CENTER FOR
Brains
Minds+
Machines

CBMM Memo No. 140 **February 14, 2023**

SGD and Weight Decay Provably Induce a Low-Rank Bias in Deep Neural Networks

Tomer Galanti[1], **Zachary Siegel**[2], **Aparna Gupte**[1] **and Tomaso Poggio**[1]

1: Center for Brains, Minds, and Machines, MIT, Cambridge, MA, USA
2: Department of Computer Science, Princeton University

Abstract

In this paper, we study the bias of Stochastic Gradient Descent (SGD) to learn low-rank weight matrices when training deep ReLU neural networks. Our results show that training neural networks with mini-batch SGD and weight decay causes a bias towards rank minimization over the weight matrices. Specifically, we show, both theoretically and empirically, that this bias is more pronounced when using smaller batch sizes, higher learning rates, or increased weight decay. Additionally, we predict and observe empirically that weight decay is necessary to achieve this bias. Finally, we empirically investigate the connection between this bias and generalization, finding that it has a marginal effect on generalization. Our analysis is based on a minimal set of assumptions and applies to neural networks of any width or depth, including those with residual connections and convolutional layers.

This material is based upon work supported by the Center for Brains, Minds and Machines (CBMM), funded by NSF STC award CCF-1231216.

SGD and Weight Decay Provably Induce a Low-Rank Bias in Deep Neural Networks

Tomer Galanti [1] **Zachary S. Siegel** [2] **Aparna Gupte** [1] **Tomaso Poggio** [1]

Abstract

In this paper, we study the bias of Stochastic Gradient Descent (SGD) to learn low-rank weight matrices when training deep ReLU neural networks. Our results show that training neural networks with mini-batch SGD and weight decay causes a bias towards rank minimization over the weight matrices. Specifically, we show, both theoretically and empirically, that this bias is more pronounced when using smaller batch sizes, higher learning rates, or increased weight decay. Additionally, we predict and observe empirically that weight decay is necessary to achieve this bias. Finally, we empirically investigate the connection between this bias and generalization, finding that it has a marginal effect on generalization. Our analysis is based on a minimal set of assumptions and applies to neural networks of any width or depth, including those with residual connections and convolutional layers.

1. Introduction

Stochastic gradient descent (SGD) is a widely used optimization technique for training deep learning models (Bottou, 1991). While it was initially developed to address the computational challenges of gradient descent, recent studies suggest that SGD also provides regularization that prevents overparameterized models from converging to minima that do not generalize well (Zhang et al., 2016; Jastrzebski et al., 2017; Keskar et al., 2017; Zhu et al., 2019). For instance, empirical studies have shown that SGD outperforms gradient descent (Zhu et al., 2019) and that smaller batch sizes result in better generalization (Hoffer et al., 2017; Keskar et al., 2017). However, the full range of regularization effects induced by SGD is not yet fully understood.

One area of recent research focuses on characterizing the implicit regularization of gradient-based optimization and its relationship to generalization in deep learning. Several papers have examined the potential bias of gradient descent or stochastic gradient descent toward rank minimization. Empirically, it was shown (Denton et al., 2014; Alvarez & Salzmann, 2017; Tukan et al., 2021; Yu et al., 2017; Arora et al., 2018) that replacing weight matrices with low-rank approximations results in only a small drop in accuracy, suggesting that the weight matrices at convergence may be close to low-rank matrices. Following this line of work, various attempts were made to understand the origins of this low-rank bias and its potential relation with generalization.

Initially, it was believed that the implicit regularization in matrix factorization could be characterized in terms of the nuclear norm of the corresponding linear predictor (Gunasekar et al., 2017). This conjecture was later refuted (Li et al., 2020). Subsequent conjecture posits that rank minimization may play a key role in explaining generalization in deep learning. For instance, (Razin & Cohen, 2020) conjectured that the implicit regularization in matrix factorization can be explained by rank minimization, and also hypothesized that some notion of rank minimization may be key to explaining generalization in deep learning. Additionally, (Li et al., 2020) established evidence that the implicit regularization in matrix factorization is a heuristic for rank minimization. Beyond factorization problems, (Ji & Telgarsky, 2020) showed that gradient flow (GF) training of univariate linear networks with respect to exponentially-tailed classification losses learns weight matrices of rank 1. Intuitively, such networks generalize well due to their effectively limited capacities.

With nonlinear neural networks, the origin of this bias and its connection with generalization is less clear. Several papers (Ongie & Willett, 2022; Le & Jegelka, 2022) studied low-rank bias in linear layers at the top of a neural network. For instance, (Le & Jegelka, 2022) analyzes low-rank bias in neural networks trained with gradient flow (GF) without regularization. While this paper makes significant strides in extending the analysis in (Ji & Telgarsky, 2020), it makes several limiting assumptions. As a result, their analysis is only applicable under very specific conditions, such as when the data is linearly separable, and their low-rank analysis

*Equal contribution [1]Massachusetts Institute of Technology [2]Princeton University. Correspondence to: Tomer Galanti <galanti@mit.edu>.

Preprint

is limited to a set of linear layers aggregated at the top of the trained network. Later, (Timor et al., 2022) showed that for ReLU networks, GF generally does not minimize rank. They also argued that sufficiently deep ReLU networks exhibits low-rank solutions under L_2 norm minimization. This interesting result, however, applies only to the global minima and only to layers added to a pre-existing network that is capable of solving the problem.

Despite the recent progress in understanding the low-rank bias in deep networks, a complete understanding of its origins and its relationship with different hyperparameters is largely missing.

Contributions. In this paper, we show that using mini-batch stochastic gradient descent (SGD) and weight decay effectively regularize the rank of the learned weight matrices during the training of neural networks. Our theoretical analysis predicts that smaller batch sizes, higher learning rates, or increased weight decay results in a decrease in the rank of the learned matrices, and that regularization is necessary to achieve this bias. The scope of the analysis is fairly general, covering deep ReLU networks trained with mini-batch SGD for minimizing a differentiable loss function with L_2 regularization (i.e., weight decay). The neural networks may include fully-connected layers, residual connections, and convolutional layers.

In addition to our theoretical analysis, we provide a comprehensive empirical study in which we examine the effects of different hyperparameters on the rank of weight matrices for various network architectures. Additionally, we carried out several experiments to examine the connection between low-rank bias and generalization. The results indicate that while low-rank bias is not a requirement for good generalization, it is correlated with a marginal improvement in performance.

2. Problem Setup

In this paper, we study the influence of using mini-batch stochastic gradient descent (SGD) in conjunction with weight decay on the inductive biases of neural networks in standard supervised learning settings. The task at hand is defined by a distribution P over samples $(x, y) \in \mathcal{X} \times \mathcal{Y}$, where $\mathcal{X} \subset \mathbb{R}^{c_1 \times h_1 \times w_1}$ is the space of instances (e.g., images), and $\mathcal{Y} \subset \mathbb{R}^k$ is the label space.

We consider a parametric model $\mathcal{F} \subset \{f' : \mathcal{X} \to \mathbb{R}^k\}$, where each function $f_W \in \mathcal{F}$ is specified by a vector of parameters $W \in \mathbb{R}^N$. The function $f_W \in \mathcal{F}$ assigns a prediction to any input point $x \in \mathcal{X}$, and its performance is measured by the Expected Risk,

$$L_P(f_W) \;:=\; \mathbb{E}_{(x,y)\sim P}[\ell(f_W(x), y)],$$

where $\ell : \mathbb{R}^k \times \mathcal{Y} \to [0, \infty)$ is a non-negative, differentiable, loss function (e.g., MSE or cross-entropy losses). For simplicity, in the analysis we assume that $k = 1$.

Since we do not have direct access to the full population distribution P, the goal is to learn a predictor, f_W, from a training dataset $S = \{(x_i, y_i)\}_{i=1}^m$ of independent and identically distributed (i.i.d.) samples drawn from P. To avoid overfitting the training data, we typically use weight decay to control the complexity of the learned model. Specifically, we aim to minimize the Regularized Empirical Risk,

$$L_S^\lambda(f_W) \;:=\; \frac{1}{m}\sum_{i=1}^{m} \ell(f_W(x_i), y_i) + \lambda \|W\|_2^2,$$

where $\lambda > 0$ is a predefined hyperparameter and $\|\cdot\|_2$ is the Frobenius norm. To accomplish this task, we typically use mini-batch SGD, as outlined in the following paragraph.

Optimization. In this study, we employ stochastic gradient descent (SGD) to minimize the regularized empirical risk $L_S^\lambda(f_W)$ over a specified number of iterations T. We begin by initializing W_1 using a standard initialization method, and then update W_t for T iterations, ultimately returning W_T. At each iteration t, we randomly select a batch $\tilde{S}_t \subset S$ of B samples, and update $W_{t+1} = W_t - \mu \nabla_W L_{\tilde{S}_t}^\lambda(fW_t)$, where $\mu > 0$ is the predefined learning rate.

Notation. In this paper, we use the following notations. For an integer $k \geq 1$, we use the notation $[k] = \{1, \ldots, k\}$. The Euclidean norm of a vector $z \in \mathbb{R}^n$ is denoted by $\|z\| := \sqrt{\sum_{i=1}^n z_i^2}$. For two vectors $x \in \mathbb{R}^n, y \in \mathbb{R}^m$ we define their concatenation as follows $(x\|y) := (x_1, \ldots, x_n, y_1, \ldots, y_m) \in \mathbb{R}^{n+m}$. For a given matrix $A \in \mathbb{R}^{n\times m}$, we denote $A_{i,:}$ its ith row and by $\text{vec}(A) := (A_1\| \ldots \|A_n)$ its vectorization. For a given tensor $A \in \mathbb{R}^{c\times h\times w}$, we denote by $\text{vec}(A) := (\text{vec}(A_1)\| \ldots \|\text{vec}(A_c))$ the vectorized form of A. Tensor slicing is defined as, $x_{a:b} := (x_a, \ldots, x_b)$.

2.1. Architectures

In this work, the function f_W represents a neural network, consisting of a set of layers of weights interlaced with ReLU activation units. Our definition of a neural network is fairly generic, including convolutional layers, pooling layers, residual connections, and fully-connected layers.

