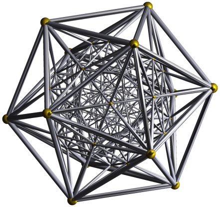
On the Emergence of Low-Dim Invariant Subspace in Gradient Descent for Learning Deep Linear Networks

Qing Qu

EECS, University of Michigan

September 7, 2023



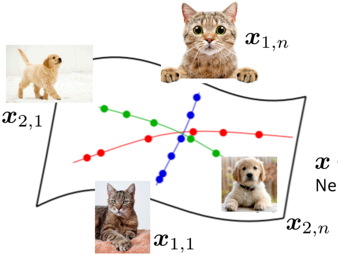
Multi-Class Image Classification Problem

- **Goal:** Learn a deep network predictor from a labelled training dataset $\{(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)});\ i = 1, \cdots, n\}$.

[1] If not, we can use data augmentation to make them balanced

Multi-Class Image Classification Problem

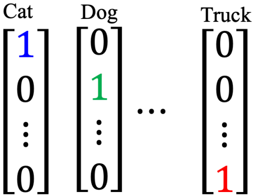
- **Goal:** Learn a deep network predictor from a labelled training dataset $\{(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)});\ i = 1, \cdots, n\}$.
- **Training Labels:** $k = 1, \ldots, K$
  - $K = 10$ classes (MNIST, CIFAR10, etc)
  - $K = 1000$ classes (ImageNet)



Data in the input space

$\boldsymbol{x} \longrightarrow \boldsymbol{y}$

Neural network

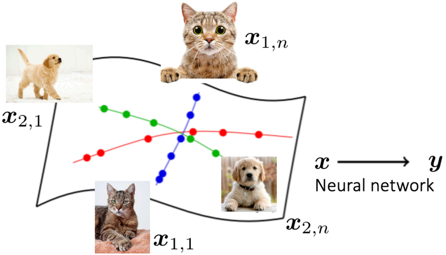


One-hot labeling vectors in $\mathbb{R}^K$

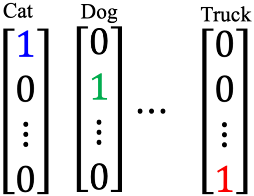
[1]If not, we can use data augmentation to make them balanced

Multi-Class Image Classification Problem

- **Goal:** Learn a deep network predictor from a labelled training dataset $\{(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)});\ i = 1, \cdots, n\}$.
- **Training Labels:** $k = 1, \ldots, K$
  - $K = 10$ classes (MNIST, CIFAR10, etc)
  - $K = 1000$ classes (ImageNet)



Data in the input space



One-hot labeling vectors in $\mathbb{R}^K$

- For simplicity, we assume **balanced** dataset where each class has n training samples.[1]

[1] If not, we can use data augmentation to make them balanced

Deep Neural Network Classifiers

- **A vanilla multi-layer perception (MLP) network:**

$$f_{\boldsymbol{\Theta}}(\boldsymbol{x}) \;=\; \underbrace{\boldsymbol{W}_L}_{\textbf{linear classifer } \boldsymbol{W}} \; \underbrace{\sigma\left(\boldsymbol{W}_{L-1} \cdots \sigma(\boldsymbol{W}_1 \boldsymbol{x} + \boldsymbol{b}_1) + \boldsymbol{b}_{L-1}\right)}_{\textbf{feature } \phi_{\boldsymbol{\theta}}(\boldsymbol{x}) =: \boldsymbol{h}} + \boldsymbol{b}_L$$

- **Features of each layer:**

$$\boldsymbol{z}^l = \sigma\left(\boldsymbol{W}_{l-1} \cdots \sigma(\boldsymbol{W}_1 \boldsymbol{x} + \boldsymbol{b}_1) + \boldsymbol{b}_{l-1}\right), l = 1, \cdots, L-1$$

Deep Neural Network Classifiers

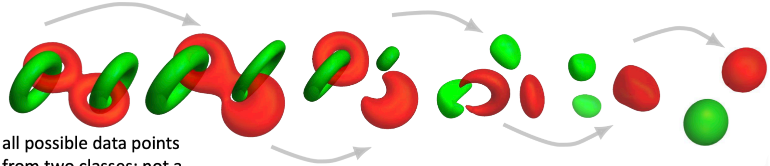
- **A vanilla multi-layer perception (MLP) network:**

$$f_{\Theta}(\boldsymbol{x}) = \underbrace{\boldsymbol{W}_L}_{\text{linear classifer } \boldsymbol{W}} \underbrace{\sigma\left(\boldsymbol{W}_{L-1} \cdots \sigma(\boldsymbol{W}_1 \boldsymbol{x} + \boldsymbol{b}_1) + \boldsymbol{b}_{L-1}\right)}_{\text{feature } \phi_{\boldsymbol{\theta}}(\boldsymbol{x}) =: \boldsymbol{h}} + \boldsymbol{b}_L$$

- **Features of each layer:**

$$\boldsymbol{z}^l = \sigma\left(\boldsymbol{W}_{l-1} \cdots \sigma(\boldsymbol{W}_1 \boldsymbol{x} + \boldsymbol{b}_1) + \boldsymbol{b}_{l-1}\right), l = 1, \cdots, L-1$$

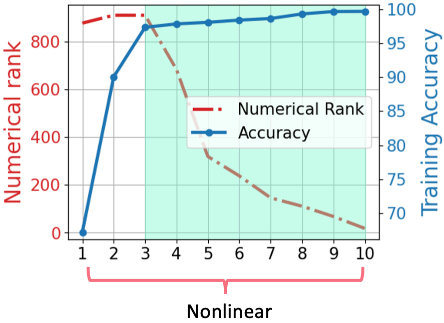
- **Progressive linear separation through nonlinear layers:**



all possible data points from two classes; not a single input!

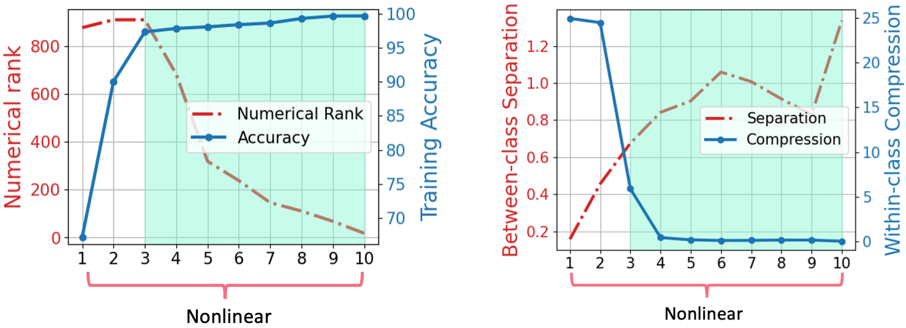
Feature Compression & Separation in Deep Networks

Training a 10-layer nonlinear MLP network on CIFAR-10



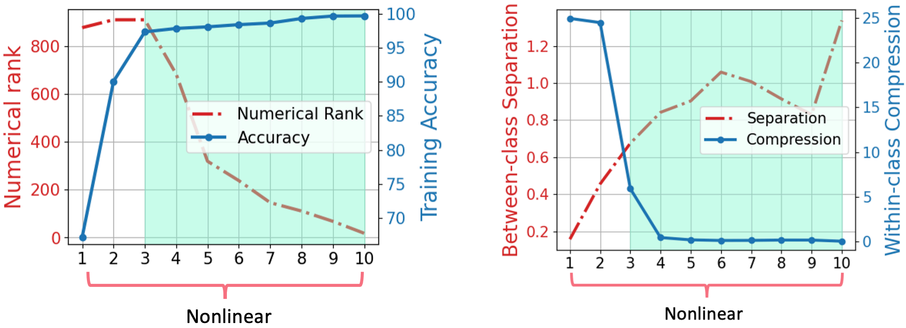
Feature Compression & Separation in Deep Networks

Training a 10-layer nonlinear MLP network on CIFAR-10



Feature Compression & Separation in Deep Networks

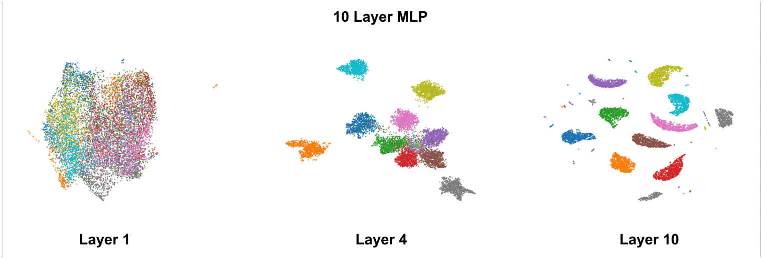
Training a 10-layer nonlinear MLP network on CIFAR-10



Progressive feature “compression” and “linear separation” from shallow to deep layers.

Feature Compression & Separation in Deep Networks

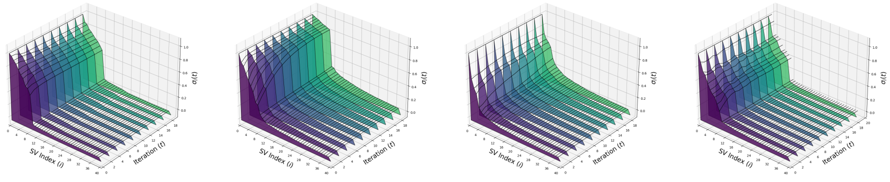
Training a 10-layer multi-layer perception (MLP) nonlinear network for classification problems (CIFAR-10)



Progressive feature "compression" and "linear separation" from shallow to deep layers.

