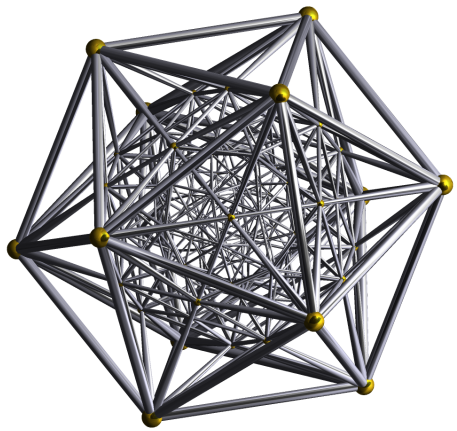
ACDL Summer Course 2023

Lecture 3: Low-Dimensional Structures in Deep Representation Learning II

Qing Qu

EECS, University of Michigan

June 10th, 2023



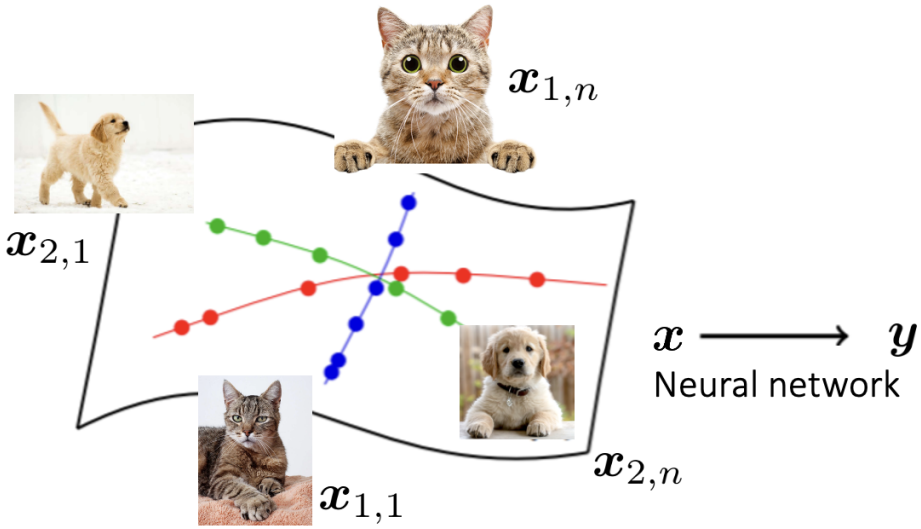
Outline

1. Neural Collapse, Transfer Learning, & Intermediate Layers
2. Law of Parsimony in Gradient Dynamics
3. Progressive Feature Separation in Deep Neural Networks
4. Efficient Deep Matrix Completion
5. Conclusion

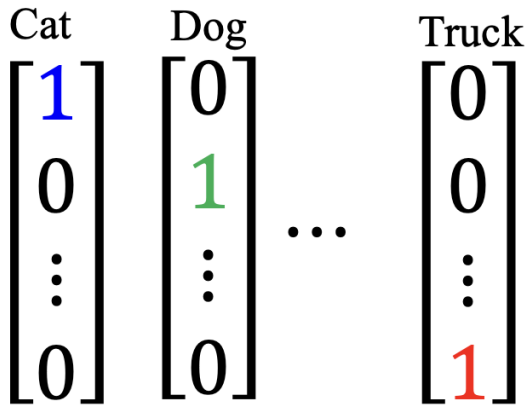
Neural Collapse in Classification

Labels: $k = 1, \ldots, K$

- $K = 10$ classes (MNIST, CIFAR10, etc)
- $K = 1000$ classes (ImageNet)



Data in the input space



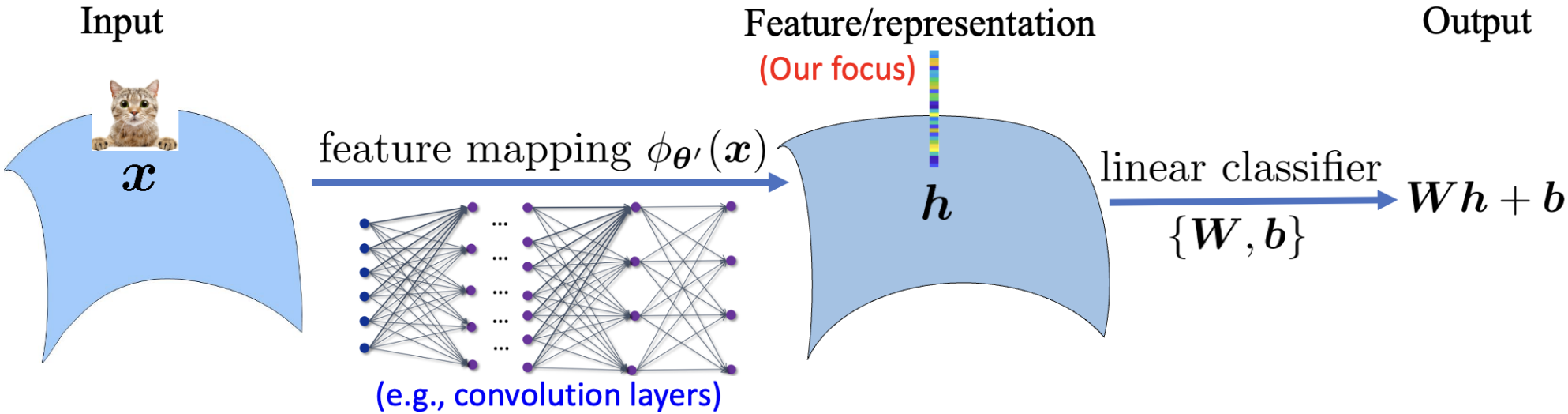
One-hot labeling vectors in $\mathbb{R}^K$

