Low-Rank Learning by Design: the Role of Network Architecture and Activation Linearity in Gradient Rank Collapse

Anonymous Author(s)
Affiliation
Address
`email`

Abstract

Our understanding of learning dynamics of deep neural networks (DNNs) remains incomplete. Recent research has begun to uncover the mathematical principles underlying these networks, including the phenomenon of "Neural Collapse", where linear classifiers within DNNs converge to specific geometrical structures during late-stage training. However, the role of geometric constraints in learning extends beyond this terminal phase. For instance, gradients in fully-connected layers naturally develop a low-rank structure due to the accumulation of rank-one outer products over a training batch. Despite the attention given to methods that exploit this structure for memory saving or regularization, the emergence of low-rank learning as an inherent aspect of certain DNN architectures has been under-explored. In this paper, we conduct a comprehensive study of gradient rank in DNNs, examining how architectural choices and structure of the data affect gradient rank bounds. Our theoretical analysis provides these bounds for training fully-connected, recurrent, and convolutional neural networks. We also demonstrate, both theoretically and empirically, how design choices like activation function linearity, bottleneck layer introduction, convolutional stride, and sequence truncation influence these bounds. Our findings not only contribute to the understanding of learning dynamics in DNNs, but also provide practical guidance for deep learning engineers to make informed design decisions.

1 Introduction

Deep Neural Networks (DNNs) continue to achieve state-of-the-art performance on a number of complex data sets for a diverse array of tasks; however, modern DNN architectures are notoriously complex, with millions of parameters, nonlinear interactions, and dozens of architectural choices and hyper-parameters which can all significantly affect model performance. Internal complexity and a lack of thorough theoretical groundwork has given DNNs a reputation as "black box" models, where architectures may excel or fail on a given problem with relatively little indication how their structure facilitated that performance. Engineering a neural network that works well on a particular problem can often take the form of fairly arbitrary and exhaustive model tweaking, and even in cases where researchers systematically perturb particular settings, the primary explanations of performance come down to observing minor changes in performance evaluation metrics such as loss, accuracy/precision, dice-scores or other related metrics. In this work, we examine a particular emergent phenomenon in deep neural networks—the collapse of gradient rank during training; however, we take a theory-first approach, examining how bounds on gradient rank collapse appear naturally and deterministically as a result of particular architectural choices such as bottleneck layers, level of nonlinearity in hidden activations, and parameter tying.

This work is part of a growing body of theoretical research studying the dynamics of learning in deep neural networks. Beginning from first principles, Andrew Saxe provided a foundational work on exact solutions to nonlinear dynamics which emerge in fully-connected networks with linear activations [24], which has inspired a body of related work on simple networks with various architectural or learning setups such as parent-teacher interactions [10], online learning and overparametrization [11], gated networks [23]. This work on theory and dynamics has also been extended to studying high-dimensional dynamics of generalization error [2], emergence of semantic representations [25], and other phenomemon which can be first characterized mathematically and observed empirically. Our work in this paper follows this tradition in the literature of beginning with mathematical principles which affect learning dynamics in simple networks, and demonstrating how these principles emerge in practical scenarios.

An additional related body of work studies the phenomemon of Neural Collapse [14, 21], in which deep classifier neural networks converge to a set of rigid geometrical constraints during the terminal phase of training, with the geometry of the output space determined exactly by the number of classes (i.e., the rank of the output space). This neural collapse phenomenon has been thoroughly studied as emerging in constrained [30] and unconstrained feature settings [19, 28, 34], with various loss functions [12, 17, 33], under transfer learning regimes [9], class imbalance scenarios [27], and exemplifying affects on the loss-landscapes [32]. This growing body of work suggests that geometric constraints during learning influence a number of desirable and undesirable deep learning behaviors in practice.

Like the works on neural collapse, we are interested in geometric constraints to learning; however, we follow the example of Saxe et al. and begin our theoretical examination with simple linear networks, showing how we can expand on simple constraints of batch size (which has been discussed previously elsewhere [20]) to constraints dependent on a number architectural features such as bottlenecked layers, parameter-tying, and level of linearity in the activation. Our work invites the study of network-wide geometric constraints, and while we do not dive into training dynamics in this work, we are able to set a stage which bounds dynamics, hopefully clearing the way for further dynamical analysis in the style of Saxe, Tishby and others.

Finally, our work studying the affect of a particular nonlinear activation and its level of linearity stands out from the existing work on purely linear networks and neural collapse in linear classifiers. Although nonlinear activations introduce some complexity to analysis, we can draw from some previous work on analyzing the spectrum of ReLU activations [7], extending the analysis in that work to its natural consequences with Leaky-ReLU activations and even deriving explicit bounds for numerical estimation of rank which require only singular values of the underlying linear transformations.

Our derivation of upper bounds on gradient dynamics during training also has implications for distributed models which utilized low-rank decompositions for efficient communication. For example, PowerSGD [29] and distributed Auto-Differentiation [3] compute a low-rank decomposition of the gradient prior to transfer between distributed data-collection sites. Our theoretical results here can help to provide insights into how high of a rank may be needed to preserve model performance between the pooled and distributed cases, or how much information may be lost when a lower-rank decomposition is used for communication efficiency.

Our primary results in this work are theoretical; however, we perform a number of empirical verifications and demonstrations which verify our theoretical results and study the implications of our derived bounds on gradient rank for various archictural design choices. In sum, the contributions here include: 1. an upper bound on the rank of the gradient in linear networks; 2. upper bounds on the rank of the gradient in linear networks with shared parameters, such as RNNs and CNNs; 3. extension of previous work on ReLU Singular Values to study the effect of Leaky-ReLUs and the level of linearity on the upper bound of rank; 4. empirical results on numerical data which verify our bounds and implicate particular architectural choices; 5. empirical demonstration of theoretical implications on large-scale networks for Computer Vision and Natural Language Processing; 6. natural extensions in the future work to rank dynamics during training, explicit connections to neural collapse, and implications for rank effects of other architectural phenomena such as dropout layers, batch norms, and more.