Network architecture. Formally, a neural network f_W can be described as a directed acyclic graph (DAG) $G = (V, E)$, where $V = \{v_1, \ldots, v_L\}$ consists of the various layers of the network, and each edge $e_{ij} = (v_i, v_j) \in E$ represents a connection between two layers. Each layer is a function $v_i : \mathbb{R}^{c_1\times h_1\times w_1} \to \mathbb{R}^{c_i\times h_i\times w_i}$, and each connection (v_i, v_j) holds a transformation $C^{ij} : \mathbb{R}^{c_j\times h_j\times w_j} \to \mathbb{R}^{c_i\times h_i\times w_i}$. The layers are divided into three categories: the input layer v_1, the output layer v_L, and intermediate layers. There are no connections directed towards the

input layer or out of the output layer (i.e., $\forall i \in [L] : (v_i, v_L), (v_0, v_i) \notin E$). Given an input $x \in \mathbb{R}^{c_i \times h_i \times w_i}$, the output of a given layer v_i is evaluated as follows $v_i(x) := \sigma(\sum_{j \in \text{pred}(i)} C^{ij}(v_j(x)))$, except for the output layer v_L that computes $f_W(x) := v_L(x) := \sum_{j \in \text{pred}(L)} C^{Lj}(v_j(x))$. Here, $\text{pred}(i) := \{j \in [L] \mid (v_i, v_j) \in E\}$, $\text{succ}(i) := \{j \in [L] \mid (v_j, v_i) \in E\}$ denote the sets of predecessor and successor layers of the ith layer and σ is the elementwise ReLU activation function. Each transformation C^{ij} is either trainable (e.g., a convolutional layer) or a constant affine transformation (e.g., a residual connection). The set of trainable connections is denoted by E_T. In this paper, we consider the following types of layers.

Convolutional layers. A convolutional layer (Lecun et al., 1998) (see also (Goodfellow et al., 2016)), commonly used in image processing tasks, is defined by a kernel tensor $Z^{ij} \in \mathbb{R}^{c_i \times c_j \times k_1 \times k_2}$, where c_j, c_i, k_1, and k_2 represent the number of input and output channels and the kernel sizes respectively. The layer applies the kernel tensor to the input tensor by sliding it across the input tensor with a specified stride length, s, after zero-padding the input tensor with p rows in each "side" of the tensor. The output tensor $y \in \mathbb{R}^{c_i \times h_i \times w_i}$ is computed by summing up the elementwise product of the kernel tensor and the corresponding section of the padded input tensor at each position of the sliding. Formally, for all indices $(c, t, l) \in [c_i] \times [h_i] \times [w_i]$,

$$y_{c,t,l} = \sum_{c'=1}^{c_j} \text{vec}(Z_{c,c',:})^\top \cdot \text{vec}(\text{Pad}_p(x)_{c',ts:(t+1)s+k_1,ls:(l+1)s+k_2}).$$

Here, Pad_p takes a tensor $x \in \mathbb{R}^{c_j \times h_j \times w_j}$ and returns a new tensor $x' \in \mathbb{R}^{c_j \times (h_j+2p) \times (w_j+2p)}$, where the first and last p rows and columns of each channel $x'_{c,:,:}$ are zeros and the middle $1 \times h_j \times w_j$ tensor is equal to $x_{c,:,:}$. The output dimensions h_i and w_i are calculated using the formulas $h_i = (\lfloor (h_j - k_j + 2p) \rfloor / s + 1)$ and $w_i = (\lfloor (w_j - k_2 + 2p) \rfloor / s + 1)$.

We can also represent the convolutional layer as a linear operation by defining a matrix $V^{ij} \in \mathbb{R}^{c_i h_i w_i \times c_j h_j w_j}$, which computes the output of the layer for a given input vectorized as a column vector, and a matrix $W^{ij} \in \mathbb{R}^{c_i \times c_j k_1 k_2}$, which has the vectorized filters as its rows, $W^{ij}_c := \text{vec}(Z^{ij}_{c,:,:})$. This allows us to express the convolutional layer as a linear operator, making it possible to analyze its properties mathematically.

Fully-connected layers. As a special case of convolutional layers, the network may also include fully-connected layers. A fully-connected layer $F : \mathbb{R}^{c_j} \to \mathbb{R}^{c_i}$, associated with a matrix $W \in \mathbb{R}^{c_i \times c_j}$, can be represented as a 1×1 convolutional layer $C : \mathbb{R}^{c_j \times 1 \times 1} \to \mathbb{R}^{c_i \times 1 \times 1}$ with $k_1 = k_2 = 1$, $p = 0$ and $s = 1$. The parameters tensor $Z \in \mathbb{R}^{c_i \times c_j \times 1 \times 1}$ satisfies $Z_{a,b,1,1} = W_{a,b}$ for all $(a, b) \in [c_i] \times [c_j]$, and the layer satisfies $\text{vec}(C(x)) = W\text{vec}(x)$.

Pooling layers. A pooling layer (Zhou & Chellappa, 1988) (see also (Goodfellow et al., 2016)) C with kernel dimensions (k_1, k_2) stride s and padding p takes an input $x \in \mathbb{R}^{c_j \times h_j \times w_j}$ and computes an output $y \in \mathbb{R}^{c_i \times h_i \times w_i}$ with $c_i = c_j$ channels, and dimensions $h_i = (\lfloor (h_j - k_1 + 2p) \rfloor / s + 1)$ and $w_i = (\lfloor (w_j - k_2 + 2p) \rfloor / s + 1)$. The output of each pooling layer is computed as follows:

$$y_{c,t,l} = \text{op}(\text{Pad}_p(x)_{c,ts:(t+1)s+k_1,ls:(l+1)s+k_2}),$$

where op is either the maximum or average operator and $(c, t, l) \in [c_i] \times [h_i] \times [w_i]$.

Rearrangement layers. To easily switch between convolutional and fully-connected layers, we should be able to represent tensor layers as vectors and vice versa. To change the shape of a specific layer, we use rearrangement layers. A rearrangement layer $C^{ij} : \mathbb{R}^{c_j \times h_j \times w_j} \to \mathbb{R}^{c_i \times h_i \times w_i}$ takes an input vector $x \in \mathbb{R}^{c_j \times h_j \times w_j}$ and rearranges its coordinates in a different shape and order. Formally, it returns a vector $(x_{\pi(k)})_{k \in [c_j] \times [h_j] \times [w_j]}$, where $\pi : [c_j] \times [h_j] \times [w_j] \to [c_i] \times [h_i] \times [w_i]$ is invertible (in particular $c_i h_i w_i = c_j h_j w_j$).

3. Theoretical Results

In this section, we present our main theoretical result. We show that when training a ReLU neural network with SGD, the weight matrices tend to be close to matrices of a bounded rank. Specifically, with a simple observation (proved in Appendix A) that the number of input patches N^{ij} of a certain convolutional layer C^{ij} sets an upper bound on the rank of the gradient of the network with respect to the parameters matrix W^{ij}. By recursively unrolling the optimization, we express the weight matrix W^{ij}_t as a sum of $(1 - \mu\lambda)^k W^{ij}_{t-k}$ and kB gradients of the loss function with respect to W^{ij} for different samples at different iterations. Since each one of these terms is a matrix of rank $\leq N^{ij}$, we conclude that the distance between W^{ij}_t and a matrix of rank $\leq N^{ij}Bk$ decays exponentially with increasing k.

Lemma 3.1. *Let f_W be a neural network and let C^{ij} be a convolutional layer within f_W with parameters matrix W^{ij}. Then,* $\text{rank}\left(\nabla_{W^{ij}} f_W(x)\right) \leq N^{ij}$.

Interestingly, we observe particularly degenerate gradients for fully-connected layers. As discussed in Sec. 2.1, for a fully-connected layer $C^{ij} : \mathbb{R}^{c_j \times 1 \times 1} \to \mathbb{R}^{c_i \times 1 \times 1}$ we have $N^{ij} = 1$, and thus, $\text{rank}\left(\nabla_{W^{ij}} f_W(x)\right) \leq 1$. To demonstrate this observation, we provide a simple proof for the case of fully-connected networks.

Lemma 3.2. *Let $f_W(x) = W^L \sigma(W^{L-1} \cdots \sigma(W_1 x) \cdots)$ be a neural network, where $W^l \in \mathbb{R}^{d_{l+1} \times d_l}$ for all $l \in [L]$ and σ is the elementwise ReLU activation. Then,* $\text{rank}\left(\nabla_{W^l} f_W(x)\right) \leq 1$.

Proof. We would like to show that the matrix $\text{rank}(\nabla_{W^l} f_W(x)) \leq 1$. We note that for any given vector $z \in \mathbb{R}^d$, we have $\sigma(z) = \text{diag}(\sigma'(z)) \cdot z$. Therefore, for any input vector $x \in \mathbb{R}^{d_1}$, the output of f_W can be written as follows,

$$\begin{aligned} f_W(x) &= W^L \sigma(W^{L-1} \cdots \sigma(W^1 x) \cdots) \\ &= W^L \cdot D_{L-1}(x;W) W^{L-1} \cdots D_1(x;W) \cdot W^1 \cdot x, \end{aligned}$$

where $D_l(x;W) = \text{diag}[\sigma'(u_l(x;W)))]$ and $u_l(x;W) = W^l \sigma(W^{l-1} \cdots \sigma(W^1 x) \dots)$. We denote by $u_{l,i}(x;W)$ the i'th coordinate of the vector $u_l(x;W)$. We note that $u_l(x;W)$ are continuous functions of W. Therefore, assuming that none of the coordinates $u_{l,i}(x;W)$ are zero, there exists a sufficiently small ball around W for which $u_{l,i}(x;W)$ does not change its sign. Hence, within this ball, $\sigma'(u_{l,i}(x;W))$ are constant. We define a set $\mathcal{W}_{l,i} = \{W \mid u_{l,i}(x;W) = 0\}$. We note that as long as $x \neq 0$, the set $\mathcal{W}_{l,i}$ is negligible within $\mathbb{R}^N$. Since there is a finite set of indices l, i, the set $\bigcup_{l,i} \mathcal{W}_{l,i}$ is also negligible.

Let W be a set of parameters for which all of the coordinates $u_{l,i}(x;W)$ are non-zero. Then, the matrices $\{D_l(x;W)\}_{l=1}^{L-1}$ are constant in the neighborhood of W, and therefore, their derivative with respect to W^l are zero. Let $a^\top = W^L \cdot D_{L-1}(x;W) W^{L-1} \cdots W^{l+1} D_l(x;W)$ and $b = D_{l-1}(x) \cdot W^{l-1} \cdots W^1 x$. We can write $f_W(x) = a(x;W)^\top \cdot W^l \cdot b(x;W)$. Since the derivatives of $a(x;W)$ and $b(x;W)$ with respect to W^l are zero, by applying the formula $\frac{\partial a^\top X b}{\partial X} = ab^\top$, we have $\nabla_{W^l} f_W(x) = a(x;W) \cdot b(x;W)^\top$ which is a matrix of rank at most 1. Therefore, for any input $x \neq 0$, with measure 1 (over the selection of W), $\nabla_{W^l} f_W(x)$ is a matrix of rank at most 1. □

The following theorem provides an upper bound on the minimal distance between the network's weight matrices and low-rank matrices.

