08}$	$78.57_{0.19}$	$78.63_{0.55}$	**$79.67_{0.24}$**	**$79.70_{0.19}$**
ResNet50	$95.79_{0.10}$	$95.78_{0.27}$	$96.03_{0.16}$	**$96.28_{0.07}$**	$77.73_{0.28}$	$78.36_{0.67}$	**$79.52_{0.31}$**	**$80.06_{0.45}$**
ResNet101	**$96.17_{0.08}$**	$96.04_{0.16}$	$96.12_{0.05}$	**$96.33_{0.08}$**	**$79.41_{0.66}$**	$79.35_{0.62}$	**$80.03_{0.67}$**	**$80.36_{0.35}$**
ResNeXt	$95.01_{0.18}$	$95.04_{0.18}$	$95.24_{0.15}$	**$95.62_{0.08}$**	$76.34_{0.06}$	$75.91_{0.19}$	$77.14_{0.21}$	**$78.06_{0.28}$**
MobileNetV2 [40]	$94.65_{0.12}$	$94.53_{0.16}$	$94.07_{0.07}$	**$95.48_{0.11}$**	$75.43_{0.18}$	$73.94_{0.21}$	$73.40_{0.07}$	**$76.36_{0.27}$**
SENet18 [21]	**$95.92_{0.16}$**	**$95.99_{0.10}$**	**$96.04_{0.08}$**	$95.80_{0.06}$	**$77.64_{0.32}$**	**$77.80_{0.23}$**	**$77.70_{0.05}$**	**$77.77_{0.15}$**
EfficientNetB0 [46]	$90.44_{0.23}$	$91.70_{0.24}$	$92.09_{0.21}$	**$92.83_{0.14}$**	**$69.10_{0.50}$**	$68.42_{0.24}$	**$68.98_{0.44}$**	**$69.87_{0.49}$**
ShuffleNetV2 [34]	**$94.71_{0.15}$**	$94.40_{0.21}$	$94.37_{0.12}$	**$95.01_{0.29}$**	**$74.70_{0.19}$**	$74.13_{0.35}$	**$74.44_{0.29}$**	**$74.38_{0.36}$**

Table 2. Test accuracy results for MobileNetV2 on ImageNet using a batch size of 128, trained a single GPU machine.

Epoch	SAM		SGD		RMSGD	
	Top-1	Top-5	Top-1	Top-5	Top-1	Top-5
150	61.41	83.86	59.80	82.56	**67.84**	**88.32**
250	63.43	85.24	62.16	84.21	**70.25**	**89.66**

Table 3. Test accuracy results for MobileNetV2 and ResNet50 on ImageNet using a batch size of 256, trained a single GPU machine.

Network	SAM	SGD	AdamP	RMSGD
MobileNetV2	63.38	64.61	69.40	**71.24**
ResNet50	75.51	76.12	75.85	**76.42**

Table 4. Test accuracy results for computational pathology datasets. Note that ResNet18=R18, ResNet34=R34, and MobileNetV2=MV2

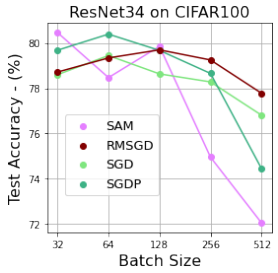
	Net	Adam	AdamP	SAM	SGD	SGDP	RMSGD
MHIST	R18	**$79.92_{0.80}$**	**$80.57_{1.43}$**	**$80.78_{1.61}$**	**$80.76_{1.32}$**	**$80.53_{0.65}$**	**$81.80_{1.20}$**
	R34	**$81.02_{0.95}$**	**$80.57_{0.88}$**	**$80.72_{1.12}$**	**$79.65_{1.98}$**	**$80.16_{0.85}$**	**$80.94_{0.85}$**
	MV2	**$79.90_{0.95}$**	**$80.59_{0.52}$**	**$82.89_{0.93}$**	**$81.41_{0.81}$**	**$80.80_{1.58}$**	**$82.58_{0.64}$**
ADP	R18	$92.75_{0.20}$	$94.04_{0.12}$	$93.28_{0.12}$	$93.22_{0.17}$	$93.48_{0.43}$	**$94.27_{0.10}$**
	R34	$92.80_{0.06}$	**$93.95_{0.08}$**	$93.24_{0.11}$	$93.38_{0.14}$	$93.65_{0.12}$	**$94.19_{0.21}$**
	MV2	$92.89_{0.17}$	**$93.78_{0.09}$**	$89.43_{1.46}$	$88.81_{1.32}$	$91.42_{0.53}$	**$93.83_{0.28}$**

SGD, and SGDP, and incurs great improvements ($\sim 1\%$) over its non-cutout performance.