2 Theoretical Methods

2.1 Reverse-Mode Auto-Differentiation

We will define a simple neural network with depth L as the operator $\Phi(\{\mathbf{W}_i\}_{i=0}^L, \{\mathbf{b}_i\}_{i=0}^L, \{\phi_i\}_{i=0}^L) : \mathbb{R}^m \to \mathbb{R}^n$. This is given in as a set of weights $\{\mathbf{W}_1, \dots, \mathbf{W}_L\}$, bias variables $\{\mathbf{b}_1, \dots, \mathbf{b}_L\}$, and activation functions $\{\phi_1, \dots, \phi_L\}$, where each function $\phi_i : \mathbb{R}^n \to \mathbb{R}^n$ is an element-wise operator on a vector space.

Let $\mathbf{x} \in \mathbb{R}^m$ be the input into the network and let $\mathbf{y} \in \mathbb{R}^n$ be a set of target variables. Then we define z_i as the *internal activations* and a_i as the external activations at layer i given with the recursive relation:

$$\begin{aligned}
\mathbf{z}_0 &= \mathbf{x} \\
\mathbf{a}_0 &= \mathbf{z}_0 \\
\mathbf{z}_i &= \mathbf{W}_i \mathbf{a}_{i-1} + \mathbf{b}_i \\
\mathbf{a}_i &= \phi_i(\mathbf{z}_i)
\end{aligned}$$

To define the sizes of the hidden layers, we have $\mathbf{W}_i \in \mathbb{R}^{h_{i-1} \times h_i}$, with $h_0 = m$ and $h_L = n$. We note then that $\mathbf{z}_i, \mathbf{a}_i$ are column vectors in $\mathbb{R}^{h_i}$.

Let $\mathcal{L}(\mathbf{y}, \mathbf{a}_L) : \mathbb{R}^n \to \mathbb{R}$ be a loss function which measures the *error* of the estimate of $\mathbf{y}$ at $\mathbf{a}_L$. The gradient update for this loss, computed for the set of weights $\mathbf{W}_i$ can be written as the outer-product

$$\nabla_{\mathbf{W}_i} = \mathbf{a}_{i-1} \delta_i^\top$$

where δ_i is the partial derivative of the output at layer i w.r.t its input. At the output layer L, δ_L is computed as

$$\delta_L = \frac{\partial \mathcal{L}}{\partial \mathbf{a}_L} \odot \frac{\partial \phi_L}{\partial \mathbf{z}_L}$$

and subsequent δ_i are computed as

$$\delta_i = \delta_{i+1} \mathbf{W}_{i+1} \odot \frac{\partial \phi_i}{\partial \mathbf{z}_i}.$$

These definitions are all given for x as a column vector in $\mathbb{R}^m$; however, for standard batch SGD we compute these quantities over a batch of N samples. If we rewrite the variables $\mathbf{x}$, $\mathbf{y}$, $\mathbf{z}_i$, $\mathbf{a}_i$ and δ_i as matrices $\mathbf{X} \in \mathbb{R}^{N \times m}$, $\mathbf{Y} \in \mathbb{R}^{N \times n}$, $\mathbf{Z}_i, \mathbf{A}_i, \mathbf{\Delta}_i \in \mathbb{R}^{N \times h_i}$. The gradient can then be computed as the matrix-product

$$\sum_k^N \mathbf{a}_{n,i-1} \delta_{n,i}^\top = \mathbf{A}_{i-1}^\top \mathbf{\Delta}_i.$$

3 Theoretical Results

3.1 Bounds on Gradients in Linear Networks

First consider the set of neural networks where $\phi_i(\mathbf{x}) = \mathbf{x}$ is the identity operator $\forall i$. These networks are called "linear networks" or equivalently "multi-layer perceptrons" (MLP), and $\mathbf{Z}_i = \mathbf{A}_i, \forall i$. For these networks, the rank of the gradients has an exact bound. Trivially, for a given gradient $\nabla_{\mathbf{W}_i}$, the rank is

$$\mathrm{rank}(\nabla_{\mathbf{W}_i}) = \mathrm{rank}(\mathbf{Z}_{i-1}^\top \mathbf{\Delta}_i) \leq \min\{\mathrm{rank}(\mathbf{Z_{i-1}}), \mathrm{rank}(\mathbf{\Delta}_i)\}.$$

Since in linear networks we can easily compute $\mathbf{Z}_i = \mathbf{X} \prod_{j=1}^i \mathbf{W}_j$, we use a similar rule as for the gradient to compute the bound

$$\mathrm{rank}(\mathbf{Z}_i) \leq \min\{\mathrm{rank}(\mathbf{X}), \mathrm{rank}(\mathbf{W}_1), \dots, \mathrm{rank}(\mathbf{W}_i)\}.$$

For the adjoint variable Δ_i we can use the fact that $\frac{\partial \phi}{\partial \mathbf{Z}_i} = \mathbf{1}$ where $\mathbf{1}$ is a matrix of ones in $\mathbb{R}^{N \times h_i}$ to derive a bound on the adjoint as

$$\mathrm{rank}(\mathbf{\Delta}_i) \leq \min\{\mathrm{rank}(\mathbf{W}_i), \mathrm{rank}(\mathbf{W}_{i+1}), \dots, \mathrm{rank}(\mathbf{W})_L, \mathrm{rank}(\frac{\partial \mathcal{L}}{\partial \mathbf{Z}_L})\}.$$

Therefore, the bound for the gradient rank is

$$\operatorname{rank}(\nabla_{\mathbf{W}_i}) \leq \min\{\operatorname{rank}(\mathbf{Z_{i-1}}), \operatorname{rank}(\mathbf{\Delta}_i)\} \tag{1}$$

$$\leq \min\{\operatorname{rank}(\mathbf{X}), \operatorname{rank}(\mathbf{W}_i), \operatorname{rank}(\mathbf{W}_{i+1}), \ldots, \operatorname{rank}(\mathbf{W})_L, \operatorname{rank}(\frac{\partial \mathcal{L}}{\partial \mathbf{Z}_L})\} \tag{2}$$

3.2 Bounds on Gradients in Linear Networks with Parameter Tying

We will begin our analysis with recurrent neural networks and BPTT.