Theorem 3.3. *Let $\|\cdot\|$ be any matrix norm and ℓ any differentiable loss function. Let $f_W(x)$ be a ReLU neural network and C^{ij} be a convolutional layer within f_W and let $B \in [m]$. Then, for all $k < t$,*

$$\min_{W:\ \text{rank}(W) \leq N^{ij} Bk} \left\| \frac{W_t^{ij}}{\|W_t^{ij}\|} - W \right\| \ \leq\ (1-2\mu\lambda)^k \cdot \frac{\|W_{t-k}^{ij}\|}{\|W_t^{ij}\|}.$$

Proof. We denote by $\tilde{S}_t \subset S$ the training batch that was used by SGD at iteration t. We have

$$\begin{aligned} W_t^{ij} &= W_{t-1}^{ij} - \mu \nabla_{W^{ij}} L_{\tilde{S}_{t-1}}(f_{W_{t-1}}) - 2\mu\lambda W_{t-1}^{ij} \\ &= (1-2\mu\lambda) W_{t-1}^{ij} - \mu \nabla_{W^{ij}} L_{\tilde{S}_{t-1}}(f_{W_{t-1}}). \end{aligned}$$

Similarly, we can write

$$W_{t-1}^{ij} = (1-2\mu\lambda) W_{t-2}^{ij} - \mu \nabla_{W^{ij}} L_{\tilde{S}_{t-2}}(f_{W_{t-2}}).$$

This gives us

$$\begin{aligned} W_t^{ij} &= (1-2\mu\lambda)^2 W_{t-2}^{ij} \\ &\quad - \mu \nabla_{W^{ij}} L_{\tilde{S}_{t-1}}(f_{W_{t-1}}) \\ &\quad - \mu(1-2\mu\lambda) \nabla_{W^{ij}} L_{\tilde{S}_{t-2}}(f_{W_{t-2}}). \end{aligned}$$

By recursively applying this process k times, we have

$$\begin{aligned} W_t^{ij} &= (1-2\mu\lambda)^k W_{t-k}^{ij} \\ &\quad \underbrace{-\mu \sum_{l=1}^{k} (1-2\mu\lambda)^{l-1} \nabla_{W^{ij}} L_{\tilde{S}_{t-l}}(f_{W_{t-l}})}_{=:U} \end{aligned}$$

By the chain rule, we can write the gradient of the loss function as follows,

$$\begin{aligned} &\nabla_{W^{ij}} L_{\tilde{S}_{t-l}}(f_{W_{t-l}}) \\ &= \frac{1}{B} \sum_{(x,y) \in \tilde{S}_{t-l}} \frac{\partial \ell(f_{W_{t-l}}(x), y)}{\partial f_{W_{t-l}}(x)} \cdot \nabla_{W^{ij}} f_{W_{t-l}}(x). \end{aligned}$$

According to Lem. 3.1, we have $\text{rank}(\nabla_{W^{ij}} f_{W_{t-l}}(x)) \leq N^{ij}$. Since f_W is a univariate function, each term $\frac{\partial \ell(f_{W_{t-l}}(x), y)}{\partial f_{W_{t-l}}(x)}$ is a scalar. Therefore, $\text{rank}(\nabla_{W^{ij}} L_{\tilde{S}_{t-l}}(f_{W_{t-l}})) \leq BN^{ij}$ since $\nabla_{W^{ij}} L_{\tilde{S}_{t-l}}(f_{W_{t-l}})$ is an average of B matrices of rank at most N^{ij}. In particular, $\text{rank}(U) \leq N^{ij} Bk$ since U is a sum of k matrices of rank at most $N^{ij} B$. Therefore, we obtain that

$$\begin{aligned} &\min_{W:\ \text{rank}(W) \leq N^{ij} Bk} \left\| W_t^{ij} - W \right\| \\ &\leq \left\| W_t^{ij} - U \right\| = (1-2\mu\lambda)^k \|W_{t-k}^{ij}\|. \end{aligned}$$

Finally, by dividing both sides by $\|W_t^{ij}\|$ we obtain the desired inequality. □

The theorem above provides an upper bound on the minimal distance between the parameters matrix W_t^{ij} and a matrix of rank $\leq N^{ij} Bk$. The parameter k controls the looseness of the bound and is independent of the optimization process. The bound is proportional to $(1-2\mu\lambda)^k \frac{\|W_t^{ij}\|}{\|W_{t-k}^{ij}\|}$, which decreases exponentially with k. Assuming the norm of W_t^{ij} converges as t approaches infinity and $k = o(t)$, we see that $\lim_{t\to\infty} \frac{\|W_t^{ij}\|}{\|W_{t-k}^{ij}\|} = 1$. Thus, SGD with weight decay provably induces a low-rank bias in each weight matrix W^{ij}. By selecting $k = \lceil \frac{\log(\epsilon)}{\log(1-2\mu\lambda)} \rceil$, we can ensure that $(1-2\mu\lambda)^k \leq \epsilon$. In this case, at the end of the training, the normalized matrix $\frac{W_t^{ij}}{\|W_t^{ij}\|}$ can be approximated by a second matrix W of rank $\leq \frac{N^{ij} B \log(\epsilon)}{\log(1-2\mu\lambda)} \leq \frac{N^{ij} B \log(1/\epsilon)}{2\mu\lambda}$

(the second inequality assumes that $\mu\lambda < 0.5$) with an error of ϵ. While the value of $\frac{N^{ij}B\log(1/\epsilon)}{2\mu\lambda}$ may be large in small or medium-scale learning settings, it still yields meaningful results for very wide neural networks. For example, since the bound is independent of the input and output channels c_j and c_i of C^{ij}, when c_i and c_j are very large, the dimensions of W^{ij} are much larger than $\frac{N^{ij}B\log(1/\epsilon)}{2\mu\lambda}$, and thus, our bound implies that the use of SGD would provably reduce the rank of W^{ij} during training.

While Thm. 3.3 provides an upper bound of $\frac{N^{ij}B\log(1/\epsilon)}{2\mu\lambda}$ on the rank of the learned matrix, it does not give a precise insight into how various parameters influence the rank. However, based on the bound, we can still make the prediction that **training with smaller batch sizes, increasing weight decay or learning rate will lead to lower rank matrices learned by SGD**. These predictions are empirically validated in the next section.

4. Experiments

In this section, we empirically study how batch size, weight decay, and learning rate affect the rank of matrices in deep ReLU networks. We conduct separate experiments where we vary one hyperparameter while keeping the others constant to isolate its effect on the averaged rank[1]. Additional experiments are provided in Appendix B.

4.1. Setup

Architectures. We evaluate several network architectures in our study. (i) The first architecture is an MLP, denoted as MLP-BN-L-H, which comprises L hidden layers, each containing a fully-connected layer with width H, followed by batch normalization and ReLU activations. This architecture ends with a fully-connected output layer. The same architecture without batch normalization is denoted by MLP-L-H. (ii) The second architecture, referred to as RES-BN-L-H, consists of a linear layer with width H, followed by L residual blocks, and ending with a fully-connected layer. Each block performs a computation of the form $z + \sigma(n_2(W_2\sigma(n_1(W_1 z))))$, where $W_1, W_2 \in \mathbb{R}^{H\times H}$, n_1, n_2 are batch normalization layers, and σ is the ReLU function. (iii) The third architecture is the convolutional network (VGG-16) proposed by (Simonyan & Zisserman, 2014), with dropout replaced by batch normalization layers, and a single fully-connected layer at the end. (iv) The fourth architecture is the residual network (ResNet-18) proposed in (He et al., 2016). (v) The fifth architecture is a small visual transformer (ViT) (Dosovitskiy et al., 2020). Our implementation of ViT splits the input images into patches of size 4×4, includes 8 self-attention heads, each composed of 6 self-attention layers. The self-attention layers are followed by two fully-connected layers with a dropout probability of 0.1, and a GELU activation in between them.

[1]The plots are best viewed when zooming into the pictures.

Training and evaluation. We trained each model for CIFAR10 classification using Cross-Entropy loss minimization between its logits and the one-hot encodings of the labels. The training was carried out by SGD with batch size B, initial learning rate μ, and weight decay λ. The MLP-BN-L-H, RES-BN-L-H, ResNet-18, and VGG-16 models were trained with a decreasing learning rate of 0.1 at epochs 60, 100, and 200, and the training was stopped after 500 epochs. The ViT models were trained using SGD with a learning rate that was decreased by a factor of 0.2 at epochs 60 and 100 and training was stopped after 200 epochs. During training, we applied random cropping, random horizontal flips, and random rotations (by $15k$ degrees for k uniformly sampled from $[24]$) and standardized the data.

To study the influence of different hyperparameters on the rank of the weight matrices, in each experiment, we trained the models while varying one hyperparameter at a time, while keeping other hyperparameters constant. After each epoch, we compute the average rank across the network's weight matrices and its train and test accuracy rates. For a convolutional layer C^{ij}, we use W^{ij} as its weight matrix. To estimate the rank of a given matrix M, we count how many of the singular values of $\frac{M}{\|M\|_2}$ are out of the range $[-\epsilon, \epsilon]$, where ϵ is a small tolerance value.

4.2. Results

Low-rank bias and the batch size. As shown in Figs.3-6, decreasing the batch size strengthens the low-rank constraint on the network's matrices, resulting in matrices of lower ranks. This aligns with the prediction made in Sec. 3 that training with smaller batch sizes leads to matrices of lower ranks. This observation highlights the impact of batch size on the rank of the weight matrices and how it can be used to control the complexity of the network.

Low-rank bias, weight decay and learning rate. As shown in Fig. 2, increasing λ imposes stronger rank minimization constraints on the weight matrices. Interestingly, the effect of batch size on the ranks of the weight matrices appears to be minimal when $\lambda = 0$, which suggests that weight decay is essential for imposing a noticeable low-rank bias on the weight matrices. Furthermore, Figs. 1, 3 and 4 show that increasing the learning rate tends to lead to smaller ranks of weight matrices, which aligns with the prediction made in Sec. 3.