Computer Vision: ImageNet. We report ImageNet results in Table 2, Table 3, and Figure 4. The $\sim 7\%$ top-1 test accuracy performance gap highlights RMSGD's strength over SGD and SAM, even with no ImageNet-specific hyperparameter tuning. This highlights how RMSGD may be used to tackle larger datasets with smaller computational resources. SGD and RMSGD took 1 week to train, while SAM took 2. The performance differences from the original MobileNetV2 [40] relate to the smaller batch size of 128.

Computational Pathology: MHIST & ADP We report our computational pathology results in Table 4 and some shown in Figure 4. We show that RMSGD is able to outperform all optimizers on ADP. All optimizers tend to perform similarly on MHIST, likely due to its smaller size. Computational pathology dataset are included in the experiments as an additional (less conventional) dataset that pose additional challenges (e.g. difficulty and scarcity of labels, image size).

Batch Size. We report our ablative batch size study in Figure 4. We highlight RMSGD's greater robustness to varying batch sizes, particularly compared to SAM. SGD exhibits robustness but has inferior performance.



6. Concluding remarks

In this work we introduced probing metrics (i.e. stable rank, condition number, and a quality measure) and demonstrated how they can be used to quantify learning of neural networks and be used as indicators for generalization performance. We demonstrated how these metrics can be exploited very simply to augment vanilla SGD and achieve significant performance gains with extremely low computational overhead ($<$ 1s per training epoch).

Computational cost vs. performance benefits. We highlight that RMSGD is able to remain extremely computationally efficient, while incurring performance benefits across many applications, network complexities, and dataset