Assume balanced dataset where each class has n training samples

- If not, we can use data augmentation to make them balanced

Deep Neural Network Classifiers

A deep neural network classifier often contains two parts: a feature mapping and a linear classifier



- Output: $f(\boldsymbol{x}; \boldsymbol{\theta}) = \boldsymbol{W}\phi_{\boldsymbol{\theta}'}(\boldsymbol{x}) + \boldsymbol{b}$ with $\boldsymbol{\theta} = (\boldsymbol{\theta}', \boldsymbol{W}, \boldsymbol{b})$.
- Training problem:

$$\min_{\boldsymbol{\theta}', \boldsymbol{W}, \boldsymbol{b}} \frac{1}{Kn} \sum_{k=1}^{K} \sum_{i=1}^{n} \underbrace{\mathcal{L}_{\mathrm{CE}}\big(\boldsymbol{W}\phi_{\boldsymbol{\theta}'}(\boldsymbol{x}_{k,i}) + \boldsymbol{b}, \boldsymbol{y}_k\big)}_{\text{cross-entropy (CE) loss}} + \lambda \underbrace{\|(\boldsymbol{\theta}', \boldsymbol{W}, \boldsymbol{b})\|_F^2}_{\text{weight decay}}$$

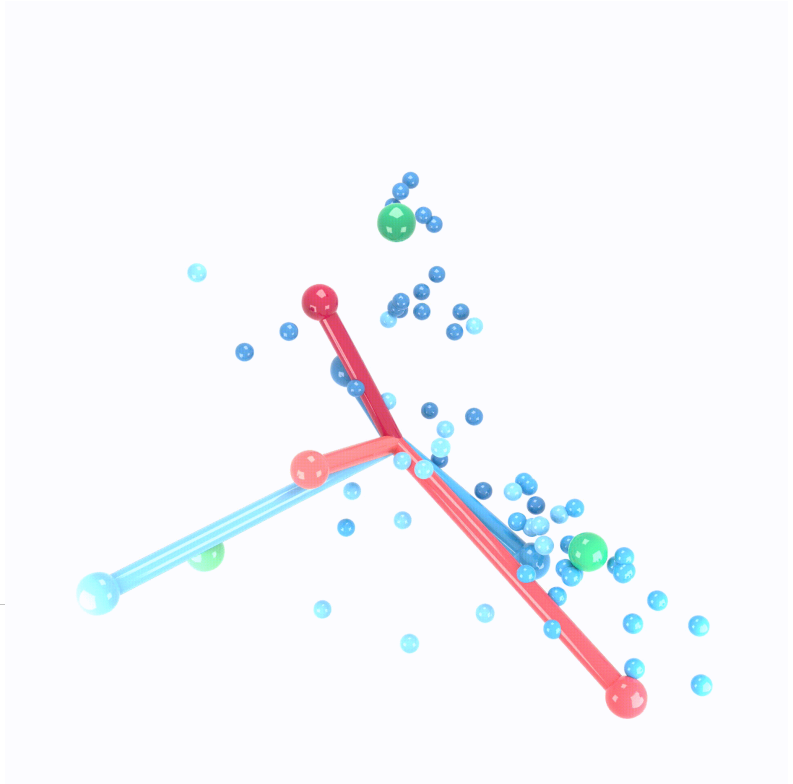
Neural Collapse in Classification

Prevalence of neural collapse during the terminal phase of deep learning training