Low-Rank Bias and Generalization. We investigated the relationship between low-rank bias and generalization by training ResNet-18 and VGG-16 models on CIFAR10 with varying batch sizes, while keeping λ and μ constant. To

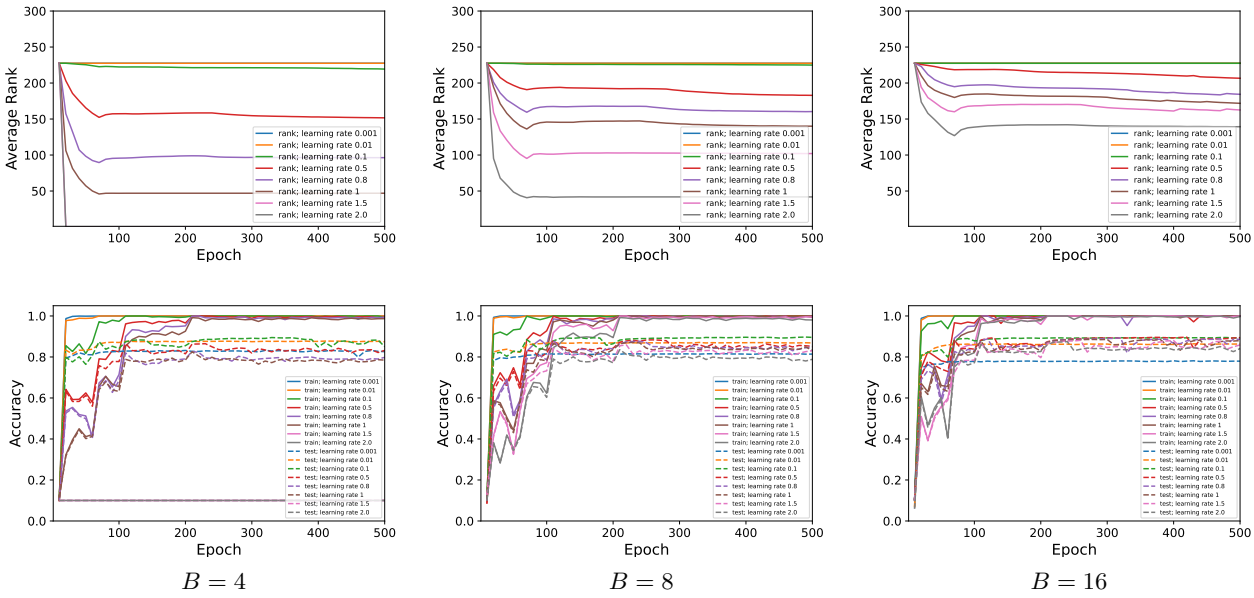


Figure 1. **Average ranks and accuracy rates of ResNet-18 trained on CIFAR10 with varying μ.** The top row shows the average rank across layers, while the bottom row shows the train and test accuracy rates for each setting. In this experiment, $\mu = 5\text{e}{-4}$ and $\epsilon = 1\text{e}{-3}$.

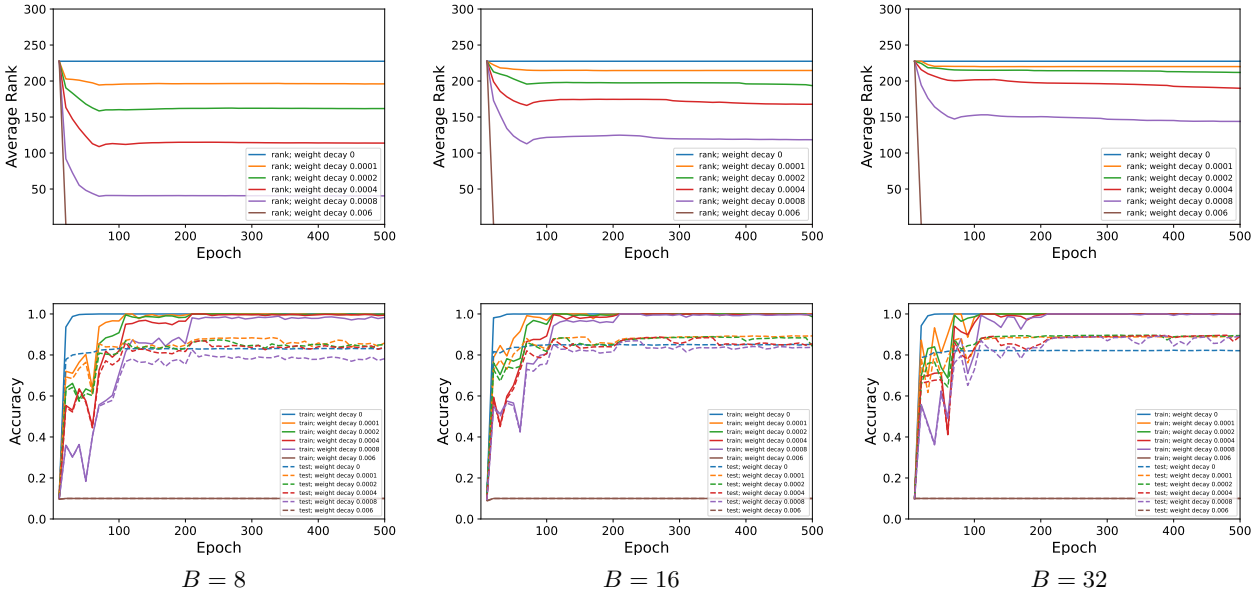


Figure 2. **Average ranks and accuracy rates of ResNet-18 trained on CIFAR10 with different λ values.** In this experiment, $\mu = 1.5$ and $\epsilon = 1\text{e}{-3}$.

provide a fair comparison, we selected λ and μ to ensure all models fit the training data equally. Our results, shown in Figs. 4-6, indicate that models trained with smaller batch sizes (i.e. lower rank in their weights) tend to generalize better as the test accuracy rate monotonically increases as the batch size decreases. Based on these findings, we hypothesize that when altering a certain hyperparameter, a neural network with a lower average rank will have better performance than a network with the same architecture but higher rank matrices, assuming both networks perfectly fit the training data.

5. Conclusions

A mathematical characterization of the biases associated with SGD-trained neural networks is regarded as a signifi-

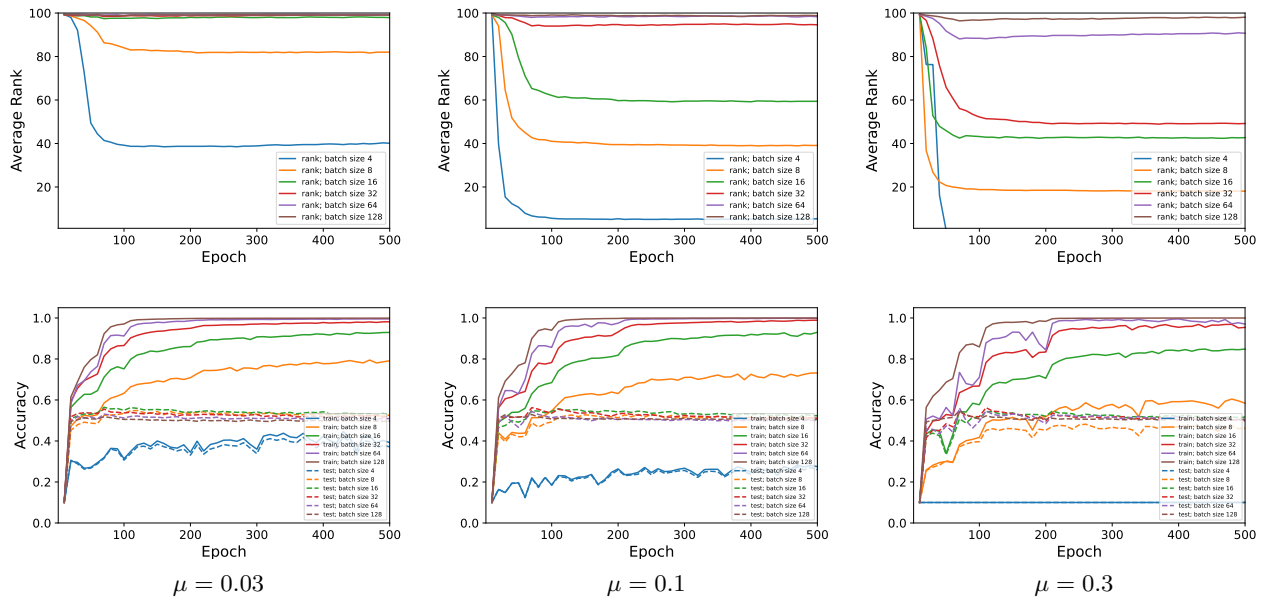


Figure 3. **Average ranks and accuracy rates of MLP-BN-10-100 trained on CIFAR10 with various batch sizes.** In this experiment, $\lambda = 5\text{e}{-4}$ and $\epsilon = 1\text{e}{-3}$.

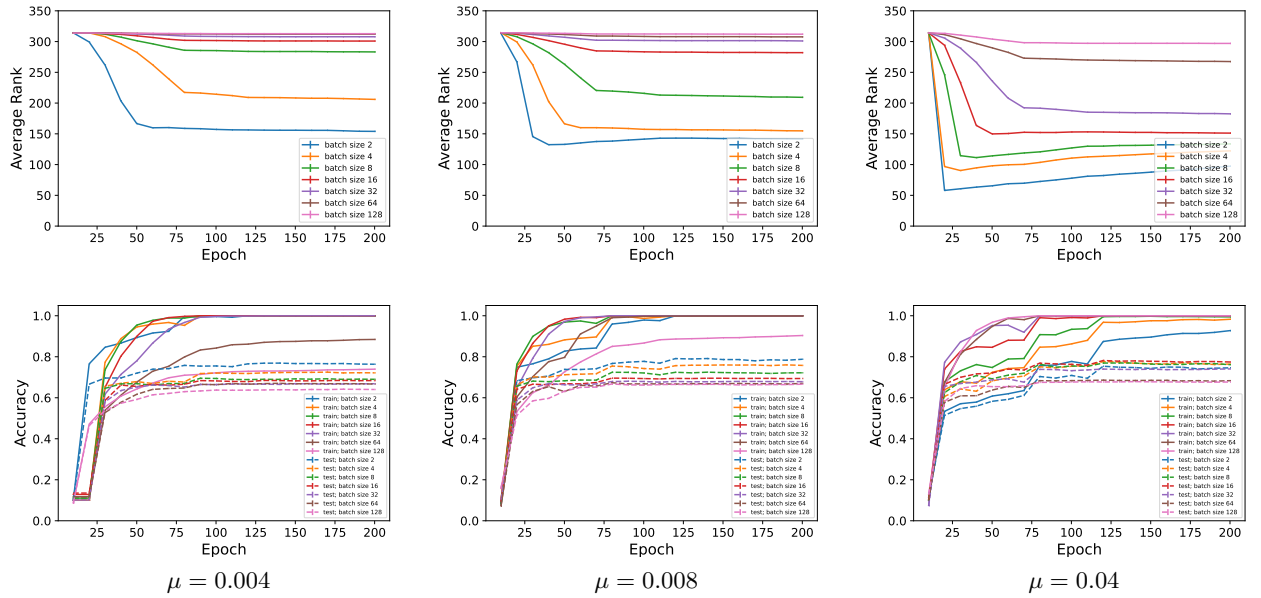


Figure 4. **Average ranks and accuracy rates of ViT trained on CIFAR10 with various batch sizes.** In this experiment, $\lambda = 5\text{e}{-4}$ and $\epsilon = 1\text{e}{-2}$.

cant open problem in the theory of deep learning (Neyshabur et al., 2017). In addition to its independent interest, a low-rank bias – though probably not necessary for generalization – may be a key ingredient in an eventual characterization of the generalization properties of deep networks. In fact, recent results (Huh et al., 2022) and our preliminary experiments (see Figs. 4-6 in the appendix) suggest that low-rank bias in neural networks improves generalization.

Our study of deep ReLU neural networks trained with mini-batch Stochastic Gradient Descent (SGD) and weight decay shows that the resulting weight matrices tend to be low-rank when training with small batch sizes, high learning rates, or high levels of weight decay. Our theoretical and empirical results provide a better understanding of how these hyperparameters can be used to control the complexity of the network and potentially improve generalization.