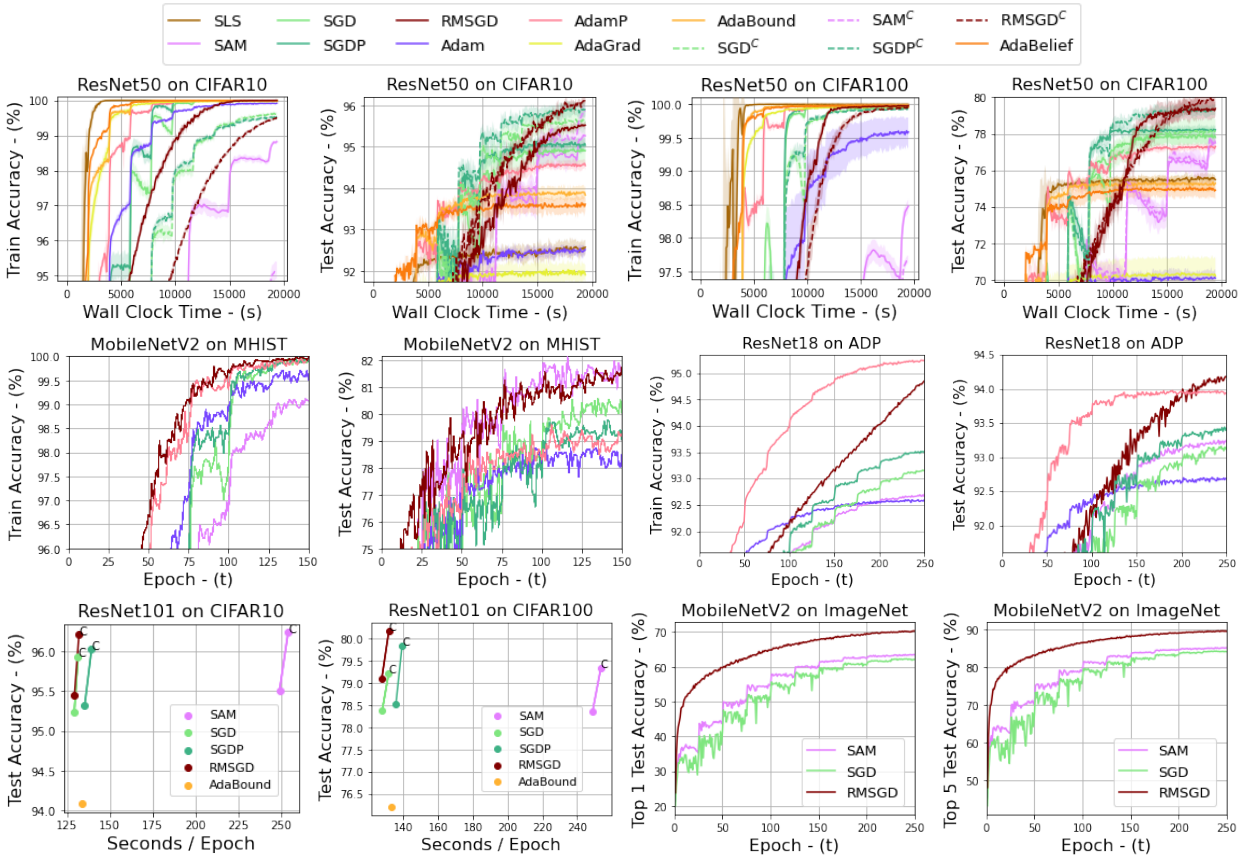


Figure 4. Test/train performance on different selections of dataset, network and optimizer. The cutout augmentation is annotated with a C. A single GPU is used for all experiments including for ImageNet results with batch size of 128.

complexities. This is in contrast to optimizers like SGDP, who are superior to SGD but incur computational cost and slightly lower performance than RMSGD. SAM has also shown to result in performance improvements over SGD, however at a significant cost computationally (double that of SGD). We further highlight that SAM tends to also lose performance in scenarios where the batch size is small for ImageNet training. RMSGD is able to remain very performant in low computational environments. A comparative study is done in Appendix-B for epoch time analysis on various setup of network and dataset using different optimizers.

Learning rate as regularization. We emphasize RMSGD's robustness to varying batch sizes, and particularly its high performance levels on low batch sizes, as compared to SGD and SAM who lose performance (e.g. on ADP – Table 4, ImageNet – Table 2). We hypothesize that RMSGD's ability to dynamically adjust its learning rate per layer regularizes training in a manner that larger batch sizes normally do [16, 18, 24, 43]. With larger batch sizes, model gradient updates in SGD yield low variance and using a higher learning rate is possible in order to achieve better performance. However, such scaling is not consistent across optimizer selection. We found that scaling the batch size with SAM can in fact degrades performance. This is similar to what has been reported in [11, 53, 54] where training stability when using larger batch sizes varies for each layer in a network. This finding led to the development of a new optimizer that used layer-wise learning rate scheduling and demonstrated improved performance in large batch size settings. The concept of RMSGD's per-layer learning rate adjustment, enabled by tracking the per-layer stable rank, also permits such a stablilization behaviour; but works well even when using small batch sizes for training. We argue that this explains why its performance levels remain consistently high in different batch size setups.

Societal impact. Recent works [13, 44] have highlighted the risks of AI on climate change, and being computationally efficient to reduce carbon footprint is important. We showed that RMSGD is able to run on minimal hardware and achieve superior results, contributing towards this idea.

References

[1] Atilim Gunes Baydin, Robert Cornish, David Martinez Rubio, Mark Schmidt, and Frank Wood. Online learning rate adapta-

tion with hypergradient descent. In *International Conference on Learning Representations*, 2018. 2

[2] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010. 2, 5

[3] Léon Bottou. Stochastic gradient descent tricks. In *Neural networks: Tricks of the trade*, pages 421–436. Springer, 2012. 2, 5

[4] Michael B Cohen, Jelani Nelson, and David P Woodruff. Optimal approximate matrix product in terms of stable rank. In *43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016. 3

[5] Ciprian A Corneanu, Sergio Escalera, and Aleix M Martinez. Computing the testing error without a testing set. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2677–2685, 2020. 2

[6] Terrance DeVries and Graham W Taylor. Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*, 2017. 6

[7] Xuanyi Dong, Lu Liu, Katarzyna Musial, and Bogdan Gabrys. Nats-bench: Benchmarking nas algorithms for architecture topology and size. *IEEE transactions on pattern analysis and machine intelligence*, 2021. 4