3.2.1 Recurrent Layers

Let $\mathcal{X} \in \mathbb{R}^{N \times n \times T}$ be the N samples of an n-dimensional variable over a sequence of length T (over time, for example). We will set an initial hidden state for this layer as $\mathbf{H}_{i,0} \in \mathbb{R}^{N \times h_i}$.

Let $f_i : \mathbb{R}^{N \times h_{i-1} \times T} \to \mathbb{R}^{N \times h_i \times T}$ be the function given by a linear layer with a set of input weights $\mathbf{U} \in \mathbb{R}^{h_{i-1} \times h_i}$ and a set of hidden weights $\mathbf{V} \in \mathbb{R}^{h_i \times h_i}$. The output of this layer is computed as the tensor

$$f_i(\mathcal{X}) = \{\mathbf{H}_{t,i}\}_{t=1}^T = \{\phi_i(\mathbf{Z}_{t,i})\}_{t=1}^T = \{\phi_i(\mathbf{X}_t \mathbf{U}_i + \mathbf{H}_{t-1,i}\mathbf{V}_i)\}_{t=1}^T.$$

Supposing the error i feeds into another recurrent layer, the error on the output (which is used for the gradients of both $\mathbf{U}$ and $\mathbf{V}$) is thus computed as the tensor

$$\mathcal{D}_i = \{\mathbf{\Delta}_{t,i}\}_{t=1}^T = \left\{(\Delta_{t,i+1}\mathbf{U}_{i+1} + \mathbf{\Delta}_{t+1,i}\mathbf{V}_i) \odot \frac{\partial \phi_i}{\partial \mathbf{Z}_{t,i}}\right\}_{t=1}^T .$$

where we have $\mathbf{\Delta}_{T+1,i} = \mathbf{0}$ for convenience of notation. In the case where the next layer in the network is not recurrent (for example if it is a linear layer receiving flattened output from the RNN), we can set $\mathbf{\Delta}_{t,i+1}$ to be the elements of $\mathbf{\Delta}_{i+1}$ which correspond to each timeptoint t

The gradients for $\mathbf{U}_i$ and $\mathbf{V}_i$ are then computed as the sum over the products of each element in the sequence

$$\nabla_{\mathbf{U}_i} = \sum_{t=1}^{T} \mathbf{X}_{t,i}^\top \mathbf{\Delta}_{t,i} \qquad \nabla_{\mathbf{V}_i} = \sum_{t=0}^{T-1} \mathbf{H}_{t,i}^\top \mathbf{\Delta}_{t,i}$$

Fully linear RNNs are not typically implemented in practice; however, for the purpose of demonstrating how parameter-tying can improve with parameter-tying, the analysis may still prove helpful. The first thing to notice is that even for as small as $T = 2$, we reach the potential for full-rank gradients quite quickly. Even in the degenerate case when the batch size is $N = 1$, $\nabla_{\mathbf{U}_i}$ and $\nabla_{\mathbf{V}_i}$ may become rank T. Thus, the analysis of rank no longer depends much on the architecture beyond the number of timepoints chosen, and parameter-tying can affect rank-collapse that emerges from low-rank product bottlenecks in other layers. Rather, it will become of interest to look at correspondences between input such as temporal correlation in order to provide a clearer picture. We will leave this for future work.

3.2.2 Convolutional Layers

We can derive similar bounds to those for recurrent networks for the gradients over convolutional layers. At its core, the underlying principle of sharing gradients over a sequence remains the same; however, the particular architectural choices (stride, image size, kernel size, padding, dilation) which influence the length of this sequence are more numerous for convolutional layers. Thus, since the core idea is the same as for RNNs, our theoretical bounds on gradient rank in convolutional layers are included in our supplementary material.

3.3 Bounds on Gradients in Leaky-ReLU Networks

The bounds we provide on gradient rank in networks with purely linear activations builds off intuitive principles from linear algebra; however, the introduction of nonlinear activations confuses these notions. For a general nonlinear operator phi, the notion of singular values which we obtain from linear algebra does not hold. Even though we can compute the SVD of the matrix $\phi_\alpha(\mathbf{Z})$ for a given layer with internal activations $\mathbf{Z}$, little can be initially said about the relationship of the this decomposition to the linear transformations which generated $\mathbf{Z}$. Thus, although we can still empirically compute the rank of the resulting matrix from the nonlinear transformation, it is not initially clear how rank-deficiency will propagate through a network as it will in the fully linear case. In this section, we show how Leaky-ReLU activations with different levels of nonlinearity affect numerical estimates of rank. In supplementary material, we also explore theoretical connections to previous work on so-called "ReLU Singular Values", which also follow our derived bounds.