Vardan Papyan, X. Y. Han, and David L. Donoho
+ See all authors and affiliations

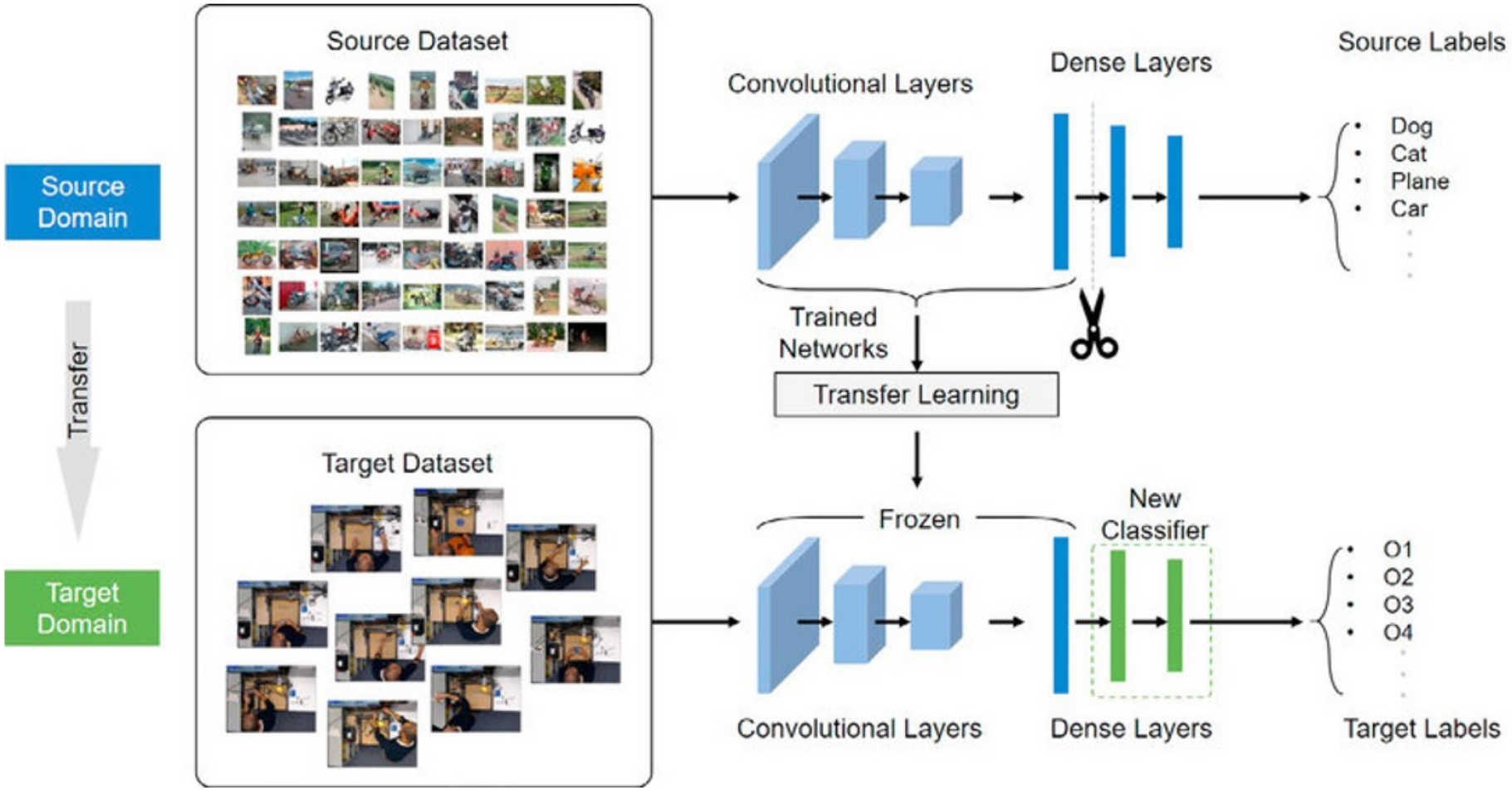
PNAS October 6, 2020 117 (40) 24652-24663; first published September 21, 2020;
https://doi.org/10.1073/pnas.2015509117

Contributed by David L. Donoho, August 18, 2020 (sent for review July 22, 2020; reviewed by Helmut Boelsckei and Stéphane Mallat)



- Reveals common outcome of learned features and classifiers across a variety of architectures and dataset
- Precise mathematical structure within the features and classifier

Implications of Neural Collapse in Transfer Learning?



Efficient Fine-tuning of Pre-trained Models

- **Full model fine-tuning** (use pre-trained model as an initialization)
  - Expensive & prune to overfitting
- **Linear probing** with penultimate layer features (use pre-trained model as a feature extractor)
  - cheap but worse performance

Efficient Fine-tuning of Pre-trained Models

- **Full model fine-tuning** (use pre-trained model as an initialization)
  - Expensive & prune to overfitting
- **Linear probing** with penultimate layer features (use pre-trained model as a feature extractor)
  - cheap but worse performance

Message: We can design simple and efficient fine-tuning approaches of pre-trained models via Neural Collapse.

Measure of Variability Collapse on Downstream Data

One key metric for Neural Collapse:

$$\boxed{\mathcal{NC}_1 = \mathsf{trace}(\boldsymbol{\Sigma}_W \boldsymbol{\Sigma}_B^{\dagger}).}$$

- **Within-class covariance:**

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

- **Between-class covariance:**

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

More Variability Collapse, Better Transfer Performance

Neural Collapse of pre-trained models evaluated on **downstream data**:

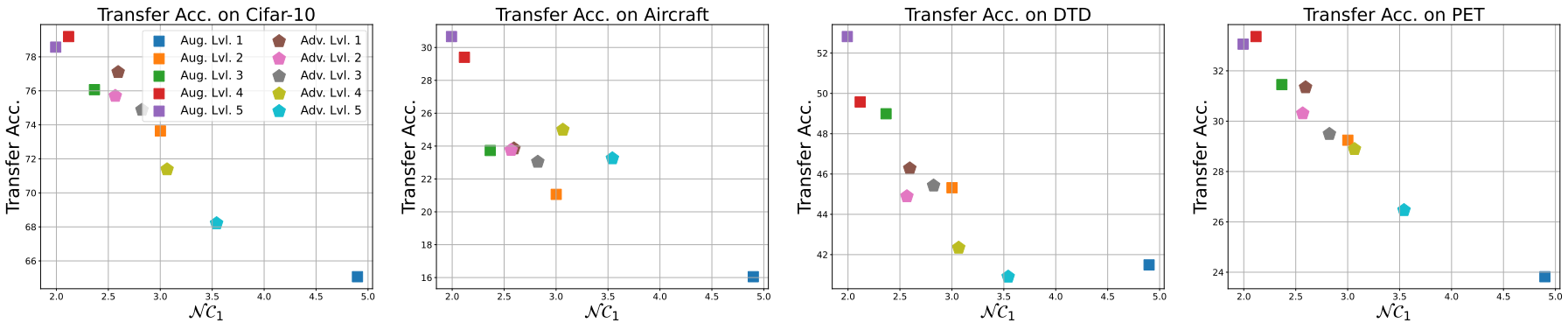


Figure: **Transfer accuracy and $\mathcal{NC}_1$ of Cifar-100 pre-trained models on different downstream tasks.**

More Variability Collapse, Better Transfer Performance

Neural Collapse of pre-trained models evaluated on **downstream data**:

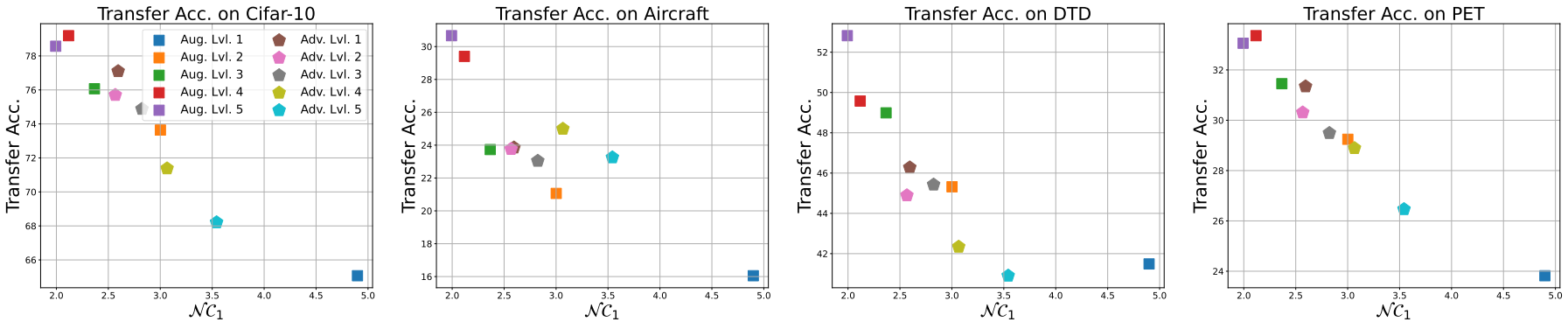


Figure: **Transfer accuracy and $\mathcal{NC}_1$ of Cifar-100 pre-trained models on different downstream tasks.**

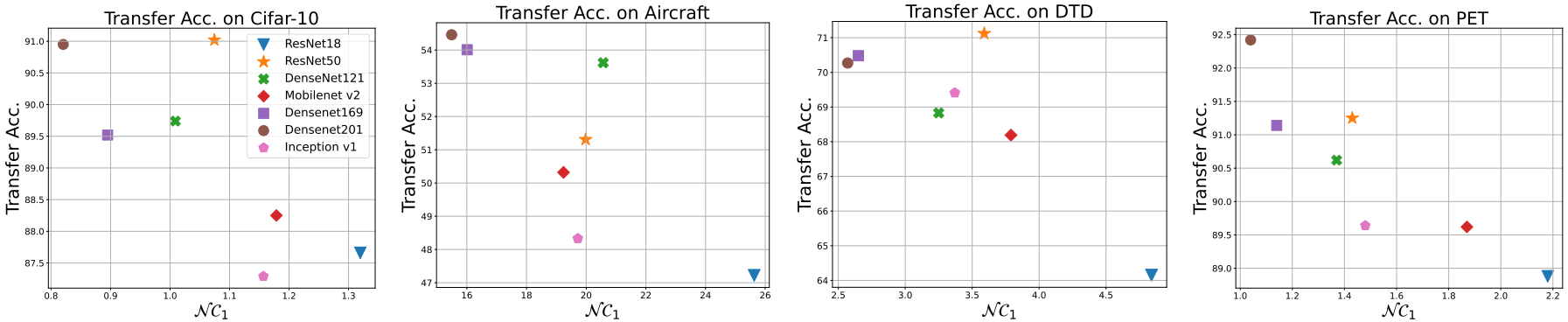


Figure: **Transfer accuracy and $\mathcal{NC}_1$ of public ImageNet-1k pre-trained models on different downstream tasks.**

Parameter-Efficient Fine-Tuning via Neural Collapse

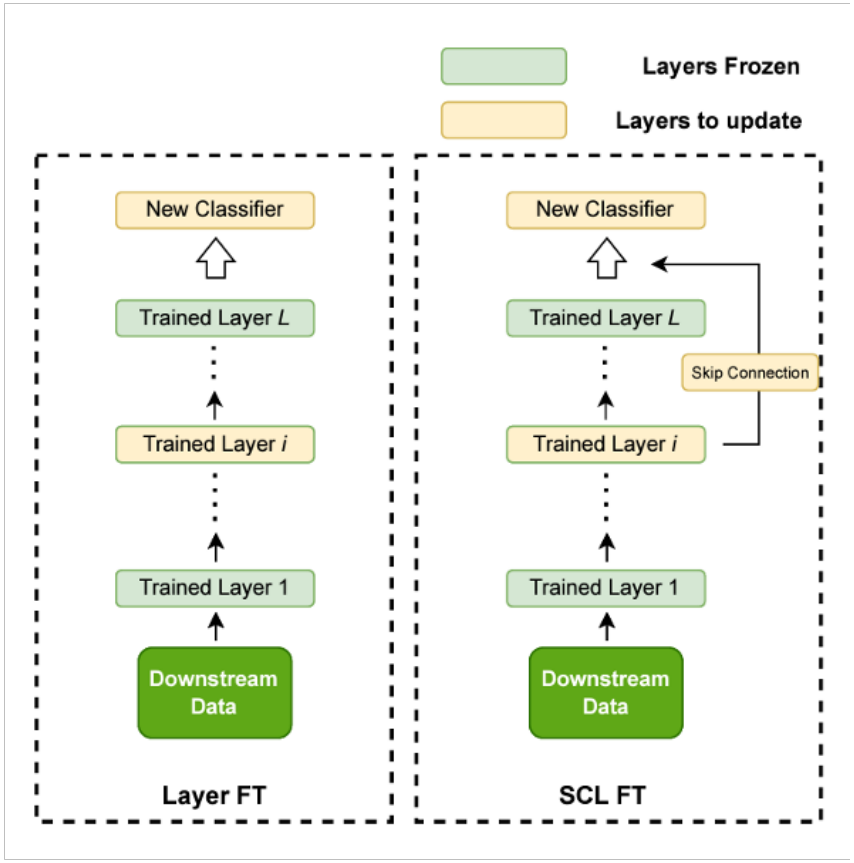
Observation: Better transfer performance can be achieved by making the last-layer features more collapsed on the downstream data.