Implication I: Invariant Subspaces of in Deeper Layers

We track the learning dynamics of singular values in the penultimate layer a wide range of models (linear model, MLP, toy ViT, ViT-base):

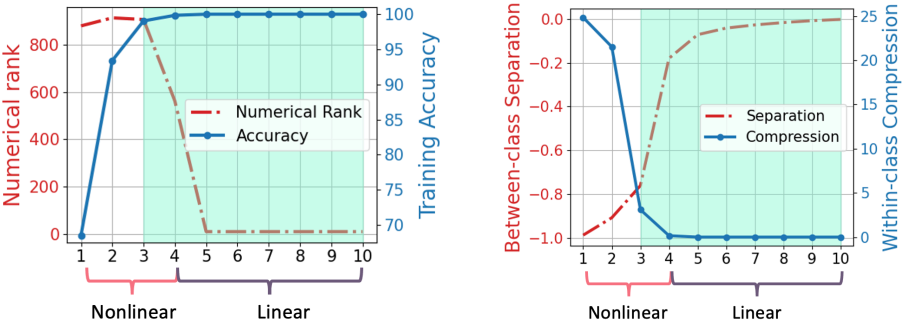


Linear Model | MLP | Toy ViT | ViT-B

> In the deeper layers, feature learning *only* happens in a low-dimensional invariant subspace of the weight matrices.

Implication II: Linear Separability in Deeper Layers

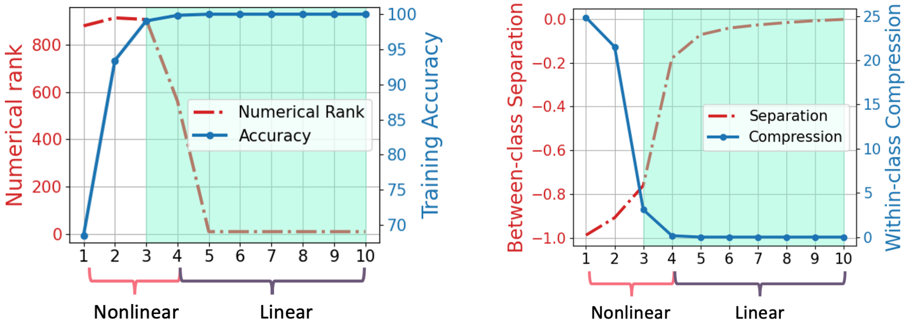
Training a hybrid (4-layer MLP + 6-layer linear) network on CIFAR-10



Progressive “compression” and “linear separation” from shallow to deep layers.

Implication II: Linear Separability in Deeper Layers

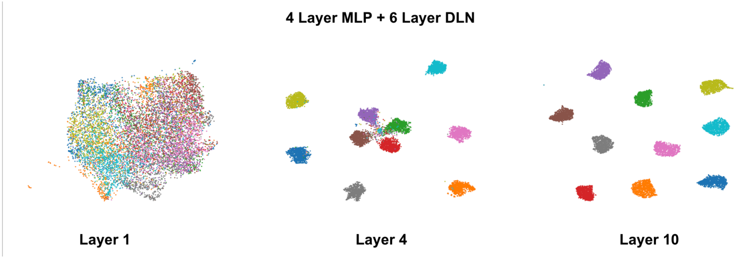
Training a hybrid (4-layer MLP + 6-layer linear) network on CIFAR-10



Progressive "compression" and "linear separation" from shallow to deep layers.

Implication II: Linear Separability in Deeper Layers

Training a hybrid (4-layer MLP + 6-layer linear) network on CIFAR-10



Progressive "compression" and "linear separation" from shallow to deep layers.

Study of Deep Linear Networks?

Deep linear network (DLN):

$$f_{\mathbf{\Theta}}(\boldsymbol{x}) \;:=\; \boldsymbol{W}_L \cdots \boldsymbol{W}_1 \boldsymbol{x} \;=\; \boldsymbol{W}_{L:1} \boldsymbol{x},$$

has been often used as prototypes for studying nonlinear counterparts:

Study of Deep Linear Networks?

Deep linear network (DLN):

$$f_{\boldsymbol{\Theta}}(\boldsymbol{x}) \;:=\; \boldsymbol{W}_L \cdots \boldsymbol{W}_1 \boldsymbol{x} \;=\; \boldsymbol{W}_{L:1} \boldsymbol{x},$$

has been often used as prototypes for studying nonlinear counterparts:

- It possess **similar linear separability** in deeper layers as nonlinear networks;

Study of Deep Linear Networks?

Deep linear network (DLN):

$$f_{\boldsymbol{\Theta}}(\boldsymbol{x}) \;:=\; \boldsymbol{W}_L \cdots \boldsymbol{W}_1 \boldsymbol{x} \;=\; \boldsymbol{W}_{L:1}\boldsymbol{x},$$

has been often used as prototypes for studying nonlinear counterparts:

- It possess **similar linear separability** in deeper layers as nonlinear networks;
- The features possess **similar compression and separation** across layers;

Study of Deep Linear Networks?

Deep linear network (DLN):

$$f_{\boldsymbol{\Theta}}(\boldsymbol{x}) \; := \; \boldsymbol{W}_L \cdots \boldsymbol{W}_1 \boldsymbol{x} \; = \; \boldsymbol{W}_{L:1} \boldsymbol{x},$$

has been often used as prototypes for studying nonlinear counterparts:

- It possess **similar linear separability** in deeper layers as nonlinear networks;
- The features possess **similar compression and separation** across layers;
- The weights possess **similar low-rank structures** throughout training.

Study of Deep Linear Networks?

Study of the training DLNs

$$\min_{\boldsymbol{\Theta}} \ \ell(\boldsymbol{\Theta}) = \frac{1}{2}\sum_{i=1}^{N} \|f_{\boldsymbol{\Theta}}(\boldsymbol{x}_i) - \boldsymbol{y}_i\|_F^2 = \frac{1}{2}\|\boldsymbol{W}_{L:1}\boldsymbol{X} - \boldsymbol{Y}\|_F^2 .$$

could be highly nontrivial:

Study of Deep Linear Networks?

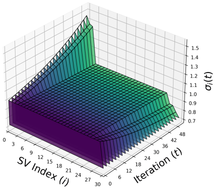
Study of the training DLNs

$$\min_{\boldsymbol{\Theta}} \ \ell(\boldsymbol{\Theta}) = \frac{1}{2}\sum_{i=1}^{N} \|f_{\boldsymbol{\Theta}}(\boldsymbol{x}_i) - \boldsymbol{y}_i\|_F^2 = \frac{1}{2}\|\boldsymbol{W}_{L:1}\boldsymbol{X} - \boldsymbol{Y}\|_F^2 .$$

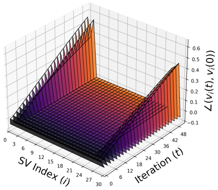
could be highly nontrivial:

- The loss landscape is highly **nonconvex**, with many saddle points;
- It is overparameterized, with **infinitely many** local solutions;
- The gradient descent learning dynamics could be highly **nonlinear**.

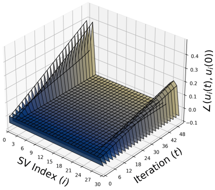
Main Results



Singular Values



Right Singular Vectors

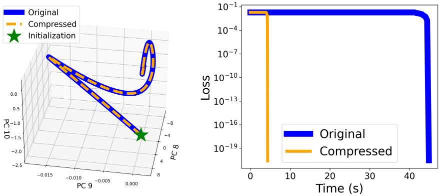


Left Singular Vectors

SCAN ME

Throughout training of deep linear networks, the gradient descent (GD) dynamics possesses certain parsimonious structures.

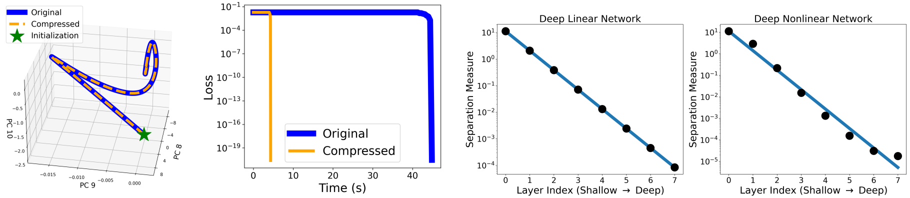
Main Results



The parsimonious structures in GD dynamics leads to

- **Efficient low-rank training and network compression**

Main Results



The parsimonious structures in GD dynamics leads to

- **Efficient low-rank training and network compression**
- **Better understandings of hierarchical representations**

Outline

Deep Linear Networks

- **Training data** $\{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^N \subset \mathbb{R}^{d_x} \times \mathbb{R}^{d_y}$ with

$$\boldsymbol{X} = [\boldsymbol{x}_1 \ \boldsymbol{x}_2 \ \dots \ \boldsymbol{x}_N] \in \mathbb{R}^{d_x \times N}, \quad \boldsymbol{Y} = [\boldsymbol{y}_1 \ \boldsymbol{y}_2 \ \dots \ \boldsymbol{y}_N] \in \mathbb{R}^{d_y \times N}$$

- **Deep linear network** (DLN):

$$f_{\boldsymbol{\Theta}}(\boldsymbol{x}) \ := \ \boldsymbol{W}_L \cdots \boldsymbol{W}_1 \boldsymbol{x} \ = \ \boldsymbol{W}_{L:1} \boldsymbol{x},$$

  where $\boldsymbol{W}_l \in \mathbb{R}^{d_l \times d_{l-1}}$ and $\boldsymbol{\Theta} = \{\boldsymbol{W}_l\}_{l=1}^L$.