3.3.1 Numerical effect of Leaky-ReLUs on Rank

In this section, we analyze the numerical effect of leaky-relu nonlinearities on the singular values of the internal activations $\mathbf{Z}_i = \mathbf{A}_{i-1}\mathbf{W}_i$. Our tactic will be to observe how the slope α can push the singular values of $\phi_\alpha(\mathbf{Z}_i)$ above a numerical threshold used for estimating rank. As a choice of threshold, we use $\epsilon \cdot \max_i \sigma_i$

where ϵ is the machine-epsilon for floating-point calculations on a computer. This threshold is utilized in most modern libraries which can perform rank estimation, including PyTorch[22] and Tensorflow[1]. We then say that a singular value σ_k does not contribute to our estimation of rank if

$$\sigma_k < \epsilon \cdot \max_i \sigma_i \tag{3}$$

Let $\mathbf{D}_\alpha(\mathbf{Z}) \in \mathbb{R}^{h \times h}$ be the matrix with entries corresponding to the linear coefficients from the LeakyReLU activation of $\mathbf{Z}_i = \mathbf{X}\mathbf{W}_i$ for $\mathbf{W} \in \mathbb{R}^{h-1 \times h}$. Leaky-ReLU activations can be written as the hadamard product

$$\phi_\alpha(\mathbf{Z}_i) = \mathbf{D}_\alpha \odot \mathbf{Z}_i \tag{4}$$

From Zhan et al. 1997 [31], we have the inequality for the singular values of the Hadamard product:

$$\sum_{i=1}^{k} \sigma_i(\mathbf{D}_\alpha \odot \mathbf{Z}_i) < \sum_{i=1}^{k} \min\{c_i(\mathbf{D}_\alpha), r_i(\mathbf{D}_\alpha)\}\sigma_i(\mathbf{Z}_i) \tag{5}$$

where $c_1(\mathbf{D}_\alpha) \geq c_2(\mathbf{D}_\alpha) \geq \cdots \geq c_h(\mathbf{D}_\alpha)$ are the 2-norm of the columns sorted in decreasing order, and $r_i(\mathbf{D}_\alpha)$ are the 2-norm also in decreasing order.

We can say that $\mathbf{D}_\alpha \odot \mathbf{Z}$ will remain numerically rank-deficient up to rank k when

$$\min\{c_k(\mathbf{D}_\alpha), r_k(\mathbf{D}_\alpha)\}\sigma_k(\mathbf{Z}) \leq \epsilon\sigma_1(\mathbf{D}_\alpha \odot \mathbf{Z}) \tag{6}$$

We then have two cases: 1) $\sigma_1(\mathbf{D}_\alpha \odot \mathbf{Z}) \leq \sigma_1(\mathbf{Z})$ or 2) $\sigma_1(\mathbf{D}_\alpha \odot \mathbf{Z}) > \sigma_1(\mathbf{Z})$. If case 1) holds, the hadamard product will remain rank deficient if the euclidean length of the corresponding column or row of $\mathbf{D}_\alpha$ is less than or equal to $\epsilon\sigma_1(\mathbf{Z})/\sigma_k(\mathbf{Z})$, and will increase otherwise.

Case 2 is best analyzed by a choice of $\mathbf{Z}$ which saturates σ_1 at the bound in equation 6, since it cannot grow anywhere beyond that bound. If we have $\sigma_1(\mathbf{D}_\alpha \odot \mathbf{Z}) = \min\{c_1(\mathbf{D}_\alpha), r_1(\mathbf{D}_\alpha)\}\sigma_1(\mathbf{Z})$, σ_k will always contribute numerically to the rank when the corresponding column/row norm at k equals

$$\epsilon \min\{c_1(\mathbf{D}_\alpha), r_1(\mathbf{D}_\alpha)\}\sigma_1(\mathbf{Z})/\sigma_k(\mathbf{Z}). \tag{7}$$

Because we are dealing with leaky-ReLU activations in particular, the 2-norm of $c_i(\mathbf{D}_\alpha)$ and $r_i(\mathbf{D}_\alpha)$ take on a particular closed-form. Indeed, we have

$$c_i(\mathbf{D}_\alpha) = \sqrt{N_-\alpha^2 + N_+},\ ,\ r_i(\mathbf{D}_\alpha) = \sqrt{M_-\alpha^2 + M_+}$$

where N_- is the number of rows in column i which fall into the negative domain of the leakyrelu activation, and N_+ is the number of rows which fall into the positive domain. Similarly, M_- and M_+ count the number of columns in the negative and positive domains in row i. In other words, we can exactly determine the threshold at which a singular value will fail to contribute to numerical estimation of rank if we know the non-negative coefficient and can count the number of samples which reside in particular regions of the activation function's domain.

So far, we have confined our theoretical analysis to the affect of rank on activations, and have not given any discussion to the effect on the adjoint variables Δ. It turns out that a similar kind of analysis to what is performed here can be applied to ***any piecewise linear function***, and since the derivative of Leaky-ReLU is piecewise-linear, we can derive a similar bound to the one provided in equation 6. We have provided the full derivation for the affect of Leaky-ReLUs on adjoint rank in the supplement.

Dataset	Samples (N)	Subspace	Data Type
Gaussian	16384	$\mathbb{R}^{N\times m}$	Numerical
Sinusoids	16384	$\mathbb{R}^{N\times m\times T}$	Numerical
MNIST	60000	$\mathbb{R}^{N\times H\times W}$	Image
CIFAR10	60000	$\mathbb{R}^{N\times H\times W}$	Image
TinyImageNet	100000	$\mathbb{R}^{N\times 3\times H\times W}$	Image
WikiText	$> 10^8$	$\mathbb{R}^{N\times \|V\|\times T}$	Text
Multi30k	300000+	$\mathbb{R}^{N\times \|V\|\times T}$	Text

Table 1: The Data Sets evaluated as part of our empirical verification. For space, only the Gaussian and Sinusoid data sets are included in the main body of text, and the remaining data sets are included in the supplementary material.