Parameter-Efficient Fine-Tuning via Neural Collapse

Observation: Better transfer performance can be achieved by making the last-layer features more collapsed on the downstream data.

Skip connection layer fine-tuning (SCL-FT):

- Only fine-tuning one key intermediate layer.
- Improving feature collapse via skip connections.

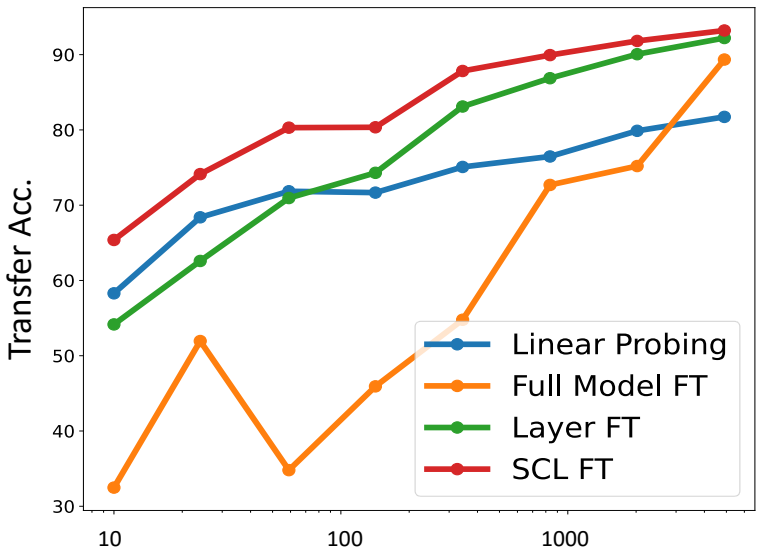


Better Performance with Significantly Fewer Parameters

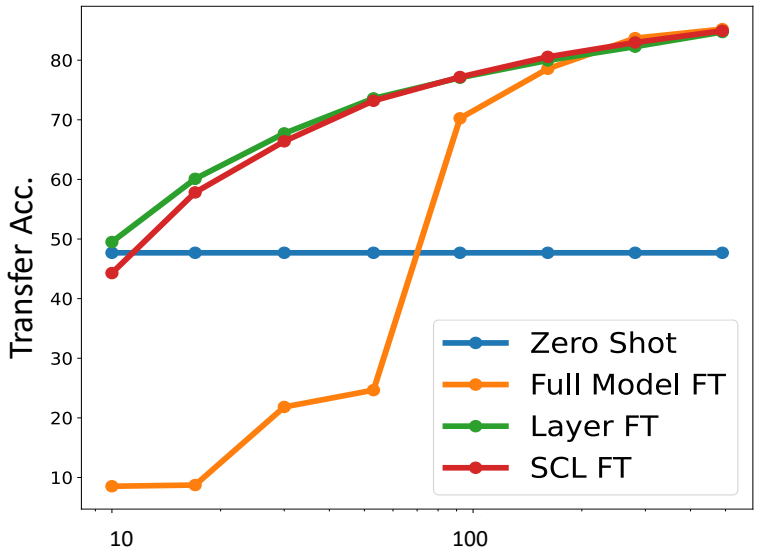
Backbone	ResNet50					Vit-B			CLIP		
Dataset	Cifar-10	Cifar-100	Aircraft	DTD	PET	Aircraft	DTD	PET	Aircraft	DTD	PET
Transfer accuracy											
Linear Probe / Zero Shot	85.33	65.47	43.23	68.46	89.26	43.65	73.88	92.23	12.87	32.34	39.66
Layer FT	94.04	77.47	70.27	67.66	89.40	**65.83**	77.13	92.94	**67.63**	**79.47**	**91.09**
SCL FT	**94.94**	78.32	70.72	**72.87**	**91.69**	65.80	**77.34**	**93.13**	66.58	79.04	90.02
Full Model FT	85.51	**78.88**	**80.77**	76.12	73.24	64.66	76.54	93.02	59.11	72.82	84.44
$\mathcal{NC}_1$ **evaluated on the penultimate layer feature** h^{L-1}											
Linear Probe / Zero Shot	1.84	18.36	20.36	3.52	1.45	17.91	1.99	0.66	17.47	2.96	3.77
Layer FT	0.28	3.22	3.37	1.68	0.68	6.98	1.62	0.44	1.30	0.42	0.32
SCL FT	0.22	2.61	1.02	0.64	0.39	7.48	1.33	0.35	1.65	0.61	0.46
Full Model FT	0.17	0.15	0.61	0.31	0.28	3.78	1.11	0.21	0.49	0.17	0.18
Percentage of parameters fine-tuned											
Linear Probe / Zero Shot	0.09%	0.86%	0.86%	0.41%	0.32%	0.09%	0.04%	0.03%	0.0%	0.0%	0.0%
Layer FT	6.52%	7.24%	7.24%	6.82%	6.73%	8.18%	8.14%	8.13%	8.16%	8.11%	8.10%
SCL FT	6.52%	7.24%	7.24%	6.82%	6.73%	8.18%	8.14%	8.13%	8.16%	8.11%	8.10%
Full Model FT	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%

Table: **Transfer learning results for linear probing / zero-shot, layer FT, SCL FT and full model FT on downstream datasets.** We use publicly available ResNet50, ViT-B and CLIP models.