- **Loss function**:

$$\min_{\boldsymbol{\Theta}} \ \ell(\boldsymbol{\Theta}) = \frac{1}{2} \sum_{i=1}^N \| f_{\boldsymbol{\Theta}}(\boldsymbol{x}_i) - \boldsymbol{y}_i \|_F^2 = \frac{1}{2} \| \boldsymbol{W}_{L:1} \boldsymbol{X} - \boldsymbol{Y} \|_F^2 .$$

Training DLNs via Gradient Descent (GD)

- **Orthogonal initialization.** We use ε-scale orthogonal matrices for some $\varepsilon > 0$, with

$$\boldsymbol{W}_l^\top(0)\boldsymbol{W}_l(0) = \varepsilon^2\boldsymbol{I} \quad \text{or} \quad \boldsymbol{W}_l(0)\boldsymbol{W}_l^\top(0) = \varepsilon^2\boldsymbol{I}, \quad \forall l \in [L],$$

depending on the size of $\boldsymbol{W}_l$.

Training DLNs via Gradient Descent (GD)

- **Orthogonal initialization.** We use ε-scale orthogonal matrices for some $\varepsilon > 0$, with

$$\boldsymbol{W}_l^\top(0)\boldsymbol{W}_l(0) = \varepsilon^2 \boldsymbol{I} \quad \text{or} \quad \boldsymbol{W}_l(0)\boldsymbol{W}_l^\top(0) = \varepsilon^2 \boldsymbol{I}, \quad \forall l \in [L],$$

depending on the size of $\boldsymbol{W}_l$.

- **Learning dynamics of GD.** We update all weights via GD for $t = 1, 2, \ldots$ as

$$\boldsymbol{W}_l(t) = (1 - \eta\lambda)\boldsymbol{W}_l(t-1) - \eta \nabla_{\boldsymbol{W}_l} \ell(\boldsymbol{\Theta}(t-1)), \;\; \forall\, l \in [L],$$

where $\eta > 0$ is the learning rate and $\lambda \geq 0$ controls weight decay.

Training DLNs via Gradient Descent (GD)

We study the GD iterates for training DLNs under the following assumptions:

- The weight matrices are *square* except the last layer, i.e., $d_x = d_1 = d_2 = \cdots = d_{L-1} = d$ for some $d \in \mathbb{N}_+$.
- The input data is *whitened* in the sense that $\boldsymbol{X}\boldsymbol{X}^\top = \boldsymbol{I}_{d_x}$.[2]
- The cross correlation matrix $\boldsymbol{Y}\boldsymbol{X}^\top$ has certain *low-dimensional structures* (e.g., low-rank or wide matrix).

[2] For any full rank $\boldsymbol{X} \in \mathbb{R}^{d_x \times N}$ with $N \geq d_x$, whitened data can always be obtained with a data pre-processing step such as preconditioning.

Training DLNs via Gradient Descent (GD)

We study the GD iterates for training DLNs under the following assumptions:

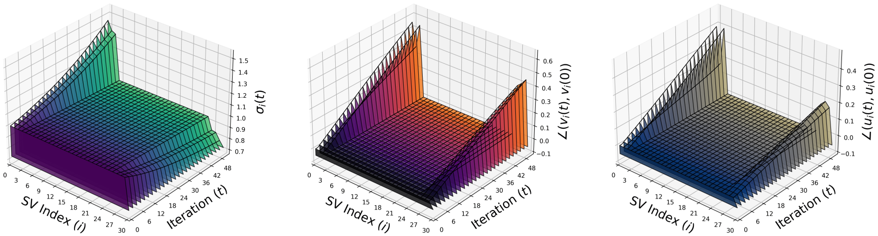
- The weight matrices are *square* except the last layer, i.e., $d_x = d_1 = d_2 = \cdots = d_{L-1} = d$ for some $d \in \mathbb{N}_+$.
- The input data is *whitened* in the sense that $\boldsymbol{X}\boldsymbol{X}^\top = \boldsymbol{I}_{d_x}$.[2]
- The cross correlation matrix $\boldsymbol{Y}\boldsymbol{X}^\top$ has certain *low-dimensional structures* (e.g., low-rank or wide matrix).

> Throughout training of deep networks, the gradient descent leads to certain parsimonious structures in the weight matrices.

[2] For any full rank $\boldsymbol{X} \in \mathbb{R}^{d_x \times N}$ with $N \geq d_x$, whitened data can always be obtained with a data pre-processing step such as preconditioning.

The Evolution of Singular Spaces in GD Iterates for DLNs

We train a $L = 3$ layer DLN with $d_x = d_y = 30$ and $r := \operatorname{rank}(\boldsymbol{Y}) = 3$.

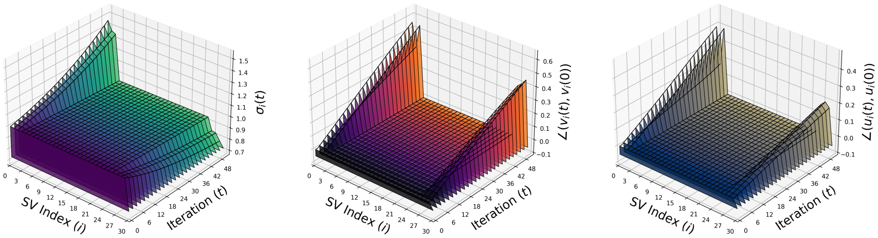


Singular Values — Right Singular Vectors — Left Singular Vectors

Figure: **Evolution of SVD of the weight matrix $\boldsymbol{W}_1(t) = \boldsymbol{U}_1(t)\boldsymbol{\Sigma}_1(t)\boldsymbol{V}_1(t)^\top$.**

The Evolution of Singular Spaces in GD Iterates for DLNs

We train a $L = 3$ layer DLN with $d_x = d_y = 30$ and $r := \mathrm{rank}(\boldsymbol{Y}) = 3$.



Singular Values | Right Singular Vectors | Left Singular Vectors

Figure: **Evolution of SVD of the weight matrix $\boldsymbol{W}_1(t) = \boldsymbol{U}_1(t)\boldsymbol{\Sigma}_1(t)\boldsymbol{V}_1(t)^\top$.**

- **Left:** the evolution of singular values $\sigma_{1i}(t)$ throughout training $t \geq 0$;

The Evolution of Singular Spaces in GD Iterates for DLNs

We train a $L = 3$ layer DLN with $d_x = d_y = 30$ and $r := \operatorname{rank}(\boldsymbol{Y}) = 3$.

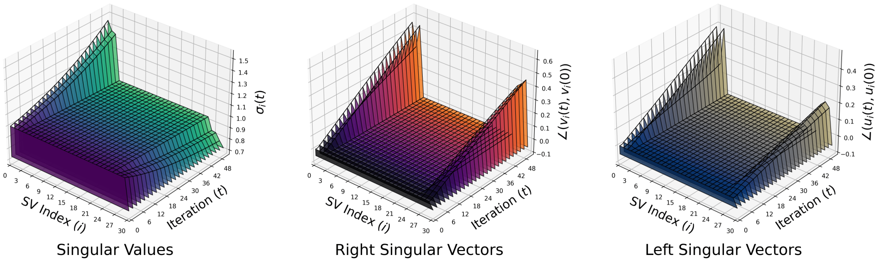
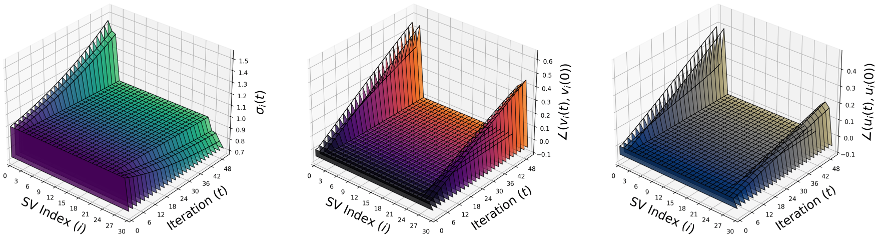


Figure: **Evolution of SVD of the weight matrix $\boldsymbol{W}_1(t) = \boldsymbol{U}_1(t)\boldsymbol{\Sigma}_1(t)\boldsymbol{V}_1(t)^\top$.**

- **Left:** the evolution of singular values $\sigma_{1i}(t)$ throughout training $t \geq 0$;
- **Middle:** the evolution of $\angle(\boldsymbol{v}_{1i}(t), \boldsymbol{v}_{1i}(0))$ throughout training $t \geq 0$;

The Evolution of Singular Spaces in GD Iterates for DLNs

We train a $L = 3$ layer DLN with $d_x = d_y = 30$ and $r := \operatorname{rank}(\boldsymbol{Y}) = 3$.

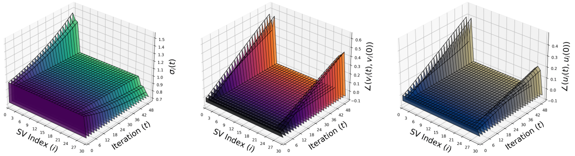


Singular Values   Right Singular Vectors   Left Singular Vectors

Figure: **Evolution of SVD of the weight matrix** $\boldsymbol{W}_1(t) = \boldsymbol{U}_1(t)\boldsymbol{\Sigma}_1(t)\boldsymbol{V}_1(t)^\top$**.**

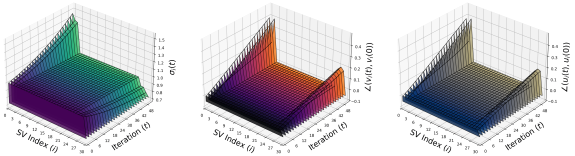
- **Left:** the evolution of singular values $\sigma_{1i}(t)$ throughout training $t \geq 0$;
- **Middle:** the evolution of $\angle(\boldsymbol{v}_{1i}(t), \boldsymbol{v}_{1i}(0))$ throughout training $t \geq 0$;
- **Right:** the evolution of $\angle(\boldsymbol{u}_{1i}(t), \boldsymbol{u}_{1i}(0))$ throughout training $t \geq 0$.