Model	Datasets
MLP	Gaussian
Elman RNN	Sinusoids
BERT	WikiText
XLM	Multi30k
ResNet16	MNIST, CIFAR10, ImageNet
VGG11	MNIST, CIFAR10, ImageNet

Table 2: The models evaluated as part of our empirical verification. For space and demonstrating key features, we include results from the Fully Connected Network, Elman RNN, and ResNet16 in the main text. Additional model architectures are included in the supplement.

4 Empirical Methods

We perform two broad classes of experiment: simple verification, and large-scale demonstration. In the first class of experiments, we show how the bounds derived in our theoretical results appear in simple, numerical experiments, where it is easy to verify how particular architectural choices and level of nonlinearity affect the bound of the gradient rank. In the second class of experiments, we perform larger-scale experiments and demonstrate how our derived bounds can also affect these models in practice. These latter class of experiments utilize models which include modules such as Drop-Out, BatchNorms, and LayerNorms. We do not explore these additional modules theoretically in this work, even though they may have an influence on gradient rank [4]; however, we believe this allows for a productive direction for future theoretical and empirical work.

In both styles of experiment, the primary architectural elements we demonstrate when possible are: 1) bottleneck layers, i.e., layers within a network which have a much smaller number of neurons than their input and output spaces, 2) length of sequences in parameter tying, 3) low-rank input/output spaces, 4) level of non-linearity in hidden activations.

For the sake of numerical verification, we implement auto-encoding networks on two numerical datasets. For our first data set, we generate an m-dimensional gaussian $X \in \mathbb{R}^{N\times m}$ as $\mathbf{x} \sim \mathcal{N}(\mu_i, \Sigma_i)$ We then use a fully-connected network as an auto-encoder to reconstruct the random samples.

Our second kind of numerical data is generated by sine waves of the form $\mathbf{x}_{i,j} = a_{i,j} \sin(b_{i,j}t) + c_{i,j}$, where we sample the parameters $a_{i,j}, b_{i,j}, c_{i,j}$ for a particular sample i and dimension j independently from their own univariate Gaussian distributions $a_{i,j} \sim \mathcal{N}(\mu_a, \sigma_a), b_{i,j} \sim \mathcal{N}(\mu_b, \sigma_b), c_{i,j} \sim \mathcal{N}(\mu_c, \sigma_c)$. We choose t to be T-many points in the interval $[-2\pi, 2\pi]$. We then used an RNN with Elman Cells, GRUs and LSTMs to reconstruct the input sequence, and demonstrate how four architectural principles affect our derived bound gradient rank.

For our larger-scale experiments, we choose two popular data sets from computer vision and natural language processing. We choose Cifar10[15] and a Tiny-ImageNet (a Subset of ImageNet[5]) for computer vision, and we choose WikiText[18] for natural language processing. Because our empirical analysis requires repeated singular value decompositions and multiple architectural tweaks, we avoid overly long experiment runtimes by using relatively smaller-sized versions of popular network architectures which can fit alongside batches on single GPUs. In terms of computer-vision architectures, we utilize ResNet16[13] and VGG11[26], and for natural language processing, we utilize the BERT [6] and XLM [16] transformers for Language Modeling and Machine Translation respectively. Again, since we are interested primarily in rank as initially bounded by architecture, we do not use pretrained weights, and we only train models for a maximum of 100 epochs.

For both numerical verification and large-scale experiments, all models were trained using $k = 5$-fold cross validation, with 4 uniquely initialized models on each fold for a total of 20 models per result. The rank metrics in the following section are thus accumulated over these 20 runs.

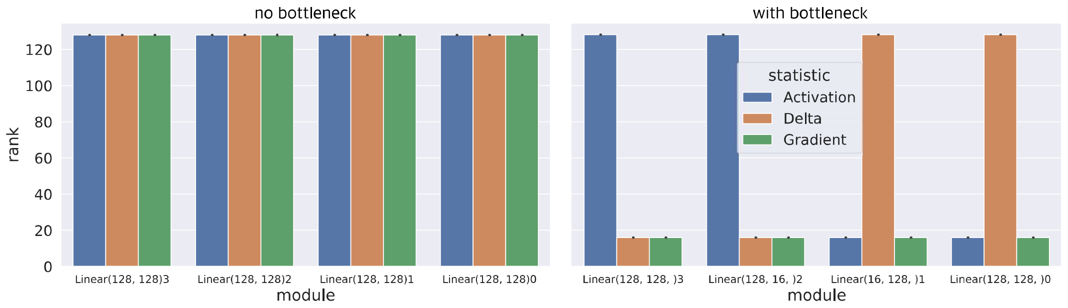

Figure 1: For a 3-layer Linear FC network, we plot the mean rank of gradients, activation, and deltas change with respect to the size of a neuron bottleneck in the middle layer. The axis axis provides the name of the module, with depth increasing from right to left. In each panel, green, blue and orange bars represent the estimated rank of gradients, activations and deltas respectively. Black vertical lines on a bar indicate the standard error in the mean estimated rank across folds and model seeds.

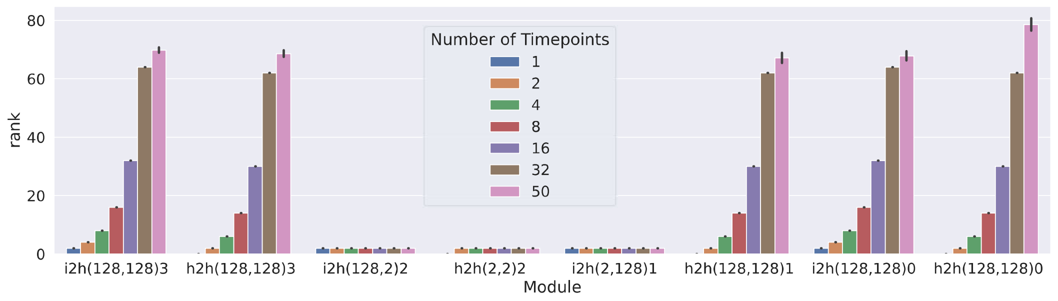

Figure 2: For a 3-layer Elman-Cell RNN, we show how mean rank of gradients, activation, and deltas change with respect to the number of timepoints used in truncated BPTT. The x axis groups particular modules, with depth increasing from right to left. Each colored bar shows the mean estimated rank over multiple seeds and folds using a different sequence length for truncated BPTT.