Less Overfitting Compared to Full Model FT



(a) ResNet18, Cifar-10



(b) CLIP, Cifar-100

Figure: **Transfer accuracy for different fine-tuning methods with varying size of downstream training dataset.**

We fine-tune ImageNet-1k pre-trained ResNet18 models and CLIP using subsets of the Cifar-10 and Cifar-100 downstream datasets, respectively with varying sizes.

Parameter-Efficient Fine-Tuning via Neural Collapse

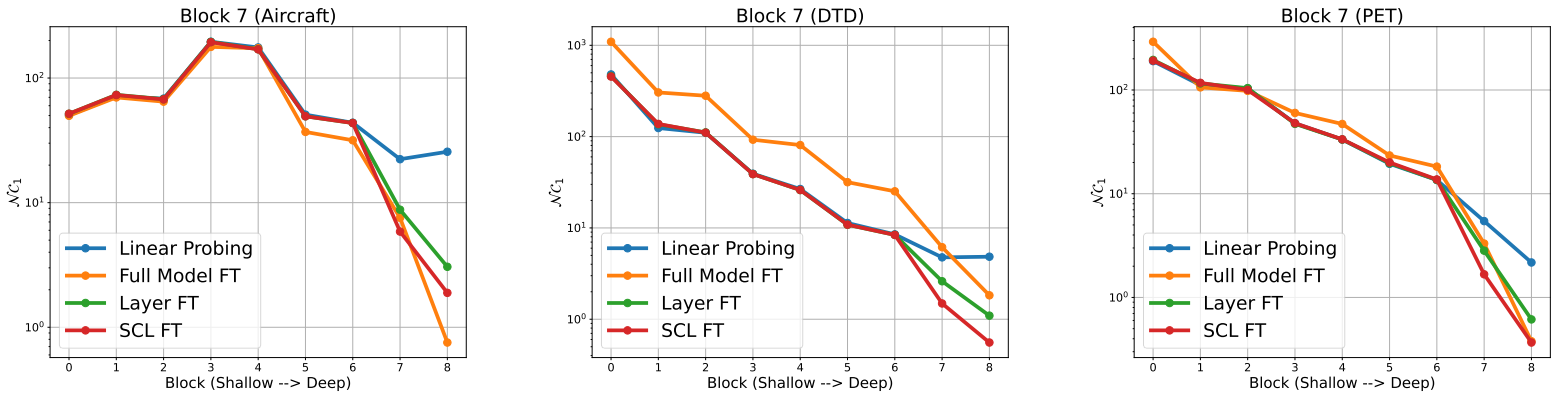


Figure: **Layer-wise $\mathcal{NC}_1$ for different fine-tuning methods on ResNet18.**

Parameter-Efficient Fine-Tuning via Neural Collapse

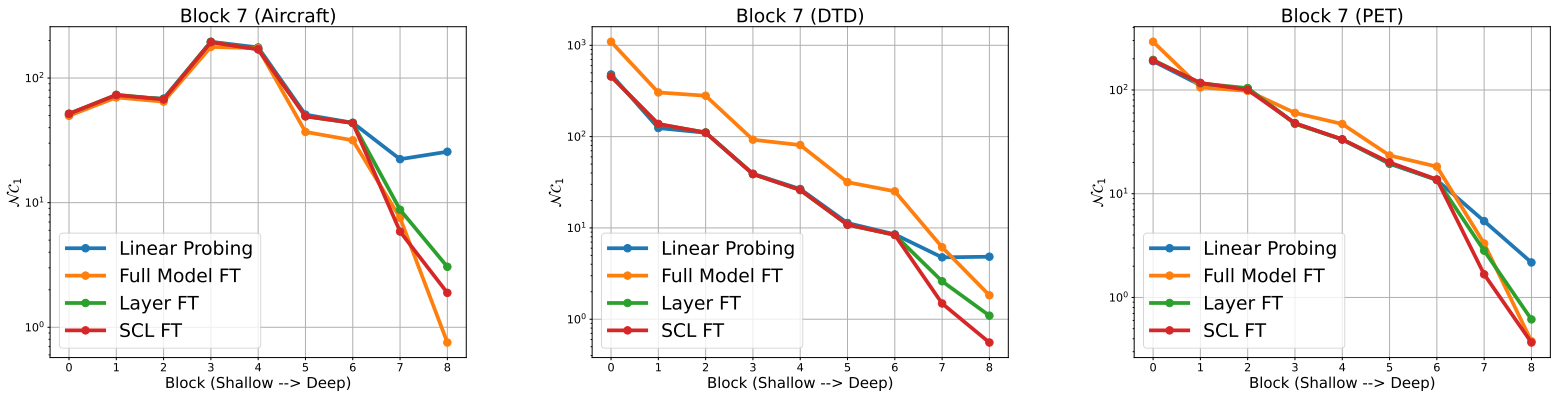


Figure: **Layer-wise $\mathcal{NC}_1$ for different fine-tuning methods on ResNet18.**