Layer 1



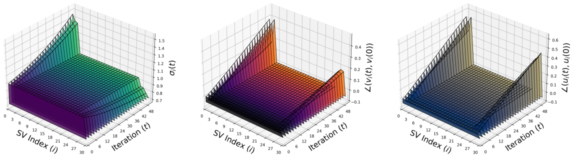
Singular Values | Right Singular Vectors | Left Singular Vectors

Layer 2



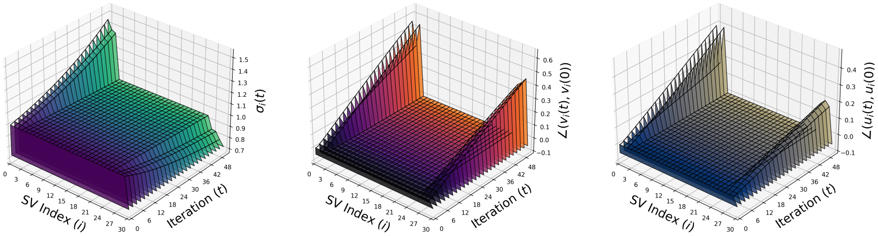
Singular Values | Right Singular Vectors | Left Singular Vectors

Layer 3



Singular Values | Right Singular Vectors | Left Singular Vectors

The Evolution of Singular Spaces in GD Iterates for DLNs



Singular Values | Right Singular Vectors | Left Singular Vectors

Figure: **Evolution of SVD of the weight matrix $\boldsymbol{W}_1(t) = \boldsymbol{U}_1(t)\boldsymbol{\Sigma}_1(t)\boldsymbol{V}_1(t)^\top$.**

> The GD learning process takes place only within a **minimal invariant subspace** of each weight matrix, while the remaining singular subspaces stay **unaffected** throughout training.

The Law of Parsimony in GD

Theorem (Yaras et al.'23)

Suppose we train an L-layer DLN $f_{\mathbf{\Theta}}(\cdot)$ on $(\boldsymbol{X}, \boldsymbol{Y})$ via GD, the iterates $\{\boldsymbol{W}_l(t)\}_{l=1}^L$ for all $t \geq 0$ satisfy the following:

- **Case 1:** *Suppose $\boldsymbol{Y}\boldsymbol{X}^\top \in \mathbb{R}^{d_y \times d_x}$ is of rank $r \in \mathbb{N}_+$ with $d_y = d_x$, and $m = d_x - 2r > 0$. Then $\exists \{\boldsymbol{U}_l\}_{l=1}^L \subseteq \mathcal{O}^d$ and $\{\boldsymbol{V}_l\}_{l=1}^L \subseteq \mathcal{O}^d$ satisfying $\boldsymbol{V}_{l+1} = \boldsymbol{U}_l$ for all $l \in [L-1]$, such that $\boldsymbol{W}_l(t)$ admits the following decomposition*

$$\boldsymbol{W}_l(t) = \boldsymbol{U}_l \begin{bmatrix} \widetilde{\boldsymbol{W}}_l(t) & \mathbf{0} \\ \mathbf{0} & \rho(t)\boldsymbol{I}_m \end{bmatrix} \boldsymbol{V}_l^\top, \quad \forall l \in [L-1],\ t \geq 0,$$

where $\widetilde{\boldsymbol{W}}_l(t) \in \mathbb{R}^{2r \times 2r}$ for all $l \in [L-1]$ with $\widetilde{\boldsymbol{W}}_l(0) = \varepsilon \boldsymbol{I}_{2r}$.

The Law of Parsimony in GD

Theorem (Yaras et al.'23)

Suppose we train an L-layer DLN $f_{\boldsymbol{\Theta}}(\cdot)$ on $(\boldsymbol{X}, \boldsymbol{Y})$ via GD, the iterates $\{\boldsymbol{W}_l(t)\}_{l=1}^L$ for all $t \geq 0$ satisfy the following:

- **Case 1:** *Suppose $\boldsymbol{Y}\boldsymbol{X}^\top \in \mathbb{R}^{d_y \times d_x}$ is of rank $r \in \mathbb{N}_+$ with $d_y = d_x$, and $m = d_x - 2r > 0$. Then $\exists\ \{\boldsymbol{U}_l\}_{l=1}^L \subseteq \mathcal{O}^d$ and $\{\boldsymbol{V}_l\}_{l=1}^L \subseteq \mathcal{O}^d$ satisfying $\boldsymbol{V}_{l+1} = \boldsymbol{U}_l$ for all $l \in [L-1]$, such that $\boldsymbol{W}_l(t)$ admits the following decomposition*

$$\boldsymbol{W}_l(t) = \boldsymbol{U}_l \begin{bmatrix} \widetilde{\boldsymbol{W}}_l(t) & \boldsymbol{0} \\ \boldsymbol{0} & \rho(t)\boldsymbol{I}_m \end{bmatrix} \boldsymbol{V}_l^\top, \quad \forall l \in [L-1],\ t \geq 0,$$

  where $\widetilde{\boldsymbol{W}}_l(t) \in \mathbb{R}^{2r \times 2r}$ for all $l \in [L-1]$ with $\widetilde{\boldsymbol{W}}_l(0) = \varepsilon \boldsymbol{I}_{2r}$.
- **Case 2:** *Suppose $\boldsymbol{Y}\boldsymbol{X}^\top \in \mathbb{R}^{d_y \times d_x}$ with $d_y = r$ and $m := d_x - 2d_y > 0$. Similar results hold with different $\rho(t)$.*

The Law of Parsimony in GD

- **Dynamics of singular values and vectors of weight matrices.** Let $\widetilde{\boldsymbol{W}}_l(t) = \widetilde{\boldsymbol{U}}_l(t)\widetilde{\boldsymbol{\Sigma}}_l(t)\widetilde{\boldsymbol{V}}_l^\top(t)$, we can rewrite our decomposition as

$$\boldsymbol{W}_l(t) = \begin{bmatrix} \boldsymbol{U}_{l,1}\widetilde{\boldsymbol{U}}_l(t) & \boldsymbol{U}_{l,2} \end{bmatrix} \begin{bmatrix} \widetilde{\boldsymbol{\Sigma}}_l(t) & \boldsymbol{0} \\ \boldsymbol{0} & \rho(t)\boldsymbol{I}_m \end{bmatrix} \begin{bmatrix} \boldsymbol{V}_{l,1}\widetilde{\boldsymbol{V}}_l(t) & \boldsymbol{V}_{l,2} \end{bmatrix}^\top,$$

[3] M. Huh et al. The Low-Rank Simplicity Bias in Deep Networks, TMLR'23. https://minyoungg.github.io/overparam/

The Law of Parsimony in GD

- **Dynamics of singular values and vectors of weight matrices.** Let $\widetilde{\boldsymbol{W}}_l(t) = \widetilde{\boldsymbol{U}}_l(t)\widetilde{\boldsymbol{\Sigma}}_l(t)\widetilde{\boldsymbol{V}}_l^\top(t)$, we can rewrite our decomposition as

$$\boldsymbol{W}_l(t) = \begin{bmatrix} \boldsymbol{U}_{l,1}\widetilde{\boldsymbol{U}}_l(t) & \boldsymbol{U}_{l,2} \end{bmatrix} \begin{bmatrix} \widetilde{\boldsymbol{\Sigma}}_l(t) & \boldsymbol{0} \\ \boldsymbol{0} & \rho(t)\boldsymbol{I}_m \end{bmatrix} \begin{bmatrix} \boldsymbol{V}_{l,1}\widetilde{\boldsymbol{V}}_l(t) & \boldsymbol{V}_{l,2} \end{bmatrix}^\top,$$

- **Invariance of subspaces in the weights.** Both $\boldsymbol{U}_{l,2}$ and $\boldsymbol{V}_{l,2}$ of size $d-2r$ are unchanged throughout training. The learning process occurs **only** within **an invariant subspace** of dimension $2r$!

[3]M. Huh et al. The Low-Rank Simplicity Bias in Deep Networks, TMLR'23.
`https://minyoungg.github.io/overparam/`

The Law of Parsimony in GD

- **Dynamics of singular values and vectors of weight matrices.** Let $\widetilde{\boldsymbol{W}}_l(t) = \widetilde{\boldsymbol{U}}_l(t)\widetilde{\boldsymbol{\Sigma}}_l(t)\widetilde{\boldsymbol{V}}_l^\top(t)$, we can rewrite our decomposition as

$$\boldsymbol{W}_l(t) = \begin{bmatrix} \boldsymbol{U}_{l,1}\widetilde{\boldsymbol{U}}_l(t) & \boldsymbol{U}_{l,2} \end{bmatrix} \begin{bmatrix} \widetilde{\boldsymbol{\Sigma}}_l(t) & \boldsymbol{0} \\ \boldsymbol{0} & \rho(t)\boldsymbol{I}_m \end{bmatrix} \begin{bmatrix} \boldsymbol{V}_{l,1}\widetilde{\boldsymbol{V}}_l(t) & \boldsymbol{V}_{l,2} \end{bmatrix}^\top,$$

- **Invariance of subspaces in the weights.** Both $\boldsymbol{U}_{l,2}$ and $\boldsymbol{V}_{l,2}$ of size $d-2r$ are unchanged throughout training. The learning process occurs **only** within **an invariant subspace** of dimension $2r$!