5 Empirical Results

5.1 Numerical Verification

Hypothesis 1: *Bottleneck layers reduce gradient rank throughout linear networks.* The bound computed in equation 2 suggests that reducing the size of a particular layer in a fully-connected network will reduce gradients throughout that network, regardless of the size of the input activations or adjoints computed at those layers. In figure 1, we provide a demonstration of this phenomenon in a simple fully-connected network used for reconstructing gaussian mixture variables. FIGURE1left In figure 1, we provide the numerical estimate of the gradient rank at each in a 3-layer network with each layer having the same dimension as the input ($d = 128$). FIGURE1right In figure 1, we provide the same rank estimates withe the middle layer being adjusted to contain only 16 neurons. We clearly see that our bound in equation 2 holds, and we can see that the two layers preceding the bottleneck have their rank bounded by the adjoint, and the following layers are bounded by activation rank.

Hypothesis 2: *Parameter-Sharing such as in BPTT restores gradient rank according to the number of points in the accumulated sequence* In §3.2 we discussed how parameter-tying restores gradient rank in models which accumulate gradients over sequence, such as RNNs using BPTT (sec 3.2.1) or CNNs (sec 3.2.2) accumulating over an image. Indeed, the number of points over which back-propagation is performed will affect how much of the gradient rank is restored in the presence of a bottleneck. In figure 2, we demonstrate the gradient rank in an 3-layer Elman-Cell RNN [8] trained to reconstruct high-dimensional, sinusoidal signals. We introduce a severe bottleneck in the second RNN, constraining its hidden units to 2, with the other RNNs having 128 hidden units each. We demonstrate how the introduced gradient bottleneck is ameliorated in the adjacent layers according to the number of timepoints included in truncated BPTT over the sequence. With a maximum of 50 timepoints, the bottleneck can at most be restored to a rank of 100.

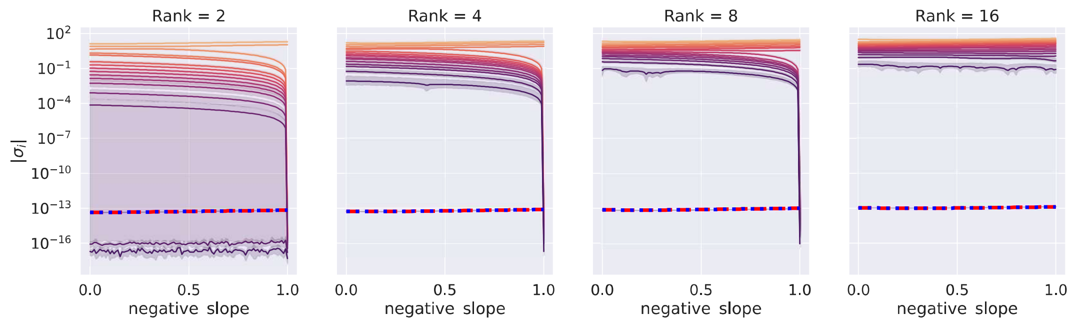

Figure 3: A numerical verification of the derived boundary over which a given eigenvalue computed on a Leaky-ReLU activation σ_k will cease to contribute to the rank. In each panel, we plot how the change in estimated singular values as solid curves, with color corresponding to order of initial magnitude. We plot the rank boundary as a function of estimated largest eigenvalue as a red dotted line, and the rank boundary using equation 6 with a blue dotted line.

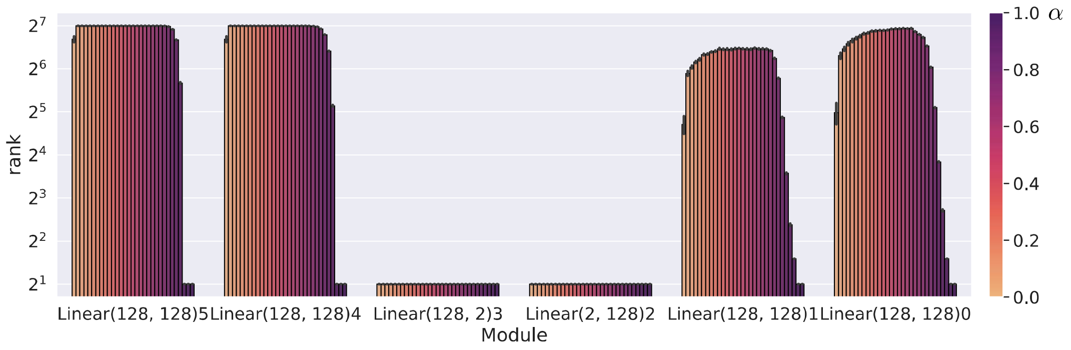

Figure 4: For a 5-layer (6 weight) FC network with Leaky-ReLU activations, we show how mean rank of gradients, activation, and deltas change with respect to the negative slope α of the noninearity. Layer sizes are plotted on the x axis with the depth increasing from left to right. We enforce a bottleneck of 2 neurons in the central layer. For each module, we estimate the rank and provide a colorbar corresponding to the level of nonlinearity increasing in the range of [0,1].