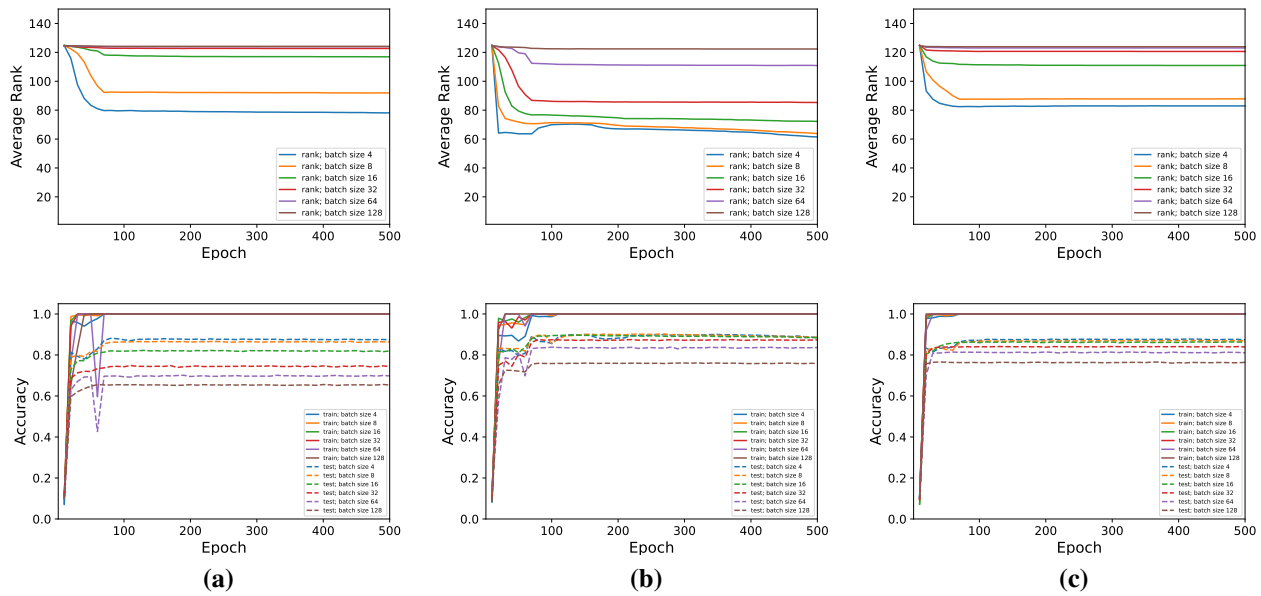


Figure 5. **Average ranks and accuracy rates of ResNet-18 trained on CIFAR10 with different batch sizes. (a)** was trained with $\mu = 1\text{e}{-3}, \lambda = 6\text{e}{-3}$, **(b)** was trained with $\mu = 5\text{e}{-3}, \lambda = 6\text{e}{-3}$, and **(c)** was trained with $\mu = 1\text{e}{-2}, \lambda = 4\text{e}{-4}$. We used a threshold of $\epsilon = 1\text{e}{-2}$.

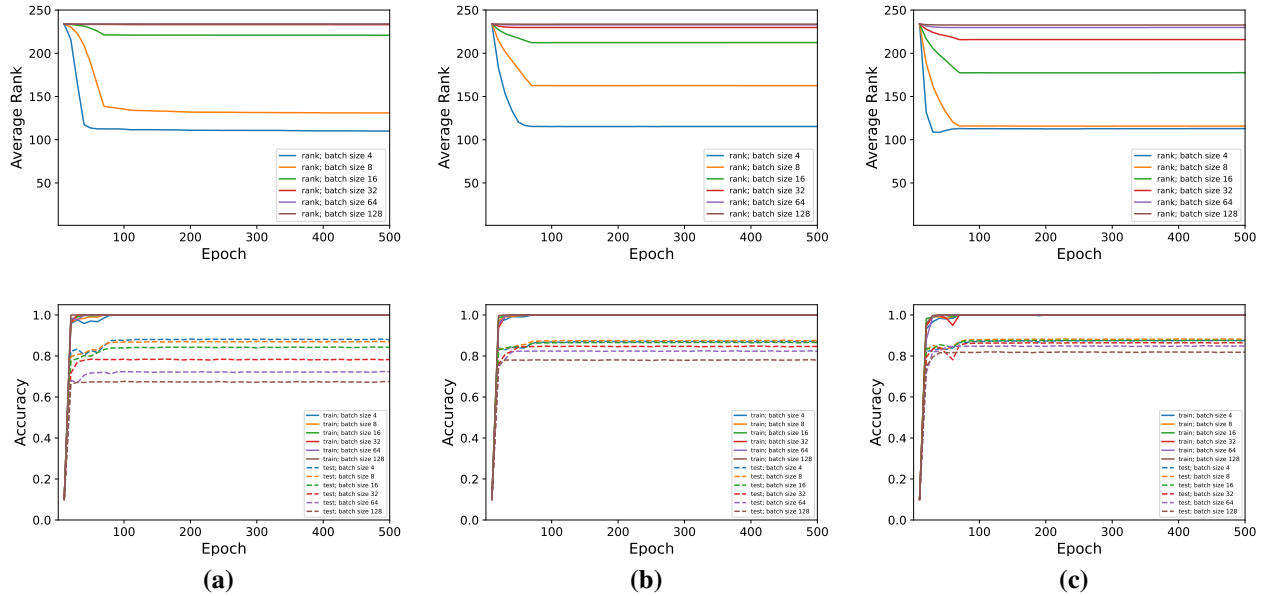


Figure 6. **Average ranks and accuracy rates of VGG-16 trained on CIFAR10 with different batch sizes. (a)** was trained with $\mu = 1\text{e}{-3}, \lambda = 6\text{e}{-3}$, **(b)** was trained with $\mu = 5\text{e}{-3}, \lambda = 5\text{e}{-4}$, and **(c)** was trained with $\mu = 1\text{e}{-2}, \lambda = 4\text{e}{-4}$. We used a threshold of $\epsilon = 4\text{e}{-2}$.

While this paper focused on a basic supervised learning setting using SGD and weight decay, but future studies could investigate the structure of weights and activations in neural networks trained with other optimization methods and regularization techniques. Additionally, it would be valuable to study these biases in unsupervised and self-supervised learning settings for deeper insights into network training. Another interesting direction would be to examine the effects of different architectures such as recurrent neural networks or transformer networks on the rank minimization bias.

Acknowledgements

We thank Mengjia Xu, Akshay Rangamani, Brian Cheung, Qianli Liao, Mikhail Belkin, Eran Malach, and Vardan Papyan for illuminating discussions during the preparation of this manuscript. This material is based upon work supported by the Center for Minds, Brains and Machines (CBMM), funded by NSF STC award CCF-1231216. This research was also sponsored by grants from the National Science Foundation (NSF-0640097, NSF-0827427) and Lockheed Martin Space Advanced Technology Center.

References

Alvarez, J. M. and Salzmann, M. Compression-aware training of deep networks. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, pp. 856–867, Red Hook, NY, USA, 2017. Curran Associates Inc. ISBN 9781510860964.

Arora, S., Ge, R., Neyshabur, B., and Zhang, Y. Stronger generalization bounds for deep nets via a compression approach. In Dy, J. and Krause, A. (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 254–263. PMLR, 10–15 Jul 2018.

Bottou, L. Stochastic gradient learning in neural networks. In *Proceedings of Neuro-Nîmes 91*, Nimes, France, 1991. EC2. URL http://leon.bottou.org/papers/bottou-91c.

Denton, E. L., Zaremba, W., Bruna, J., LeCun, Y., and Fergus, R. Exploiting linear structure within convolutional networks for efficient evaluation. In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N., and Weinberger, K. (eds.), *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014. URL https://proceedings.neurips.cc/paper/2014/file/2afe4567e1bf64d32a5527244d104cea-Paper.pdf.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

Goodfellow, I., Bengio, Y., and Courville, A. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

Gunasekar, S., Woodworth, B., Bhojanapalli, S., Neyshabur, B., and Srebro, N. Implicit regularization in matrix factorization, 2017. URL https://arxiv.org/abs/1705.09280.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016. doi: 10.1109/CVPR.2016.90.

Hoffer, E., Hubara, I., and Soudry, D. Train longer, generalize better: closing the generalization gap in large batch training of neural networks. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/file/a5e0ff62be0b08456fc7f1e88812af3d-Paper.pdf.

Huh, M., Mobahi, H., Zhang, R., Cheung, B., Agrawal, P., and Isola, P. The low-rank simplicity bias in deep networks, 2022. URL https://openreview.net/forum?id=dn4B7Mes2z.

Jastrzebski, S., Kenton, Z., Arpit, D., Ballas, N., Fischer, A., Bengio, Y., and Storkey, A. Three factors influencing minima in sgd, 2017. URL https://arxiv.org/abs/1711.04623.

Ji, Z. and Telgarsky, M. Directional convergence and alignment in deep learning. *CoRR*, abs/2006.06657, 2020. URL https://arxiv.org/abs/2006.06657.

Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M., and Tang, P. T. P. On large-batch training for deep learning: Generalization gap and sharp minima. In *International Conference on Learning Representations (ICLR)*, 2017.

Le, T. and Jegelka, S. Training invariances and the low-rank phenomenon: beyond linear networks. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=XEW8CQgArno.

Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. doi: 10.1109/5.726791.

Li, Z., Luo, Y., and Lyu, K. Towards resolving the implicit bias of gradient descent for matrix factorization: Greedy low-rank learning. *CoRR*, abs/2012.09839, 2020. URL https://arxiv.org/abs/2012.09839.

Neyshabur, B., Bhojanapalli, S., Mcallester, D., and Srebro, N. Exploring generalization in deep learning. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates,

Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/file/10ce03a1ed01077e3e289f3e53c72813-Paper.pdf.

Ongie, G. and Willett, R. The role of linear layers in nonlinear interpolating networks, 2022. URL https://arxiv.org/abs/2202.00856.

Razin, N. and Cohen, N. Implicit regularization in deep learning may not be explainable by norms. *CoRR*, abs/2005.06398, 2020. URL https://arxiv.org/abs/2005.06398.

Simonyan, K. and Zisserman, A. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

Timor, N., Vardi, G., and Shamir, O. Implicit regularization towards rank minimization in relu networks. *CoRR*, abs/2201.12760, 2022. URL https://arxiv.org/abs/2201.12760.

Tukan, M., Maalouf, A., Weksler, M., and Feldman, D. No fine-tuning, no cry: Robust svd for compressing deep networks. *Sensors*, 21(16), 2021. ISSN 1424-8220. doi: 10.3390/s21165599. URL https://www.mdpi.com/1424-8220/21/16/5599.

Yu, X., Liu, T., Wang, X., and Tao, D. On compressing deep models by low rank and sparse decomposition. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 67–76, 2017. doi: 10.1109/CVPR.2017.15.

Zhang, C., Bengio, S., Hardt, M., Recht, B., and Vinyals, O. Understanding deep learning requires rethinking generalization. *CoRR*, abs/1611.03530, 2016. URL http://arxiv.org/abs/1611.03530.