- **Implicit low-rank bias.**[3] As $\lim_{\varepsilon \to 0} \rho(t) = 0$ for all $t \geq 0$, all the weights $\boldsymbol{W}_l(t)$ and the end-to-end matrix $\boldsymbol{W}_{L:1}(t)$ are inherently low-rank (e.g., at most rank $2r$).

[3] M. Huh et al. The Low-Rank Simplicity Bias in Deep Networks, TMLR'23. https://minyoungg.github.io/overparam/

The Evolution of Singular Spaces in More Generic Settings

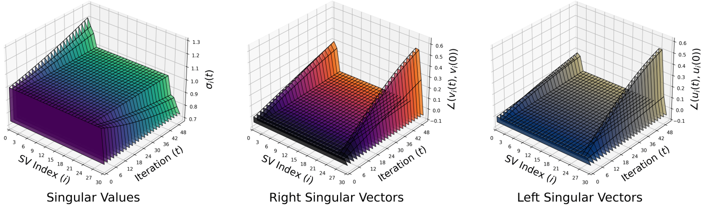
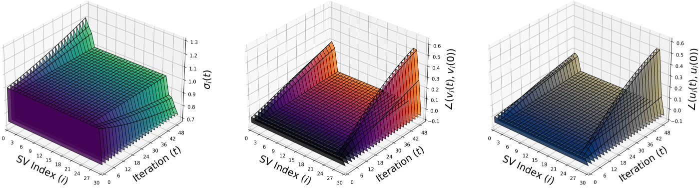


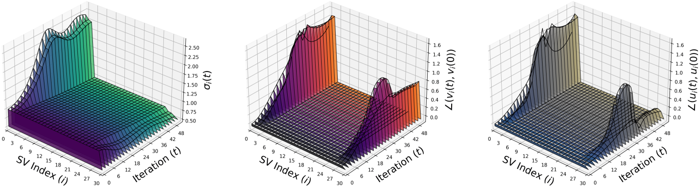
Figure: **Evolution of SVD of weight matrices without whitened data.**

The Evolution of Singular Spaces in More Generic Settings



Singular Values | Right Singular Vectors | Left Singular Vectors

Figure: **Evolution of SVD of weight matrices without whitened data.**



Singular Values | Right Singular Vectors | Left Singular Vectors

Figure: **Evolution of SVD of weight matrices with momentum.**

Outline

Main Message

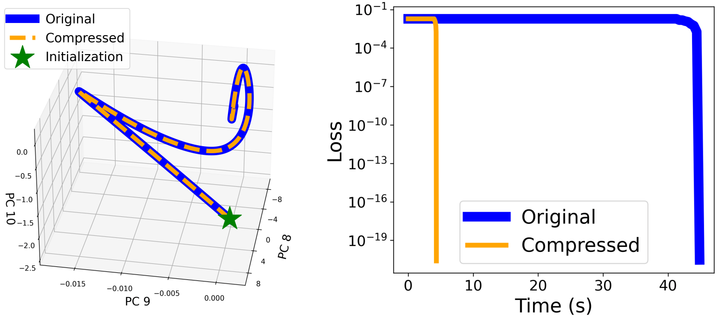


Figure: **Efficient training of deep linear networks.**

The law of parsimony in GD leads to efficient network compression.

Deep Matrix Completion

Consider recovering $\mathbf{\Phi} \in \mathbb{R}^{d \times d}$ with $r := \text{rank}(\mathbf{\Phi}) \ll d$ with minimum number of observation encoded by $\mathbf{\Omega} \in \{0,1\}^{d \times d}$:

$$\min_{\mathbf{\Theta}} \ell_{\text{mc}}(\mathbf{\Theta}) := \frac{1}{2} \|\mathbf{\Omega} \odot (\boldsymbol{W}_{L:1} - \mathbf{\Phi})\|_F^2.$$

Deep Matrix Completion

Consider recovering $\boldsymbol{\Phi} \in \mathbb{R}^{d \times d}$ with $r := \text{rank}(\boldsymbol{\Phi}) \ll d$ with minimum number of observation encoded by $\boldsymbol{\Omega} \in \{0, 1\}^{d \times d}$:

$$\min_{\boldsymbol{\Theta}} \ell_{\mathrm{mc}}(\boldsymbol{\Theta}) := \frac{1}{2} \|\boldsymbol{\Omega} \odot (\boldsymbol{W}_{L:1} - \boldsymbol{\Phi})\|_F^2.$$

- If full observation $\boldsymbol{\Omega} = \mathbf{1}_d \mathbf{1}_d^\top$ available, the problem simplifies to deep matrix factorization.

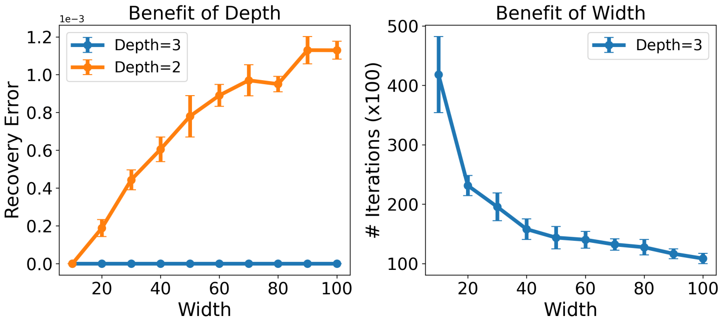
Deep Matrix Completion

Consider recovering $\mathbf{\Phi} \in \mathbb{R}^{d \times d}$ with $r := \text{rank}(\mathbf{\Phi}) \ll d$ with minimum number of observation encoded by $\mathbf{\Omega} \in \{0,1\}^{d \times d}$:

$$\min_{\mathbf{\Theta}} \ell_{\mathrm{mc}}(\mathbf{\Theta}) := \frac{1}{2} \|\mathbf{\Omega} \odot (\boldsymbol{W}_{L:1} - \mathbf{\Phi})\|_F^2.$$

- If full observation $\mathbf{\Omega} = \mathbf{1}_d \mathbf{1}_d^\top$ available, the problem simplifies to deep matrix factorization.
- If the network depth $L = 2$, it reduces to the Burer-Monteiro factorization formulation.

Why Deep Matrix Factorization and Overparameterization?



- **Benefits of Depth (Left):** Improved sample complexity[4] and less prone to overfitting.
- **Benefits of Width (Right):** Increasing the width of the network results in accelerated convergence in terms of iterations.

[4] Arora, S., Cohen, N., Hu, W., & Luo, Y. (2019). Implicit regularization in deep matrix factorization. Advances in Neural Information Processing Systems, 32.

Overparameterization: A Double Edged Sword

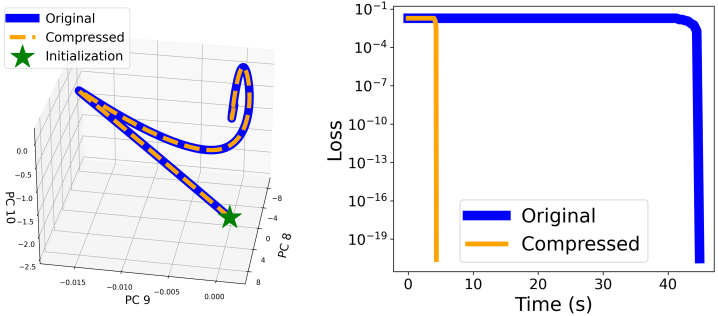


Figure: **Efficient training of deep linear networks.**

Cons: Increasing the depth and width of the network leads to much **more parameters**. Could be **expensive to optimize!**

How to Achieve the Best of Two Worlds?

- **Deep matrix factorization.** As a starting point, consider the simple deep matrix factorization setting:

$$\min_{\mathbf{\Theta}} \; \frac{1}{2}\|\boldsymbol{W}_{L:1} - \mathbf{\Phi}\|_F^2,$$

with $\mathbf{\Omega} = \mathbf{1}_d \mathbf{1}_d^\top$. We optimize the problem via GD from ε-scale orthogonal initialization.

How to Achieve the Best of Two Worlds?

- **Deep matrix factorization.** As a starting point, consider the simple deep matrix factorization setting:

$$\min_{\boldsymbol{\Theta}} \ \frac{1}{2}\|\boldsymbol{W}_{L:1} - \boldsymbol{\Phi}\|_F^2,$$

with $\boldsymbol{\Omega} = \mathbf{1}_d \mathbf{1}_d^\top$. We optimize the problem via GD from ε-scale orthogonal initialization.

- **Law of parsimony in GD** for the end-to-end matrix $\boldsymbol{W}_{L:1}$:

$$\begin{aligned}\boldsymbol{W}_{L:1}(t) &= \begin{bmatrix}\boldsymbol{U}_{L,1} & \boldsymbol{U}_{L,2}\end{bmatrix} \begin{bmatrix}\widetilde{\boldsymbol{W}}_{L:1}(t) & \mathbf{0} \\ \mathbf{0} & \rho^L(t)\boldsymbol{I}_m\end{bmatrix} \begin{bmatrix}\boldsymbol{V}_{1,1}^\top \\ \boldsymbol{V}_{1,2}^\top\end{bmatrix} \\ &= \boldsymbol{U}_{L,1}\widetilde{\boldsymbol{W}}_{L:1}(t)\boldsymbol{V}_{1,1}^\top + \rho^L(t)\boldsymbol{U}_{L,2}\boldsymbol{V}_{1,2}^\top,\end{aligned}$$

where we overestimate the rank $\widehat{r} > r$ and let $m = d - 2\widehat{r}$.

How to Achieve the Best of Two Worlds?