[8] Timothy Dozat. Incorporating nesterov momentum into adam. In *International Conference on Learning Representations*, 2016. 2

[9] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(61):2121–2159, 2011. 7

[10] Gintare Karolina Dziugaite, Alexandre Drouin, Brady Neal, Nitarshan Rajkumar, Ethan Caballero, Linbo Wang, Ioannis Mitliagkas, and Daniel M Roy. In search of robust measures of generalization. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 11723–11733. Curran Associates, Inc., 2020. 1, 2

[11] Yong Liu et al. Sharpness-aware minimization in large-batch training: Training vision transformer in minutes, 2022. 8

[12] Pierre Foret, Ariel Kleiner, Hossein Mobahi, and Behnam Neyshabur. Sharpness-aware minimization for efficiently improving generalization. In *International Conference on Learning Representations*, 2021. 1, 2, 4, 7

[13] Andre Fu, Mahdi S Hosseini, and Konstantinos N Plataniotis. Reconsidering co2 emissions from computer vision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2311–2317, 2021. 8

[14] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016. 1, 7

[15] 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*, pages 770–778, 2016. 7

[16] Tong He, Zhi Zhang, Hang Zhang, Zhongyue Zhang, Junyuan Xie, and Mu Li. Bag of tricks for image classification with convolutional neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 558–567, 2019. 1, 5, 8

[17] Byeongho Heo, Sanghyuk Chun, Seong Joon Oh, Dongyoon Han, Sangdoo Yun, Gyuwan Kim, Youngjung Uh, and Jung-Woo Ha. Adamp: Slowing down the slowdown for momentum optimizers on scale-invariant weights. In *International Conference on Learning Representations*, pages 3–7, 2021. 1, 2, 4, 7

[18] 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. 8

[19] Roger A Horn and Charles R Johnson. *Matrix analysis*. Cambridge university press, 2012. 3

[20] Mahdi S Hosseini, Lyndon Chan, Gabriel Tse, Michael Tang, Jun Deng, Sajad Norouzi, Corwyn Rowsell, Konstantinos N Plataniotis, and Savvas Damaskinos. Atlas of digital pathology: A generalized hierarchical histological tissue type-annotated database for deep learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11747–11756, 2019. 6

[21] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7132–7141, 2018. 7

[22] Yiding Jiang, Dilip Krishnan, Hossein Mobahi, and Samy Bengio. Predicting the generalization gap in deep networks with margin distributions. In *International Conference on Learning Representations*, 2019. 1, 4

[23] Yiding Jiang*, Behnam Neyshabur*, Hossein Mobahi, Dilip Krishnan, and Samy Bengio. Fantastic generalization measures and where to find them. In *International Conference on Learning Representations*, 2020. 1, 2

[24] Nitish Shirish Keskar, Jorge Nocedal, Ping Tak Peter Tang, Dheevatsa Mudigere, and Mikhail Smelyanskiy. On large-batch training for deep learning: Generalization gap and sharp minima. In *International Conference on Learning Representations*, 2017. 8

[25] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015. 2, 7

[26] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. *N/A*, 2009. 6

[27] Vadim Lebedev, Yaroslav Ganin, Maksim Rakhuba, Ivan V Oseledets, and Victor S Lempitsky. Speeding-up convolutional neural networks using fine-tuned cp-decomposition. In *International Conference on Learning Representations*, 2015. 2

[28] Yoonho Lee, Juho Lee, Sung Ju Hwang, Eunho Yang, and Seungjin Choi. Neural complexity measures. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 9713–9724. Curran Associates, Inc., 2020. 2

[29] Jinlong Liu, Yunzhi Bai, Guoqing Jiang, Ting Chen, and Huayan Wang. Understanding why neural networks generalize well through gsnr of parameters. In *International Conference on Learning Representations*, 2020. 1, 4

[30] Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. On the variance

of the adaptive learning rate and beyond. In *International Conference on Learning Representations*, 2020. 2

[31] Nicolas Loizou and Peter Richtárik. Momentum and stochastic momentum for stochastic gradient, newton, proximal point and subspace descent methods. *Computational Optimization and Applications*, pages 1–58, 2020. 2, 5, 6

[32] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. In *International Conference on Learning Representations*, 2017. 1