Zhou, Y.-T. and Chellappa, R. Computation of optical flow using a neural network. *IEEE 1988 International Conference on Neural Networks*, pp. 71–78 vol.2, 1988.

Zhu, Z., Wu, J., Yu, B., Wu, L., and Ma, J. The anisotropic noise in stochastic gradient descent: Its behavior of escaping from sharp minima and regularization effects. In *Proceedings of the 36th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*. PMLR, 2019.

A. Proofs

Lemma 3.1. *Let f_W be a neural network and let C^{ij} be a convolutional layer within f_W with parameters matrix W^{ij}. Then,* $\text{rank}\left(\nabla_{W^{ij}} f_W(x)\right) \leq N^{ij}$.

Proof. Let $x \in \mathbb{R}^{c_1 \times h_1 \times w_1}$ be an input tensor, and C^{ij} be a certain convolutional layer with kernel size (k_1, k_2), stride s, and padding p. We wish to show that $\text{rank}\left(\nabla_{W^{ij}} f_W(x)\right) \leq N^{ij}$. We begin by expressing the output of f_W as a sum of paths that pass through C^{ij} and paths that do not. We can express the output as follows:

$$f_W(x) \;=\; \sum_{l_1 \in \text{pred}(l_0)} C^{l_0 l_1} \circ v_{l_1}(x),$$

where $l_0 = L$ and $C^{l_0 l_1} \circ z := C^{l_0 l_1}(z)$. Each layer v_l can also be expressed as:

$$v_{l_1}(x) \;=\; D_{l_1} \odot \sum_{l_2 \in \text{pred}(l_1)} C^{l_1 l_2} \circ v_{l_2}(x),$$

where $D_l := D_l(x) := \sigma'(v_l(x))) \in \mathbb{R}^{c_l \times h_l \times w_l}$.

A path π within the network's graph G is a sequence $\pi = (\pi_0, \ldots, \pi_T)$, where $\pi_0 = 1$, $\pi_T = L$ and for all $i = 0, \ldots, T-1 :\ (v_{\pi_i}, v_{\pi_{i+1}}) \in E$. We can write $f_W(x)$ as the sum of matrix multiplications along paths π from v_1 to v_{l_0}. Specifically, we can write $f_W(x)$ as a follows

$$\begin{aligned} f_W(x) \;=\; & \sum_{\pi \text{ from } i \text{ to } l_0} C^{\pi_T \pi_{T-1}} \circ D_{\pi_{T-1}} \cdots D_{\pi_2} \odot C^{\pi_2 \pi_1} \circ D_{\pi_1} \odot C^{ij} \circ v_j(x) \\ & + \sum_{\substack{\pi \text{ from } 1 \text{ to } l_0 \\ (i,j) \notin \pi}} C^{\pi_T \pi_{T-1}} \circ D_{\pi_{T-1}} \odot C^{\pi_{T-1} \pi_{T-2}} \cdots D_{\pi_2} \odot C^{\pi_2 \pi_1} \circ x, \\ =:\; & A_W(x) + B_W(x) \end{aligned}$$

where $T = T(\pi)$ denotes the length of the path π. Since σ is a piece-wise linear function with a finite number of pieces, for any $x \in \mathbb{R}^{c_1 \times h_1 \times w_1}$, with measure 1 over W, the matrices $\{D_l(x)\}_{l=1}^{L-1}$ are constant in the neighborhood of W. Furthermore, W^{ij} does not appear in the multiplications along the paths π from 1 to l_0 that exclude (i, j). Therefore, we conclude that $\frac{\partial B_W(x)}{\partial W^{ij}} = 0$.

As a next step we would like to analyze the rank of $\frac{\partial A_W(x)}{\partial W^{ij}}$. For this purpose, we rewrite the convolutional layers and the multiplications by the matrices $D_l(x)$ as matrix multiplications.

Representing C^{ij}. We begin by representing the layer C^{ij} as a linear transformation of its input with N^{ij} blocks of W^{ij}. For this purpose, we define a representation of a given 3-dimensional tensor input $z \in \mathbb{R}^{c_j \times h_j \times w_j}$ as a vector $\text{vec}^{ij}(z) \in \mathbb{R}^{N^{ij} c_j k_1 k_2}$. First, we pad z with p rows and columns of zeros and obtain $\text{Pad}_p(z)$. We then vectorize each one of its patches (of dimensions $c_j \times k_1 \times k_2$) that the convolutional layer is acting upon (potentially overlapping) and concatenate them. We can write the vectorized output of the convolutional layer as $U^{ij}\text{vec}^{ij}(z)$, where

$$U^{ij} := \begin{pmatrix} W^{ij} & & 0 & 0 & 0 & 0 \\ & & 0 & 0 & 0 & 0 \\ 0 & 0 & & \ddots & 0 & 0 \\ 0 & 0 & & & 0 & 0 \\ 0 & 0 & 0 & 0 & W^{ij} & \\ 0 & 0 & 0 & 0 & & \end{pmatrix} \tag{1}$$

is a $(N^{ij} c_i) \times (N^{ij} c_j k_1 k_2)$ matrix with N^{ij} copies of W^{ij}. We note that this is a non-standard representation of the convolutional layer's operation as a linear transformation. Typically, we write the convolutional layer as a linear transformation V^{ij} acting on the vectorized version $\text{vec}(z) \in \mathbb{R}^{c_j k_1 k_2}$ of its input z. Since $\text{vec}^{ij}(z)$ consists of the same variables as in $\text{vec}(z)$ with potentially duplicate items, there is a linear transformation that translates $\text{vec}(z)$ into $\text{vec}^{ij}(z)$. Therefore, we can simply write $V^{ij}\text{vec}(z) = U^{ij}\text{vec}^{ij}(z)$.

Representing convolutional layers. Except of C^{ij}, we represent each one of the network's convolutional layers C^{ld} in f_W as linear transformations. As mentioned earlier, we can write $\text{vec}(C^{ld}(z)) = V^{ld}\text{vec}(z)$, for any input $z \in \mathbb{R}^{c_d \times h_d \times w_d}$.

Representing pooling and rearrangement layers. An average pooling layer or a rearrangement layer C^{ld} can be easily represented as a (non-trainable) linear transformation of its input. Namely, we can write $\text{vec}(C^{ld}(z)) = V^{ld}\text{vec}(z)$ for some matrix V^{ld}. A max-pooling layer can be written as a composition of ReLU activations and multiple (non-trainable) linear transformations, since $\max(x, y) = \sigma(x - y) + y$. Therefore, without loss of generality we can replace the pooling layers with non-trainable linear transformations and ReLU activations.

Computing the rank. Finally, we note that $\text{vec}(C^{ij} \circ z) = U^{ij}\text{vec}^{ij}(z) = V^{ij}\text{vec}(z)$, $\text{vec}(D_l \odot z) = P_l \cdot \text{vec}(z)$ for $P_l := \text{diag}(\text{vec}(D_l))$. Therefore, we can write

$$\begin{aligned} A_W(x) &= \sum_{\pi \text{ from } i \text{ to } l_0} W^{\pi_T \pi_{T-1}} \cdot P_{\pi_{T-1}} \cdots P_{\pi_2} \cdot W^{\pi_2 \pi_1} \cdot P_{\pi_1} \cdot U^{ij} \cdot \text{vec}^{ij}(v_j(x)) \\ &=: a(x)^\top \cdot U^{ij} \cdot b(x), \end{aligned}$$

where $a(x)^\top := \sum_{\pi \text{ from } i \text{ to } l_0} W^{\pi_T \pi_{T-1}} \cdot P_{\pi_{T-1}} \cdots P_{\pi_2} \cdot W^{\pi_2 \pi_1} \cdot P_{\pi_1}$ and $b(x) := \text{vec}^{ij}(v_j(x))$. We note that with measure 1, the matrices $\{P_l\}_{l=1}^{L-1}$ are constant in the neighborhood of W. In addition, $a(x)$ and $b(x)$ are computed as multiplications of matrices W^{ld} and P_l excluding $(i, j) = (p, q)$. Therefore, with measure 1 over the selection of W, the Jacobians of $a(x)$ and $b(x)$ with respect to V^{ij} are 0. Furthermore, due to equation 1 and the definition of U^{ij}, we can write

$$a(x)^\top \cdot U^{ij} \cdot b(x) = \sum_{t=1}^{N^{ij}} a_t(x)^\top \cdot W^{ij} \cdot b_t(x),$$

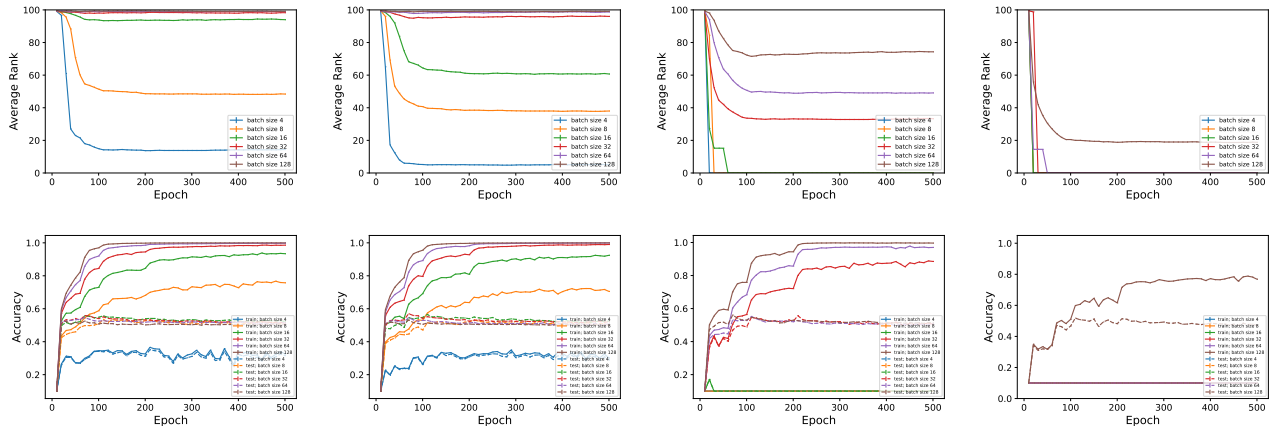
where $a_t(x)$ and $b_t(x)$ are the slices of $a(x)$ and $b(x)$ that are multiplied by the t'th W^{ij} block in U^{ij}. Since the Jacobians of $a_i(x)$ and $b_i(x)$ with respect to W^{ij} are 0 with measure 1 over the selection of W (from the same argument as in the proof of Lem. 3.2), we have,

$$\frac{\partial a(x)^\top \cdot U^{ij} \cdot b(x)}{\partial W^{ij}} = \sum_{t=1}^{N^{ij}} a_t(x) \cdot b_t(x)^\top. \tag{2}$$