- **The effects of small initialization ε and depth L:**

$$\begin{aligned} \boldsymbol{W}_{L:1}(t) &= \boldsymbol{U}_{L,1}\widetilde{\boldsymbol{W}}_{L:1}(t)\boldsymbol{V}_{1,1}^{\top} + \rho^{L}(t)\boldsymbol{U}_{L,2}\boldsymbol{V}_{1,2}^{\top} \\ &\approx \boldsymbol{U}_{L,1}\widetilde{\boldsymbol{W}}_{L:1}(t)\boldsymbol{V}_{1,1}^{\top}, \quad \forall t \geq 0, \end{aligned}$$

How to Achieve the Best of Two Worlds?

- **The effects of small initialization ε and depth L:**

$$\begin{aligned} \boldsymbol{W}_{L:1}(t) &= \boldsymbol{U}_{L,1}\widetilde{\boldsymbol{W}}_{L:1}(t)\boldsymbol{V}_{1,1}^\top + \rho^L(t)\boldsymbol{U}_{L,2}\boldsymbol{V}_{1,2}^\top \\ &\approx \boldsymbol{U}_{L,1}\widetilde{\boldsymbol{W}}_{L:1}(t)\boldsymbol{V}_{1,1}^\top, \quad \forall t \geq 0, \end{aligned}$$

Claim: With small initialization, running GD on the original weights $\{\boldsymbol{W}_l\}_{l=1}^L \subseteq \mathbb{R}^{d\times d}$ is **almost equivalent** to running GD on the compressed weights $\{\widetilde{\boldsymbol{W}}_l\}_{l=1}^L \subseteq \mathbb{R}^{2\widehat{r}\times 2\widehat{r}}$.

The Simple Case: Deep Matrix Factorization

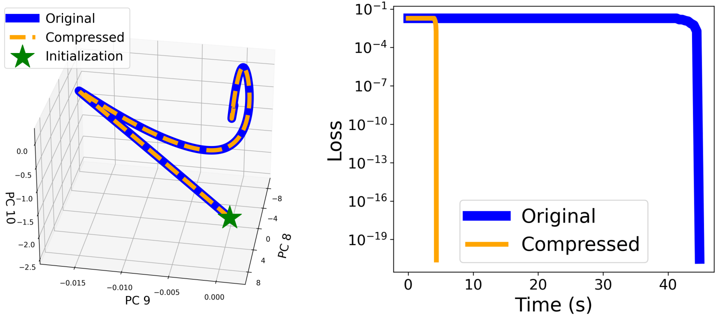
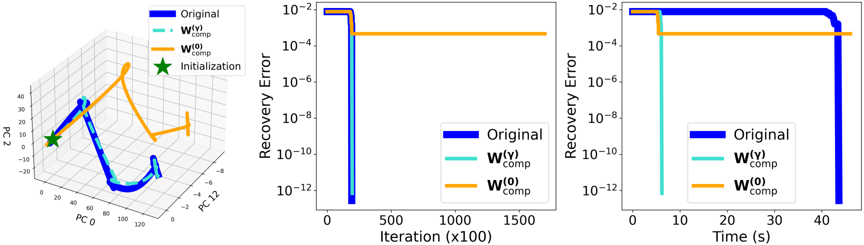


Figure: **Efficient training of deep linear networks.**

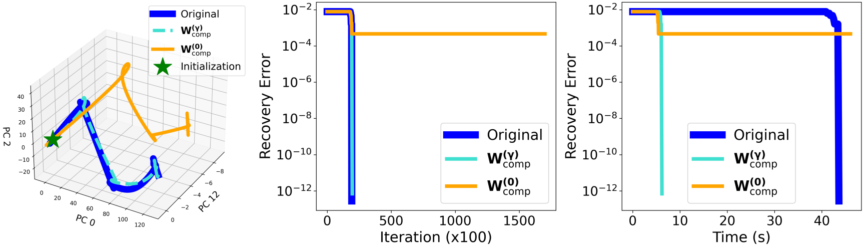
Comparison on the number of parameters: original network Ld^2 vs. compressed network $L\widehat{r}^2$.

From Deep Matrix Factorization to Completion?



- However, directly applying our approach from deep matrix factorization to completion does not work well...

From Deep Matrix Factorization to Completion?



- However, directly applying our approach from deep matrix factorization to completion does not work well...
- This is due to the fact that the law of parsimony in GD:

$$\boldsymbol{W}_{L:1}(t) \approx \boldsymbol{U}_{L,1} \widetilde{\boldsymbol{W}}_{L:1}(t) \boldsymbol{V}_{1,1}^{\top}, \quad \forall t \geq 0,$$

does NOT hold, because $\boldsymbol{\Omega} \odot \boldsymbol{\Phi}$ is not low-rank for arbitrary $\boldsymbol{\Omega}$.

How to Achieve the Best of Two Worlds?

- **The effects of small initialization ε and depth L:**

$$\begin{aligned} \boldsymbol{W}_{L:1}(t) &= \boldsymbol{U}_{L,1}\widetilde{\boldsymbol{W}}_{L:1}(t)\boldsymbol{V}_{1,1}^\top + \rho^L(t)\boldsymbol{U}_{L,2}\boldsymbol{V}_{1,2}^\top \\ &\approx \boldsymbol{U}_{L,1}\widetilde{\boldsymbol{W}}_{L:1}(t)\boldsymbol{V}_{1,1}^\top, \quad \forall t \geq 0, \end{aligned}$$

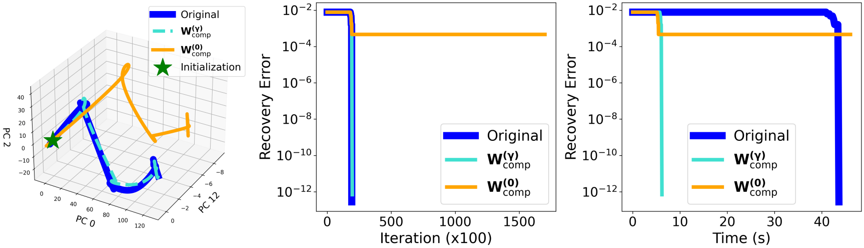
How to Achieve the Best of Two Worlds?

- **The effects of small initialization ε and depth L:**

$$\begin{aligned} \boldsymbol{W}_{L:1}(t) &= \boldsymbol{U}_{L,1}\widetilde{\boldsymbol{W}}_{L:1}(t)\boldsymbol{V}_{1,1}^\top + \rho^L(t)\boldsymbol{U}_{L,2}\boldsymbol{V}_{1,2}^\top \\ &\approx \boldsymbol{U}_{L,1}\widetilde{\boldsymbol{W}}_{L:1}(t)\boldsymbol{V}_{1,1}^\top, \quad \forall t \geq 0, \end{aligned}$$

Claim: With small initialization, running GD on the original weights $\{\boldsymbol{W}_l\}_{l=1}^L \subseteq \mathbb{R}^{d\times d}$ is **almost equivalent** to running GD on the compressed weights $\{\widetilde{\boldsymbol{W}}_l\}_{l=1}^L \subseteq \mathbb{R}^{2\widehat{r}\times 2\widehat{r}}$.

From Deep Matrix Factorization to Completion?

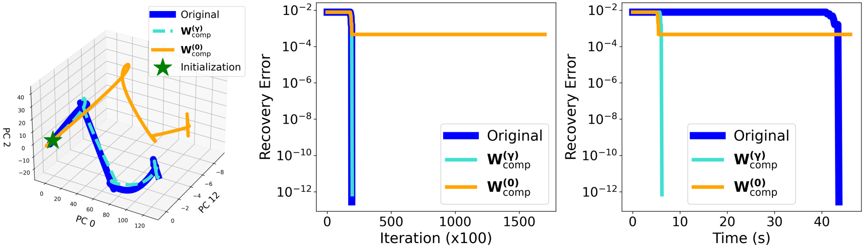


- **Remedy:** update both $V_{1,1}(t)$ and $U_{L,1}(t)$ factors via GD with a **discrepant** learning rate $\gamma\eta$ in the "compressed network":[5]

$$W^{(\gamma)}_{\text{comp}}(t) := U_{L,1}(t)\widetilde{W}_{L:1}(t)V^{\top}_{1,1}(t).$$

[5] This is done simultaneously with the GD updates on the subnetwork $\widetilde{W}_{L:1}(t)$, which uses the original learning rate η.

From Deep Matrix Factorization to Completion?



- **Remedy:** update both $V_{1,1}(t)$ and $U_{L,1}(t)$ factors via GD with a **discrepant** learning rate $\gamma\eta$ in the "compressed network":[5]

$$W^{(\gamma)}_{\mathrm{comp}}(t) := U_{L,1}(t)\widetilde{W}_{L:1}(t)V_{1,1}^{\top}(t).$$

- **Complexity:** original network $O(Ld^2)$ vs compressed network $O(Ld)$.

[5] This is done simultaneously with the GD updates on the subnetwork $\widetilde{W}_{L:1}(t)$, which uses the original learning rate η.

Low-Rank Training of Nonlinear Networks?

Factorize the weights of deeper layers in nonlinear networks into low-rank counterparts throughout training:

$$\mathbf{W}_{\mathbf{new}} = \mathbf{B}\mathbf{A}$$

where $\mathbf{B} \in \mathbb{R}^{d \times r}, \mathbf{A} \in \mathbb{R}^{r \times d}$ are trainable parameters.

- The rank r of factorization should correspond to class number K, and relaxed in shallower layers.
- This can reduce the memory and latency during training, without harming the performance.

Low-Rank Training of Nonlinear Networks?