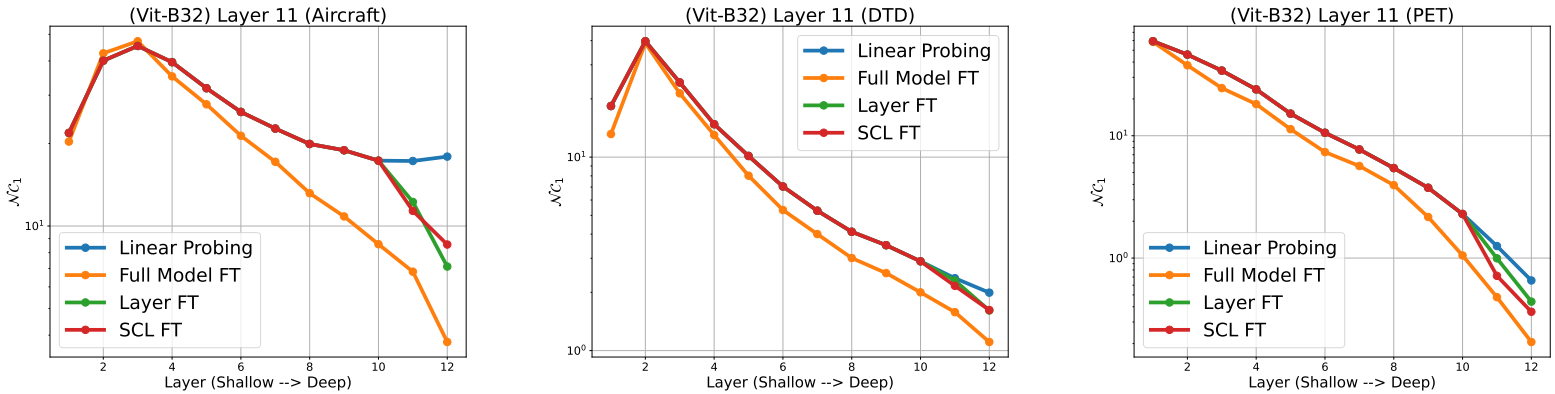


Figure: **Layer-wise $\mathcal{NC}_1$ for different fine-tuning methods on Vision Transformer ViT-B32.**

Representation Structures within the Intermediate Layers?

Is there any structure in the representations of intermediate layers?[1]

$$\boxed{\mathcal{NC}_1^l = \text{trace}(\boldsymbol{\Sigma}_W^l \boldsymbol{\Sigma}_B^{l\dagger}).}$$

- Between-class covariance:

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

- Within-class covariance:

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

[1] V. Papyan, Traces of class/cross-class structure pervade deep learning spectra, JMLR, 2021. He & Su, A Law of Progressive Separation for Deep Learning, 2022.

Representation Structures within the Intermediate Layers?

Is there any structure in the representations of intermediate layers?[1]

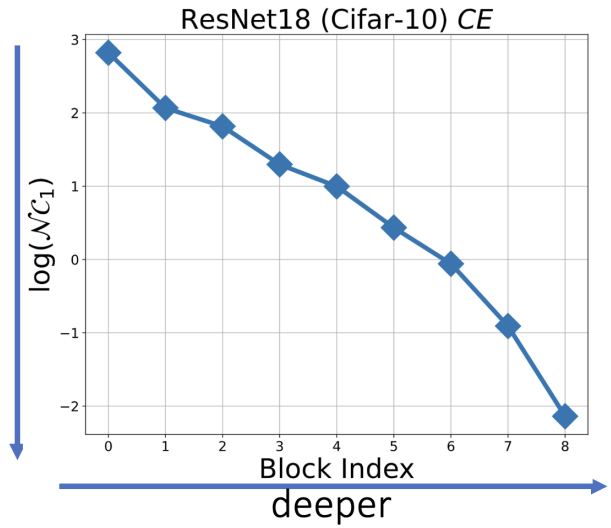
$$\boxed{\mathcal{NC}_1^l = \text{trace}(\boldsymbol{\Sigma}_W^l \boldsymbol{\Sigma}_B^{l\dagger}).}$$

- Between-class covariance:

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

- Within-class covariance:

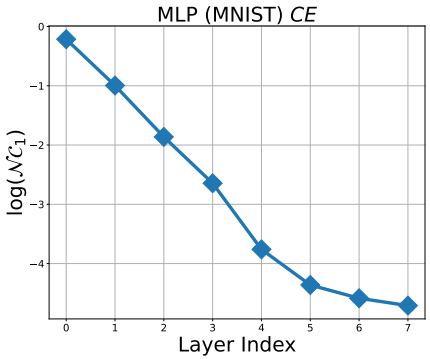
$$\boldsymbol{\Sigma}_W^l = \frac{1}{nK}\sum_{k=1}^{K}\sum_{i=1}^{n}\left(\boldsymbol{z}_{k,i}^l - \bar{\boldsymbol{z}}_k^l\right)\left(\boldsymbol{z}_{k,i} - \bar{\boldsymbol{z}}_k^l\right)^\top$$

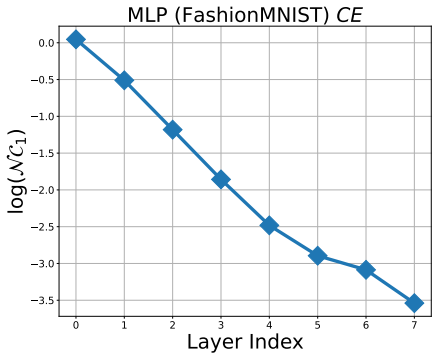


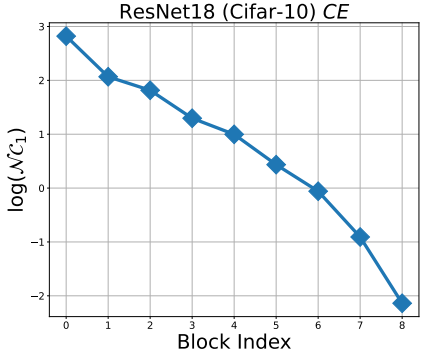
Effects of Depth: Creating **progressive** separation and concentration of data from shallow to deep layers!