Therefore, we conclude that, with measure 1 over the selection of W, we have $\frac{\partial f_W(x)}{\partial W^{ij}} = \sum_{t=1}^{N^{ij}} a_t(x) \cdot b_t(x)^\top$ which is a matrix of rank $\leq N^{ij}$. □

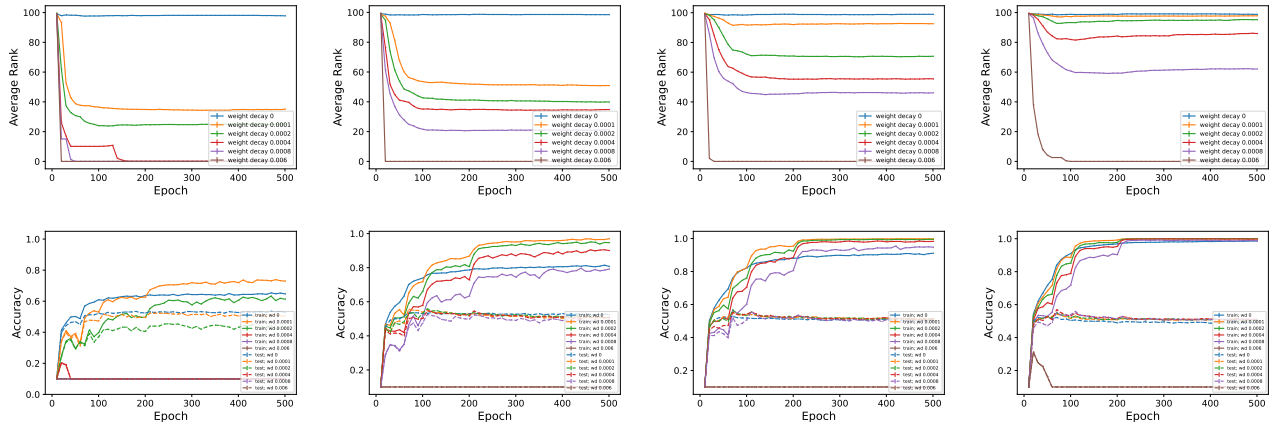
B. Additional Experiments

To further demonstrate the bias towards rank minimization of SGD with weight decay, we conducted a series of experiments with different learning settings. We follow the same training and evaluation protocol described in Sec. 2. The results are summarized in Figs. 7-17.



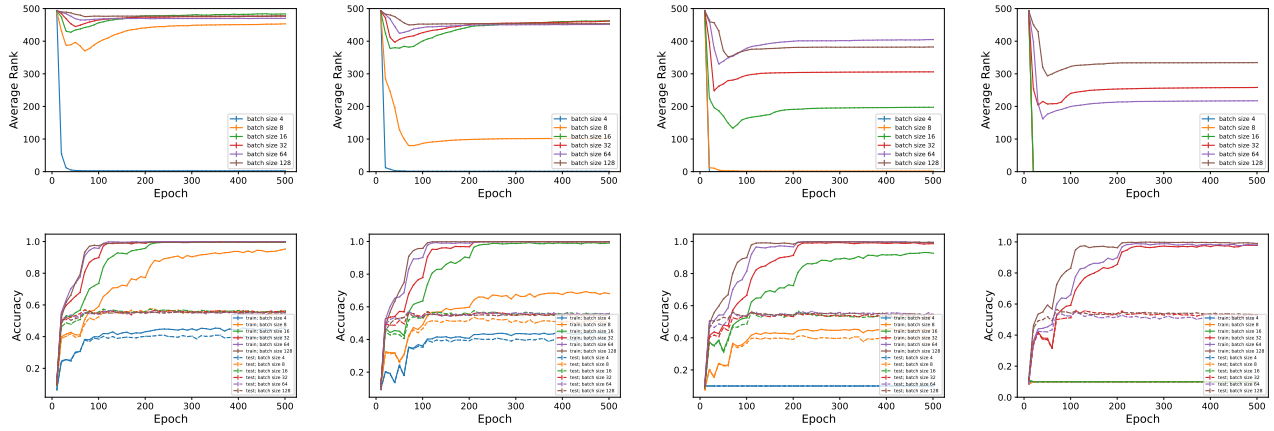
$\mu = 0.001$ $\mu = 0.01$ $\mu = 0.1$ $\mu = 0.5$

Figure 7. **Average rank of MLP-BN-10-100 trained on CIFAR10 with various batch sizes.** In this experiment, $\lambda = 5\text{e}{-4}$ and $\epsilon = 1\text{e}{-3}$.



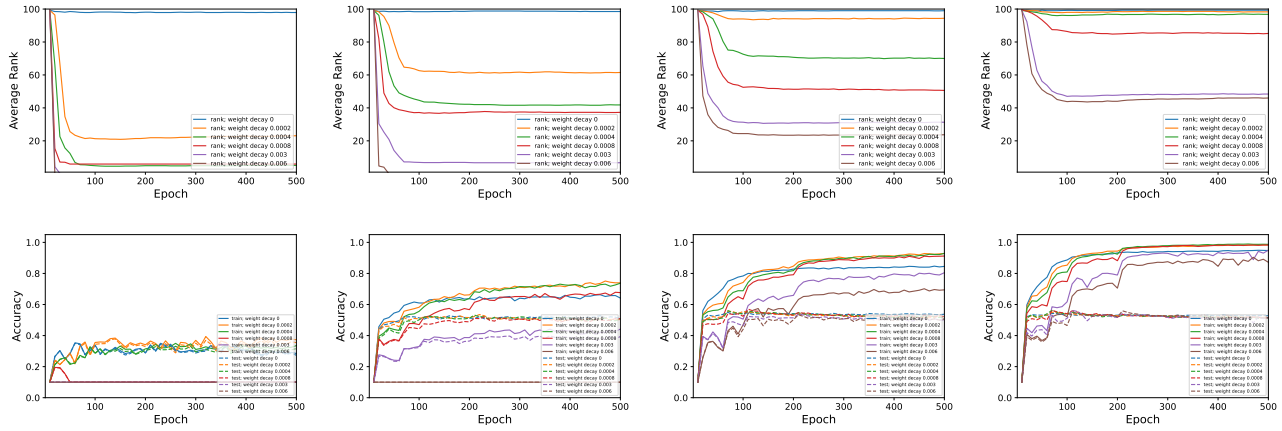
$B = 16$ $B = 32$ $B = 64$ $B = 128$

Figure 8. **Average rank of MLP-BN-10-100 trained on CIFAR10 with varying λ.** In this experiment, $\mu = 0.1$, momentum 0.9 and $\epsilon = 1\text{e}{-3}$.



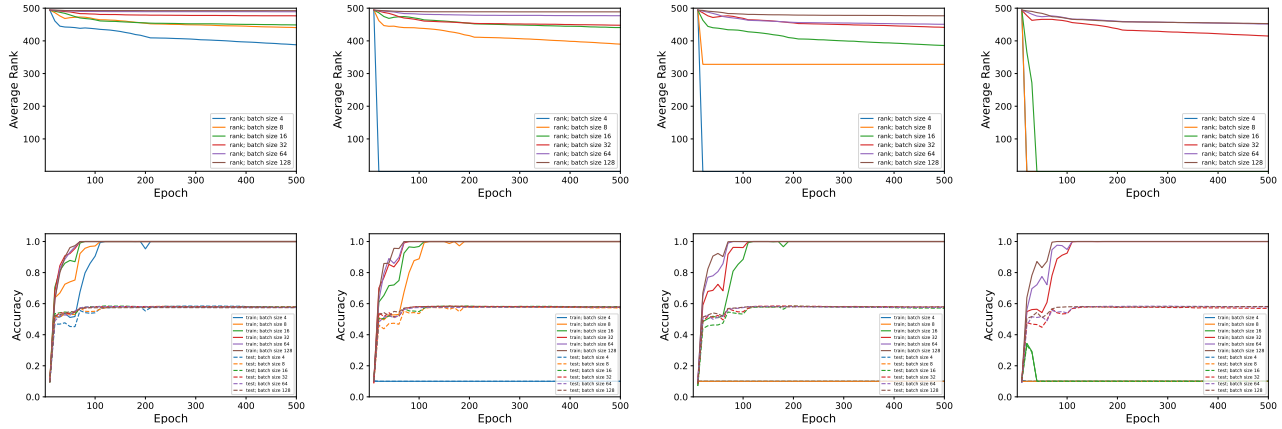
$\mu = 0.0125$ $\mu = 0.025$ $\mu = 0.05$ $\mu = 0.1$

Figure 9. **Average rank of RES-BN-5-500 trained on CIFAR10 with various batch sizes.** In this experiment, $\lambda = 5\text{e}{-4}$ and $\epsilon = 1\text{e}{-3}$.



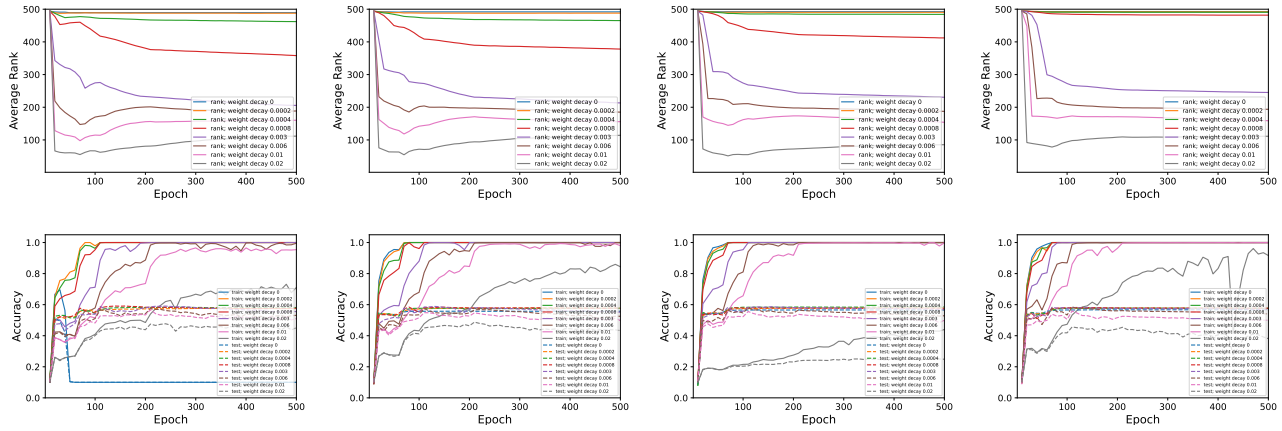
$B = 4$ $B = 8$ $B = 16$ $B = 32$

Figure 10. **Average ranks and accuracy rates of MLP-BN-10-100 trained on CIFAR10 with varying λ.** In this experiment, $\mu = 0.1$ and $\epsilon = 1\text{e}{-3}$.