Hypothesis 3:*Using the derivation in §3.3, we can compute the bound over which an estimated singular value will contribute to the rank, without computing the eigenvalues themselves* One of the primary empirical upshots of the theoretical work in §3.3.1 is that using only the singular values of the underlying linearity and a count of how samples contribute to particular areas of the domain of our nonlinearity, we can compute the bound over which singular values will cease to contribute to the estimated rank. In figure 3 we construct a low-rank random variable by computing the matrix product of two low-rank gaussian variables. We then compute the estimated eigenvalues after applying a Leaky-ReLU nonlinearity. We then estimate the bound to rank contribution first by computing it using the maximum estimated singular value, and then using the boundary we derived in equation 6 which does not require singular values. This empirical demonstration indicates that this derived bound is numerically equivalent to the original bound, and that we can exactly compute the threshold of rank collapse without using singular values computed after applying the nonlinearity.

Hypothesis 4:*The negative slope of the leaky-ReLU activation is related to the amount of gradient rank restored.* Although our theoretical analysis in §3.3.1 was given primarily in terms of how leaky-ReLU contributes to the rank of the input activations, it stands to reason that the resulting product of activations and adjoint variables would be affected by the negative slope as well. In figure 4, we implement a 3-layer fully-connected network for reconstruction of Gaussian variables, and we compute the rank of the gradients changing as a function of the slope of the Leaky-ReLU activation. We use a network with 128 neurons at the input and output layers and a 2-neuron bottleneck layer. Our results indicate that pure ReLU activations do not fully restore gradient rank in either direction of back-propagation; however, the negative slope does increase estimated rank close to full-linearity. As the activations approach linearity, the rank returns to the gradient bottleneck.

6 Discussion

Our theoretical bound on gradient rank in linear networks provides a number of startling insights. As we empirically demonstrate in figure 1, linear networks with bottleneck layers are constrained to low-rank learning dynamics, where the rank cannot exceed that of the smallest number of neurons in the network. Beyond the somewhat artificial introduction of bottlenecked layers, these constraints also emerge when dealing with low-dimensional input spaces (even when they are embedded in higher-dimensional spaces like in figure 5). Perhaps more startling is the realization that in supervised learning tasks with relatively few output variables (such as only 10 neurons for 10 classes in Cifar10/MNIST, which can be seen in the rank of the linear classifier in figure 6), the gradient rank throughout the entire network will not exceed that number of variables. When using linear activations, this finding suggests that the ramifications of neural collapse to input layers beyond the terminal linear classifier, with Neural Collapse playing a role at *all* phases of training, not just during the terminal phase. Since our analysis in this work is on architectural constraints which will affect dynamics during *all* training, further work studying the actual gradient dynamics and their effect on the weights during training is needed to fully flesh-out the connection to neural collapse.

Through our analysis of linear networks, we also provide an explanation for how parameter-tying mitigates the tight constraints on rank which may emerge due to bottlenecks. In figures 3 and 4 our empirical verification demonstrates that when low-rank gradients computed at each point in the sequence, rank may be partially restored; however, the level to which rank is restored depends on the length of the sequence. The implication for networks with parameter tying is that aggressively truncated sequences in parameter-tied networks will still be affected by low-rank gradient dynamics.

Our theoretical result identifying how to compute the numerical boundary on rank for leaky-relu networks provides a novel theoretical extension to how nonlinear activations can affect learning dynamics. Additionally, the ability to control the negative slope of the leaky-relu activations allows us to demonstrate how numerical precision can affect the bounds. At the end of our analysis; however, we are left with a boundary that is highly data-dependent, and as we show in figure 4 even fully nonlinear ReLU activations may lower the numerical estimation of rank. This remaining data-dependence in our theory suggests that there is only so much insight to be gained from architectural choice alone, and future work will require analysis of how particular input and output spaces may impose particular boundaries or dynamics on gradient rank. This input/output analysis may also provide deeper insights and tighter bounds the affects of nonlinear activations and parameter tying in particular, with highly correlated or sparse input and output spaces potentially affecting bounds and dynamics.

One limitation of our analysis in this work is that we have purposely avoided relating the emergence of low-rank phenomenon to model performance on particular tasks. Our reason for shying away from this discussion is that model performance is closely related to dynamics throughout the entire training phase, and our theoretical results apply to networks at *any* phase of training, and as such are agnostic to whether a model performs well or not. Our work here provides ample groundwork for analyzing the dynamics within our derived boundaries, and so we leave the connection of gradient rank collapse and performance as future work which can focus on correspondences between collapse and dynamics.

7 Conclusion

In this work, we have presented a theoretical analysis of gradient rank in linear and leaky-ReLU networks. Specifically, we have shown that intuitive bounds on gradient rank emerge as direct consequences of a number of architectural choices such as bottleneck layers, parameter tying, and level of linearity in hidden activations. Our empirical verification and demonstration illustrate the computed bounds in action on numerical and real data sets. The bounds we provide connect simple principles of architectural design with gradient dynamics, carving out the possible space in which those dynamics may emerge. Our work thus serves as a groundwork for continued study of the dynamics of neural spectrum collapse and gradient dynamics, not only in linear classifiers, but in many classes of network architecture.

References

[1] ABADI, M., BARHAM, P., CHEN, J., CHEN, Z., DAVIS, A., DEAN, J., DEVIN, M., GHEMAWAT, S., IRVING, G., ISARD, M., ET AL. Tensorflow: a system for large-scale machine learning. In *Osdi* (2016), vol. 16, Savannah, GA, USA, pp. 265–283.

[2] ADVANI, M. S., SAXE, A. M., AND SOMPOLINSKY, H. High-dimensional dynamics of generalization error in neural networks. *Neural Networks 132* (2020), 428–446.