Comparison between normal training and low rank training on MNIST, FashionMNIST, USPS using a MLP with 3 hidden layers.
We factorized the weights of the last two hidden layers, and reduced the memory and latency with comparable accuracy.

Method	# Params	Memory	FLOPs	Avg Acc.
Normal training	5.59M	0.376 GiB	1.65 TFLOPs	95.09
Low rank(r=10)	1.67M	0.113 GiB	1.17 TFLOPs	94.57
Low rank(r=1)	1.59M	0.108 GiB	1.17 TFLOPs	90.86

Low-rank Adaptation (LoRA) of Large Models?

LoRA is an SoTA parameter-efficient adaptation technique for transformers:

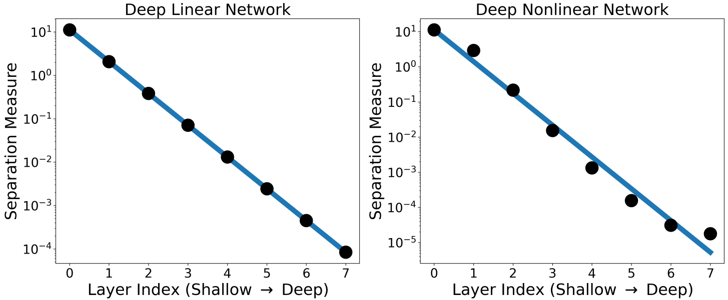
$$\mathbf{W_{new}} = \mathbf{W_0} + \mathbf{BA} \tag{1}$$

where $\mathbf{B} \in \mathbb{R}^{d\times r}, \mathbf{A} \in \mathbb{R}^{r\times d}$ are trainable parameters.

Method	**# Params**	**CIFAR10**	**CIFAR100**
Full-model	86.7M/86.7M	99.07	93.27
LoRA	0.33M/86.9M	98.97	92.85
AdaLoRA	0.33M/86.9M	98.87	92.93

Outline

Main Message



For classification problem, the law of parsimony in GD explains progressive feature separation in deep linear networks.

Problem Setup: Train DLNs for Classification Problems

- **Balanced Training Data:** $\{(\boldsymbol{x}_{k,i}, \boldsymbol{y}_k)\}_{i\in[n],k\in[K]}$ for K-class classification: $\boldsymbol{x}_{k,i} \in \mathbb{R}^d$ is the i-th sample in the k-th class, $\boldsymbol{y}_k \in \mathbb{R}^K$ is an one-hot label.
- **Feature in the l-th Layer of DLN:**

$$\boldsymbol{z}_{k,i}^l := \boldsymbol{W}_l \dots \boldsymbol{W}_1 \boldsymbol{x}_{k,i} = \boldsymbol{W}_{l:1} \boldsymbol{x}_{k,i}, \ \forall l \in [L],$$

- **With-class and between-class covariance matrices**

$$\boldsymbol{\Sigma}_W^l = \frac{1}{nK} \sum_{k=1}^{K} \sum_{i=1}^{n} \left(\boldsymbol{z}_{k,i}^l - \bar{\boldsymbol{z}}_k^l \right) \left(\boldsymbol{z}_{k,i} - \bar{\boldsymbol{z}}_k^l \right)^\top,$$

$$\boldsymbol{\Sigma}_B^l = \frac{1}{K} \sum_{k=1}^{K} \left(\bar{\boldsymbol{z}}_k^l - \bar{\boldsymbol{z}}_G^l \right) \left(\bar{\boldsymbol{z}}_k^l - \bar{\boldsymbol{z}}_G^l \right)^\top,$$

where

$$\bar{\boldsymbol{z}}_k^l = \frac{1}{n_k} \sum_{i=1}^{n_k} \boldsymbol{z}_{k,i}^l, \quad \bar{\boldsymbol{z}}_G^l = \frac{1}{K} \sum_{k=1}^{K} \bar{\boldsymbol{z}}_k^l$$

Measure of Feature Compression and Separation

- **Measure of feature compression:** (He & Su. 2022, Tirer et al. (2022))

$$\boxed{D_l \ := \ \text{trace}(\boldsymbol{\Sigma}_W^l)/\text{trace}(\boldsymbol{\Sigma}_B^l),}$$

$$\boldsymbol{\Sigma}_W^l = \frac{1}{nK} \sum_{k=1}^{K} \sum_{i=1}^{n} \left(\boldsymbol{z}_{k,i}^l - \overline{\boldsymbol{z}}_k^l \right) \left(\boldsymbol{z}_{k,i} - \overline{\boldsymbol{z}}_k^l \right)^\top ,$$

Measure of Feature Compression and Separation

- **Measure of feature compression:** (He & Su. 2022, Tirer et al. (2022))

$$\boxed{D_l \ := \ \text{trace}(\mathbf{\Sigma}_W^l)/\text{trace}(\mathbf{\Sigma}_B^l),}$$

$$\mathbf{\Sigma}_W^l = \frac{1}{nK}\sum_{k=1}^{K}\sum_{i=1}^{n}\left(\boldsymbol{z}_{k,i}^l - \bar{\boldsymbol{z}}_k^l\right)\left(\boldsymbol{z}_{k,i} - \bar{\boldsymbol{z}}_k^l\right)^\top,$$

- **Measure of between-class feature separation:**

$$\boxed{S_l := 1 - \max_{k \neq k'} \frac{|\langle \boldsymbol{\mu}_k^l, \boldsymbol{\mu}_{k'}^l \rangle|}{\|\boldsymbol{\mu}_k^l\|\|\boldsymbol{\mu}_{k'}^l\|}},$$

where

$$\boldsymbol{\mu}_k^l = \bar{\boldsymbol{z}}_k^l - \bar{\boldsymbol{z}}_G^l$$

Progressive Feature Compression with Linear Rate

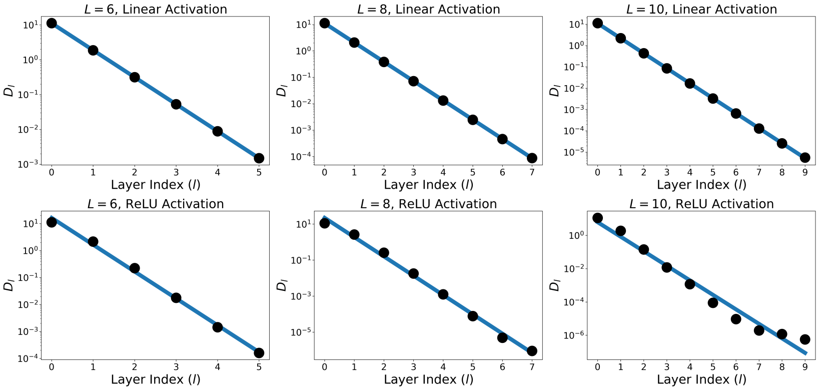


Figure: **Linear decay of feature compression in trained deep networks.** Linear networks (top) vs. nonlinear networks (bottom)

Progressive Feature Separation with Sub-Linear Rate

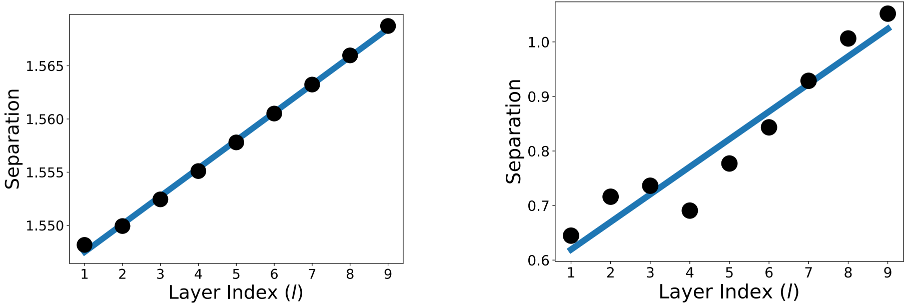


Figure: **Feature separation in trained deep networks.** Linear network (left) vs. nonlinear (right)

Assumptions

- **Assumption on the input data $X \in R^{d \times N}$ $(d \geq N)$:**

$$\left|\|x_i\|^2 - 1\right| \leq \frac{\theta}{N}, \ |\langle x_i, x_j \rangle| \leq \frac{\theta}{N}, \ \forall 1 \leq i \neq j \leq N,$$

Assumptions

- **Assumption on the input data $\boldsymbol{X} \in R^{d \times N}$ $(d \geq N)$:**

$$\left| \|\boldsymbol{x}_i\|^2 - 1 \right| \leq \frac{\theta}{N},\ |\langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle| \leq \frac{\theta}{N},\ \forall 1 \leq i \neq j \leq N,$$

- **Assumption on the trained weights Θ:**
  1. **Minimum norm solution** with zero training loss $\boldsymbol{Y} = \boldsymbol{W}_{L:1}\boldsymbol{X}$:

     $$\boldsymbol{W}_{L:1} = (\boldsymbol{X}^\top \boldsymbol{X})^{-1}\boldsymbol{X}^\top.$$

  2. **Weight balancedness:** There exists a numerical constant $\delta > 0$ s.t.