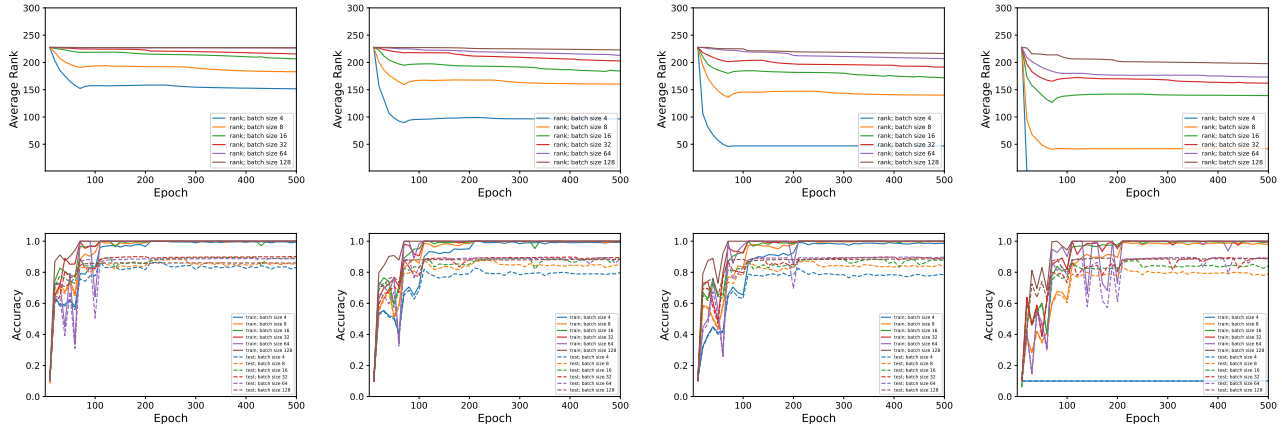
$\mu = 0.05$ $\mu = 0.1$ $\mu = 0.2$ $\mu = 0.3$

Figure 11. **Average ranks and accuracy rates of MLP-5-500 trained on CIFAR10 with various batch sizes.** In this experiment, $\lambda = 5\text{e}{-4}$ and $\epsilon = 1\text{e}{-3}$.



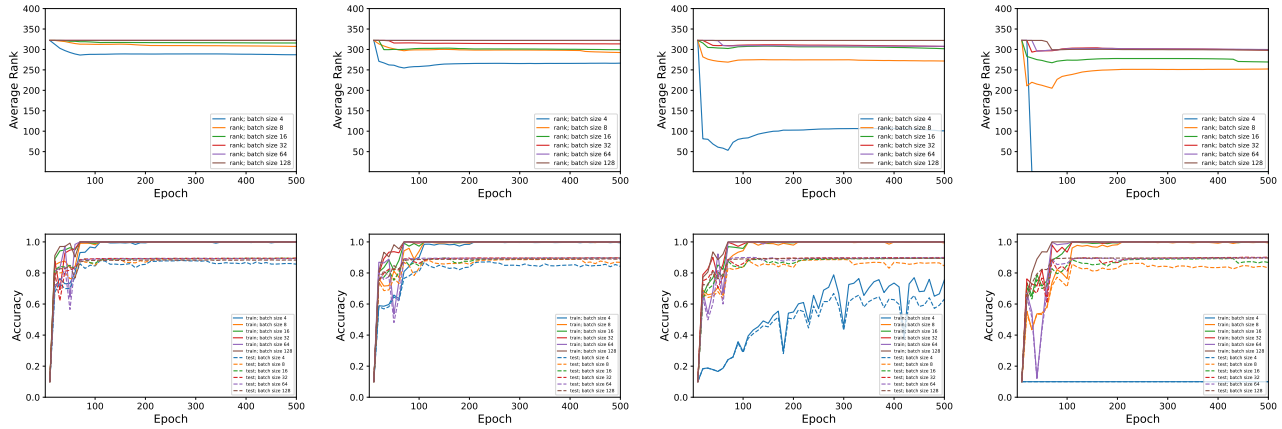
$B = 4$ $B = 8$ $B = 16$ $B = 32$

Figure 12. **Average ranks and accuracy rates of MLP-5-500 trained on CIFAR10 with varying λ.** The models were trained with a $\mu = 0.025$ initial learning rate. To estimate the rank, we used an $\epsilon = 0.001$ threshold.



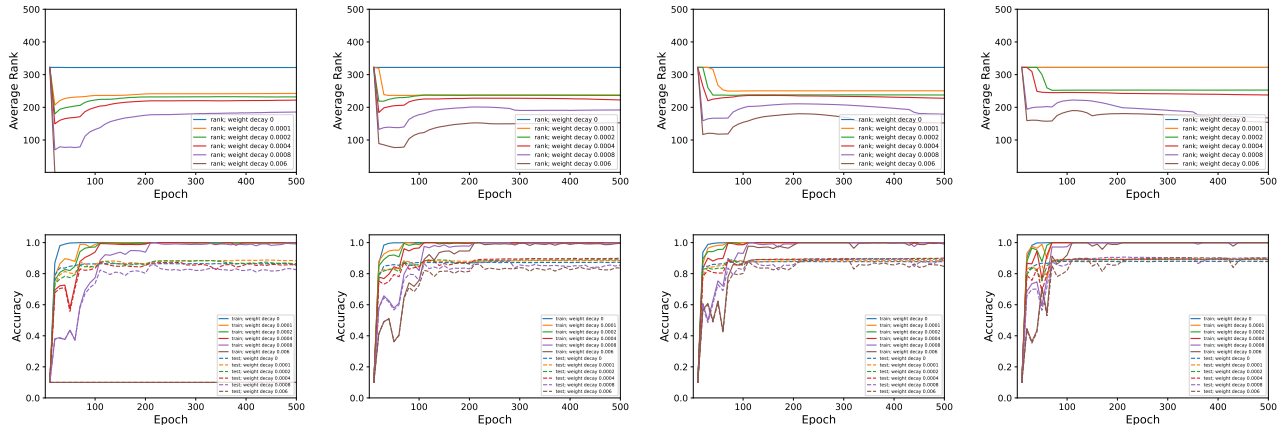
$\mu = 0.5$ $\mu = 0.8$ $\mu = 1.0$ $\mu = 2.0$

Figure 13. **Average ranks and accuracy rates of ResNet-18 trained on CIFAR10 with various batch sizes.** In this experiment, $\lambda = 5\text{e}{-4}$ and $\epsilon = 1\text{e}{-3}$.



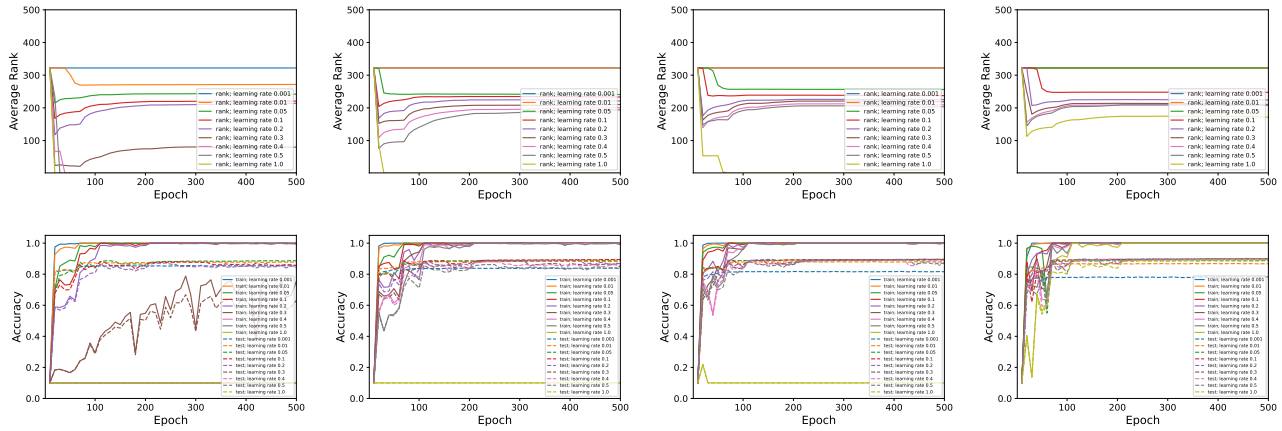
$\mu = 0.1$ $\mu = 0.2$ $\mu = 0.3$ $\mu = 0.5$

Figure 14. **Average ranks and accuracy rates of VGG-16 trained on CIFAR10 with various batch sizes.** In this experiment, $\mu = 5\text{e}{-4}$ and $\epsilon = 1\text{e}{-3}$.



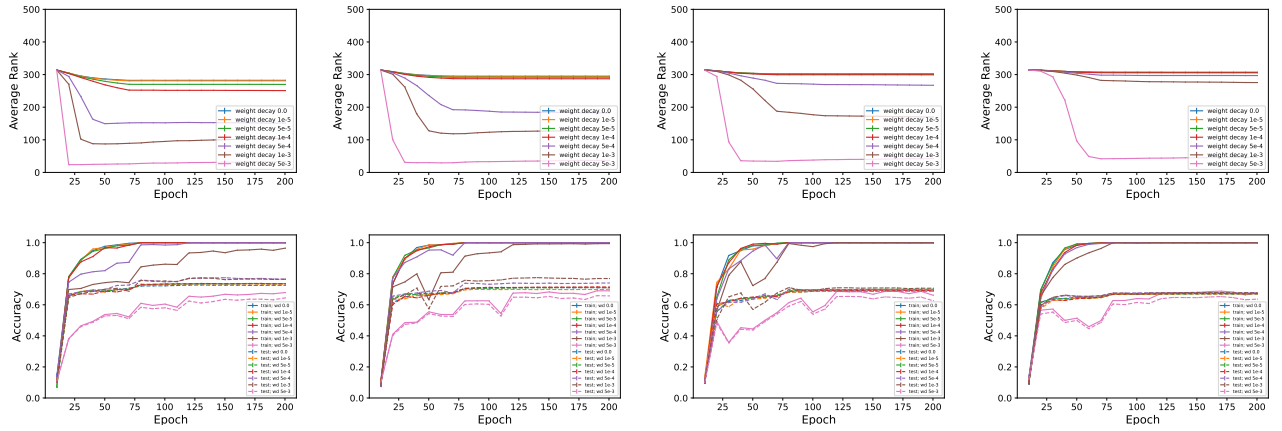
$B = 4$ $B = 8$ $B = 16$ $B = 32$

Figure 15. **Average ranks and accuracy rates of VGG-16 trained on CIFAR10 with varying λ.** In this experiment, $\mu = 0.1$ and $\epsilon = 0.01$.



$B = 4$ $B = 8$ $B = 16$ $B = 32$

Figure 16. **Average ranks and accuracy rates of VGG-16 trained on CIFAR10 with varying μ.** In this experiment, $\mu = 5\text{e}{-4}$ and $\epsilon = 1\text{e}{-3}$.



$B = 16$ $B = 32$ $B = 64$ $B = 128$

Figure 17. **Average ranks and accuracy rates of ViT trained on CIFAR10 with varying λ.** In this experiment, $\mu = 4\text{e}{-2}$ and $\epsilon = 0.01$.