[33] Liangchen Luo, Yuanhao Xiong, and Yan Liu. Adaptive gradient methods with dynamic bound of learning rate. In *International Conference on Learning Representations*, 2019. 1, 2, 7

[34] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *Proceedings of the European conference on computer vision (ECCV)*, pages 116–131, 2018. 7

[35] Shinichi Nakajima, Masashi Sugiyama, S Derin Babacan, and Ryota Tomioka. Global analytic solution of fully-observed variational bayesian matrix factorization. *Journal of Machine Learning Research*, 14(Jan):1–37, 2013. 2, 6

[36] Behnam Neyshabur, Srinadh Bhojanapalli, David Mcallester, and Nati Srebro. Exploring generalization in deep learning. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. 1, 4

[37] Boris Polyak. Some methods of speeding up the convergence of iteration methods. *Ussr Computational Mathematics and Mathematical Physics*, 4:1–17, 12 1964. 2, 5

[38] Mark Rudelson and Roman Vershynin. Sampling from large matrices: An approach through geometric functional analysis. *Journal of the ACM (JACM)*, 54(4):21–es, 2007. 3

[39] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. 6

[40] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018. 7

[41] Leslie N Smith. Cyclical learning rates for training neural networks. In *IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 464–472, 2017. 1, 5

[42] Leslie N Smith and Nicholay Topin. Super-convergence: Very fast training of neural networks using large learning rates. In *Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications*, volume 11006, page 1100612. International Society for Optics and Photonics, 2019. 1, 5

[43] Samuel L. Smith, Pieter-Jan Kindermans, and Quoc V. Le. Don't decay the learning rate, increase the batch size. In *International Conference on Learning Representations*, 2018. 8

[44] Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for deep learning in nlp. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3645–3650, 2019. 8

[45] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147, 2013. 2

[46] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*, pages 6105–6114. PMLR, 2019. 7

[47] Thomas Unterthiner, Daniel Keysers, Sylvain Gelly, Olivier Bousquet, and Ilya O. Tolstikhin. Predicting neural network accuracy from weights. *CoRR*, abs/2002.11448, 2020. 2

[48] Sharan Vaswani, Aaron Mishkin, Issam Laradji, Mark Schmidt, Gauthier Gidel, and Simon Lacoste-Julien. Painless stochastic gradient: Interpolation, line-search, and convergence rates. In *Advances in Neural Information Processing Systems*, pages 3727–3740, 2019. 2, 7

[49] Jerry Wei, Arief Suriawinata, Bing Ren, Xiaoying Liu, Mikhail Lisovsky, Louis Vaickus, Charles Brown, Michael Baker, Naofumi Tomita, Lorenzo Torresani, et al. A petri dish for histopathology image analysis. In *International Conference on Artificial Intelligence in Medicine*, pages 11–24. Springer, 2021. 6

[50] Ashia C Wilson, Rebecca Roelofs, Mitchell Stern, Nati Srebro, and Benjamin Recht. The marginal value of adaptive gradient methods in machine learning. In *Advances in Neural Information Processing Systems*, pages 4148–4158, 2017. 2, 4

[51] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1492–1500, 2017. 7

[52] Scott Yak, Javier Gonzalvo, and Hanna Mazzawi. Towards task and architecture-independent generalization gap predictors. *arXiv preprint arXiv:1906.01550*, 2019. 2

[53] Yang You, Igor Gitman, and Boris Ginsburg. Scaling sgd batch size to 32k for imagenet training. *arXiv preprint arXiv:1708.03888*, 6:12, 2017. 8

[54] 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 *International Conference on Learning Representations*, 2020. 8

[55] Kun Yuan, Bicheng Ying, and Ali H Sayed. On the influence of momentum acceleration on online learning. *The Journal of Machine Learning Research*, 17(1):6602–6667, 2016. 2, 5

[56] Bai-cun Zhou, Cong-ying Han, et al. Convergence of stochastic gradient descent in deep neural network. *Acta Mathematicae Applicatae Sinica, English Series*, 37(1):126–136, 2021. 6

[57] Juntang Zhuang, Tommy Tang, Yifan Ding, Sekhar C Tatikonda, Nicha Dvornek, Xenophon Papademetris, and James Duncan. Adabelief optimizer: Adapting stepsizes

by the belief in observed gradients. *Advances in neural information processing systems*, 33:18795–18806, 2020. 1, 7