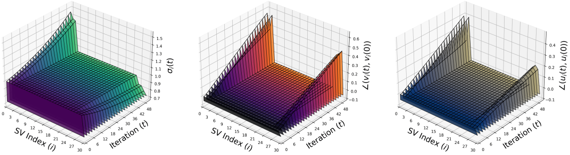
     $$\boldsymbol{W}_{l+1}^\top \boldsymbol{W}_{l+1} = \boldsymbol{W}_l \boldsymbol{W}_l^\top, \forall l \in [L-2],\ \|\boldsymbol{W}_L^\top \boldsymbol{W}_L - \boldsymbol{W}_{L-1}\boldsymbol{W}_{L-1}^\top\|_F \leq \delta.$$

  3. **Approximate low-rankness:** There exist positive constants $\varepsilon \in (0,1)$ and $\rho \in [0, \varepsilon)$,

     $$\varepsilon - \rho \leq \sigma_i(\boldsymbol{W}_l) \leq \varepsilon,\ i = K+1, \ldots, d-K$$

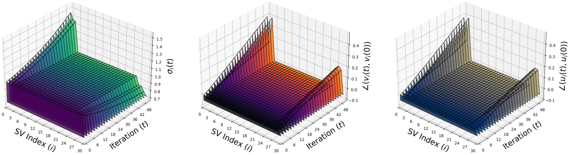
     for all $l = 1, \cdots, L-1$, where $\sigma_i(\boldsymbol{W}_l)$ is the i-th largest singular value.

Layer 1



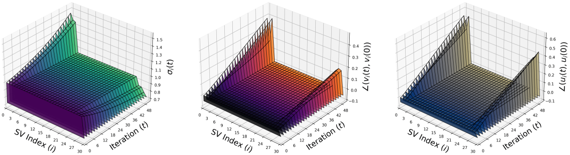
Singular Values | Right Singular Vectors | Left Singular Vectors

Layer 2



Singular Values | Right Singular Vectors | Left Singular Vectors

Layer 3



Singular Values | Right Singular Vectors | Left Singular Vectors

Progressive Feature Compression with Linear Rate

Theorem (Wang et al.'23)

Suppose our training data $(\boldsymbol{X}, \boldsymbol{Y})$ *and the trained weights* $\boldsymbol{\Theta}$ *of an* L*-layer DLN satisfy the above assumptions. Then we have*

- **Progressive feature compression:** *For all* $l \in [L-2]$*, we have*

$$\frac{c\varepsilon^2}{\kappa(4n)^{1/L}} \leq \frac{D_{l+1}}{D_l} \leq \frac{\kappa\varepsilon^2}{c(n/2)^{1/L}},$$

- **Progressive feature separation:**

$$S_l \geq 1 - \frac{32\,(\theta + 4\delta)}{L}\,(L - l - 1) + o(1)$$

Effects of Initialization Scale ε

As predicted by our theory, the decay ratio critically depends on the scale of initialization ε:

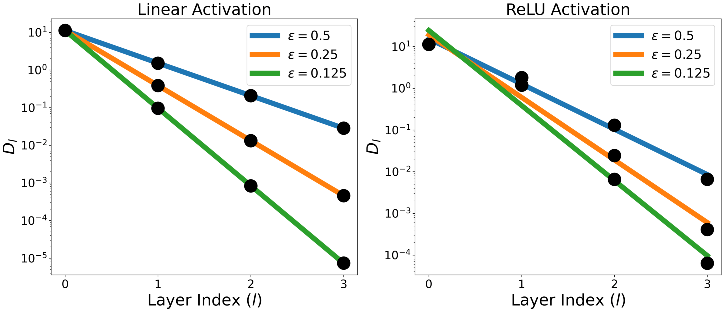


Figure: **Linear decay of feature compression D_l in trained deep networks with varying initialization scale ε.**

Tradeoffs Between Decay Rate and Convergence

However, there is trade-off between decay rate ε and training speed of GD:

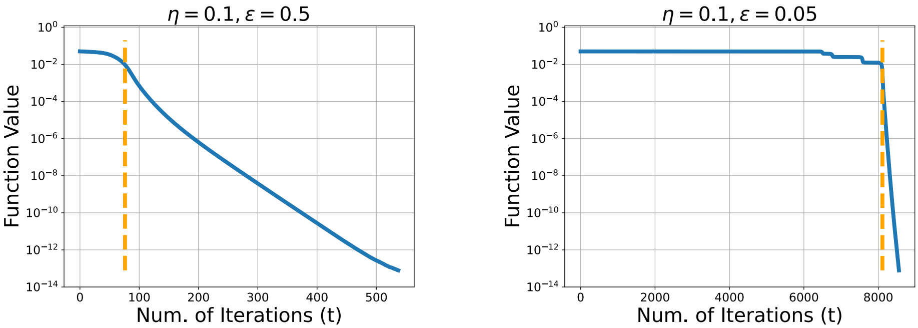


Figure: **The dynamics of GD for DLNs with learning rate $\eta = 0.1$.**

Effects of Initialization Type

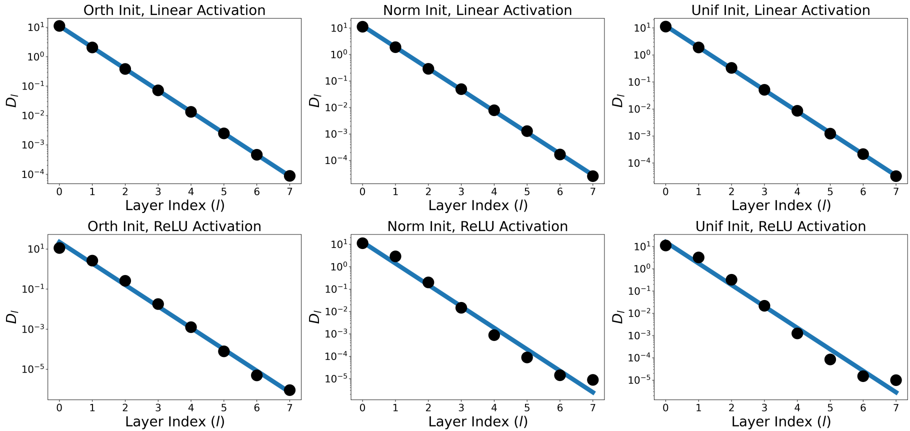


Figure: **Linear decay of feature compression in trained DLNs with different initialization types (left to right: Orth., Norm, Unif).**

Outline

Conclusion

The GD learning process takes place only within a **minimal invariant subspace** of each weight matrix, while the remaining singular subspaces stay **unaffected** throughout training.

- **Efficient low-rank training and network compression.**
- **Understanding hierarchal representations in deep networks.**

References

1 Yaras, C.*, Wang, P.*, Hu, W., Zhu, Z., Balzano, L., Qu, Q. (2023). The Law of Parsimony in Gradient Descent for Learning Deep Linear Networks. arXiv preprint arXiv:2306.01154.

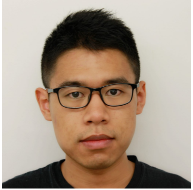
2 Wang, P., Yaras, C., Li, X., Hu, W., Zhu, Z., Balzano, L., Qu, Q. (2023). Unveiling Hierarchical Representations in Deep Networks via Feature Compression and Discrimination. Working paper.

3 Li, X., Liu S., Zhou, J., Lu, X., Fernandez-Granda, C., Zhu, Z., Qu, Q. (2023) Principled and Efficient Transfer Learning of Deep Models via Neural Collapse, arXiv preprint arXiv:2212.12206.

4 Kwon S., Zhang Z., Song D., Qu Q., Fast and Compressed Deep Linear Networks for Learning Low-Dimensional Models, Working paper.

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

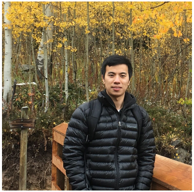
Acknowledgement



Wei Hu

Peng Wang

Xiao Li



Zhihui Zhu

Laura Balzano

Can Yaras

Soo-Min Kwon

Dogyoon Song

Conclusion

The GD learning process takes place only within a **minimal invariant subspace** of each weight matrix, while the remaining singular subspaces stay **unaffected** throughout training.

- **Efficient low-rank training and network compression.**
- **Understanding hierarchal representations in deep networks.**

Thank You! Questions?

Compressed Networks vs. Narrow Networks?

Question: Does law of parsimony imply that optimizing a narrow network of the same width $2\widehat{r}$ would perform just as efficiently as the compressed network with a true width of $d \gg \widehat{r}$?

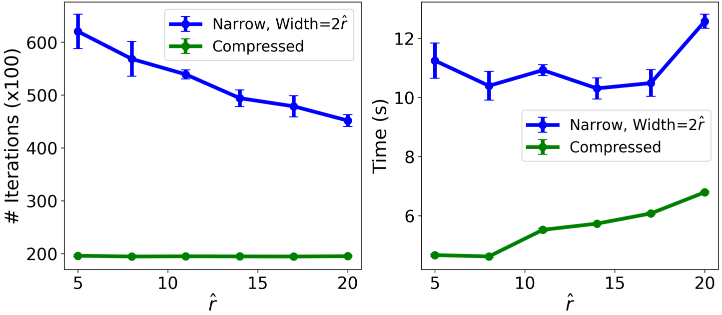


Figure: **Efficiency of compressed networks vs. narrow network.**

Compressed Networks vs. Narrow Networks?

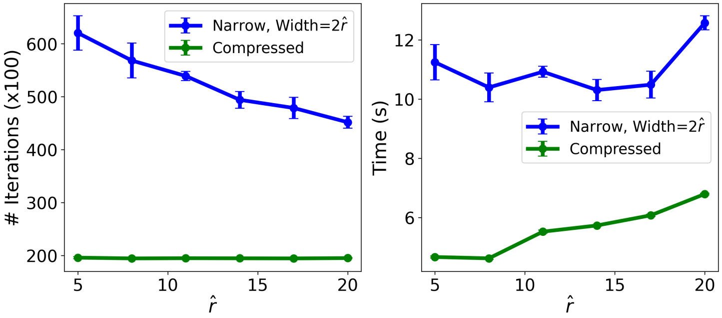


Figure: **Efficiency of compressed networks vs. narrow network.**

Answer: No! Over-parameterized networks are “easier” to train.