[3] BAKER, B. T., KHANAL, A., CALHOUN, V. D., PEARLMUTTER, B., AND PLIS, S. M. Peering beyond the gradient veil with distributed auto differentiation. *arXiv preprint arXiv:2102.09631* (2021).

[4] DANESHMAND, H., KOHLER, J., BACH, F., HOFMANN, T., AND LUCCHI, A. Batch normalization provably avoids ranks collapse for randomly initialised deep networks. *Advances in Neural Information Processing Systems 33* (2020), 18387–18398.

[5] DENG, J., DONG, W., SOCHER, R., LI, L.-J., LI, K., AND FEI-FEI, L. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition* (2009), Ieee, pp. 248–255.

[6] DEVLIN, J., CHANG, M.-W., LEE, K., AND TOUTANOVA, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[7] DITTMER, S., KING, E. J., AND MAASS, P. Singular values for ReLU layers. *IEEE Transactions on Neural Networks and Learning Systems 31*, 9 (2019), 3594–3605.

[8] ELMAN, J. L. Finding structure in time. *Cognitive science 14*, 2 (1990), 179–211.

[9] GALANTI, T., GYÖRGY, A., AND HUTTER, M. On the role of neural collapse in transfer learning. *arXiv preprint arXiv:2112.15121* (2021).

[10] GOLDT, S., ADVANI, M., SAXE, A. M., KRZAKALA, F., AND ZDEBOROVÁ, L. Dynamics of stochastic gradient descent for two-layer neural networks in the teacher-student setup. *Advances in neural information processing systems 32* (2019).

[11] GOLDT, S., ADVANI, M. S., SAXE, A. M., KRZAKALA, F., AND ZDEBOROVÁ, L. Generalisation dynamics of online learning in over-parameterised neural networks. *arXiv preprint arXiv:1901.09085* (2019).

[12] HAN, X., PAPYAN, V., AND DONOHO, D. L. Neural collapse under mse loss: Proximity to and dynamics on the central path. *arXiv preprint arXiv:2106.02073* (2021).

[13] HE, K., ZHANG, X., REN, S., AND SUN, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2016), pp. 770–778.

[14] KOTHAPALLI, V. Neural collapse: A review on modelling principles and generalization. *arXiv preprint arXiv:2206.04041* (2023).

[15] KRIZHEVSKY, A., HINTON, G., ET AL. Learning multiple layers of features from tiny images.

[16] LAMPLE, G., AND CONNEAU, A. Cross-lingual language model pretraining. *arXiv preprint arXiv:1901.07291* (2019).

[17] LU, J., AND STEINERBERGER, S. Neural collapse with cross-entropy loss. *arXiv preprint arXiv:2012.08465* (2020).

[18] MERITY, S. The wikitext long term dependency language modeling dataset. *Salesforce Metamind 9* (2016).

[19] MIXON, D. G., PARSHALL, H., AND PI, J. Neural collapse with unconstrained features. *arXiv preprint arXiv:2011.11619* (2020).

[20] PAPYAN, V. The full spectrum of deepnet hessians at scale: Dynamics with sgd training and sample size. *arXiv preprint arXiv:1811.07062* (2018).

[21] Papyan, V., Han, X., and Donoho, D. L. Prevalence of neural collapse during the terminal phase of deep learning training. *Proceedings of the National Academy of Sciences 117*, 40 (2020), 24652–24663.

[22] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems 32* (2019).

[23] Saxe, A., Sodhani, S., and Lewallen, S. J. The neural race reduction: dynamics of abstraction in gated networks. In *International Conference on Machine Learning* (2022), PMLR, pp. 19287–19309.

[24] Saxe, A. M., McClelland, J. L., and Ganguli, S. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv preprint arXiv:1312.6120* (2013).

[25] Saxe, A. M., McClelland, J. L., and Ganguli, S. A mathematical theory of semantic development in deep neural networks. *Proceedings of the National Academy of Sciences 116*, 23 (2019), 11537–11546.

[26] Simonyan, K., and Zisserman, A. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).

[27] Thrampoulidis, C., Kini, G. R., Vakilian, V., and Behnia, T. Imbalance trouble: Revisiting neural-collapse geometry. *Advances in Neural Information Processing Systems 35* (2022), 27225–27238.

[28] Tirer, T., and Bruna, J. Extended unconstrained features model for exploring deep neural collapse. In *International Conference on Machine Learning* (2022), PMLR, pp. 21478–21505.

[29] Vogels, T., Karimireddy, S. P., and Jaggi, M. Powersgd: Practical low-rank gradient compression for distributed optimization. *Advances in Neural Information Processing Systems 32* (2019).

[30] Yaras, C., Wang, P., Zhu, Z., Balzano, L., and Qu, Q. Neural collapse with normalized features: A geometric analysis over the riemannian manifold. *arXiv preprint arXiv:2209.09211* (2022).

[31] Zhan, X. Inequalities for the singular values of hadamard products. *SIAM Journal on Matrix Analysis and Applications 18*, 4 (1997), 1093–1095.

[32] Zhou, J., Li, X., Ding, T., You, C., Qu, Q., and Zhu, Z. On the optimization landscape of neural collapse under mse loss: Global optimality with unconstrained features. In *International Conference on Machine Learning* (2022), PMLR, pp. 27179–27202.

[33] Zhou, J., You, C., Li, X., Liu, K., Liu, S., Qu, Q., and Zhu, Z. Are all losses created equal: A neural collapse perspective. *arXiv preprint arXiv:2210.02192* (2022).

[34] Zhu, Z., Ding, T., Zhou, J., Li, X., You, C., Sulam, J., and Qu, Q. A geometric analysis of neural collapse with unconstrained features. *Advances in Neural Information Processing Systems 34* (2021), 29820–29834.