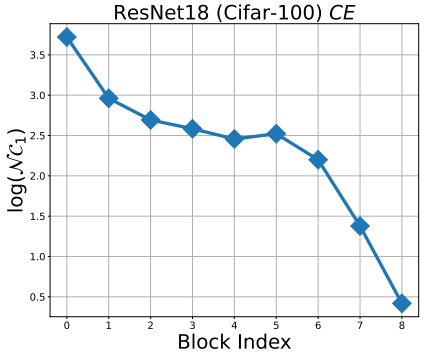
[1] V. Papyan, Traces of class/cross-class structure pervade deep learning spectra, JMLR, 2021. He & Su, A Law of Progressive Separation for Deep Learning, 2022.

Prevalence of Progressive Separation in Deep Learning

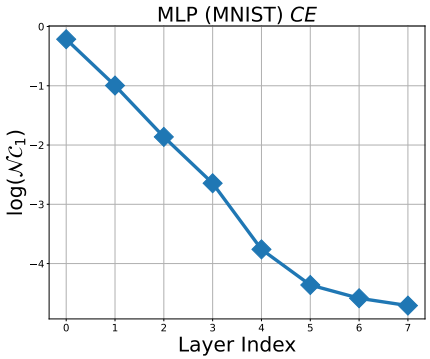


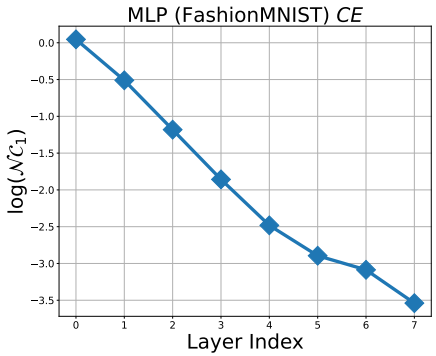


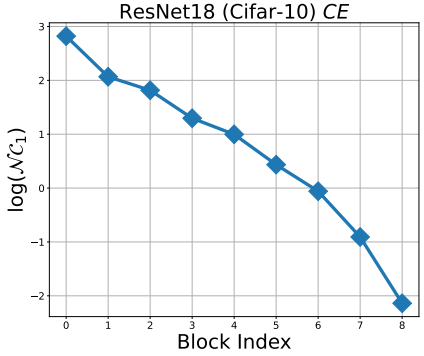


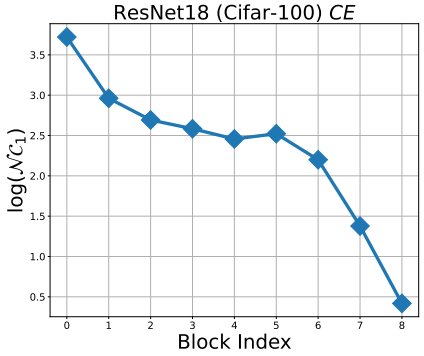


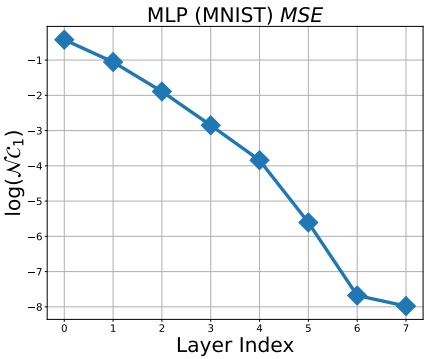
Prevalence of Progressive Separation in Deep Learning

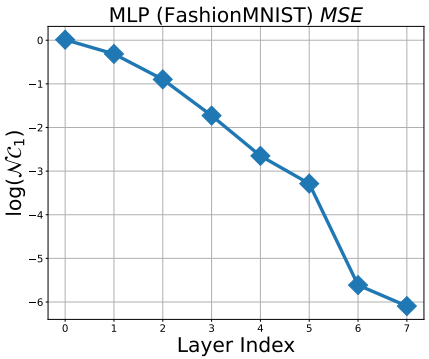


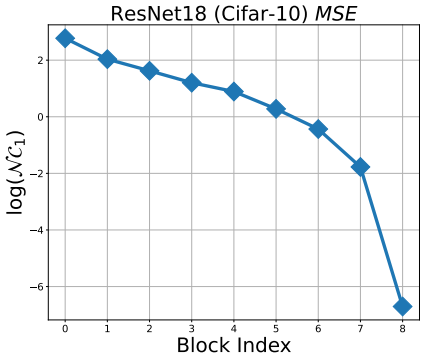


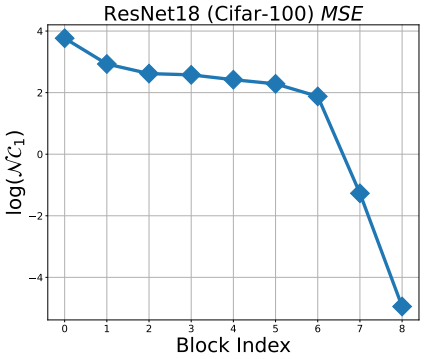




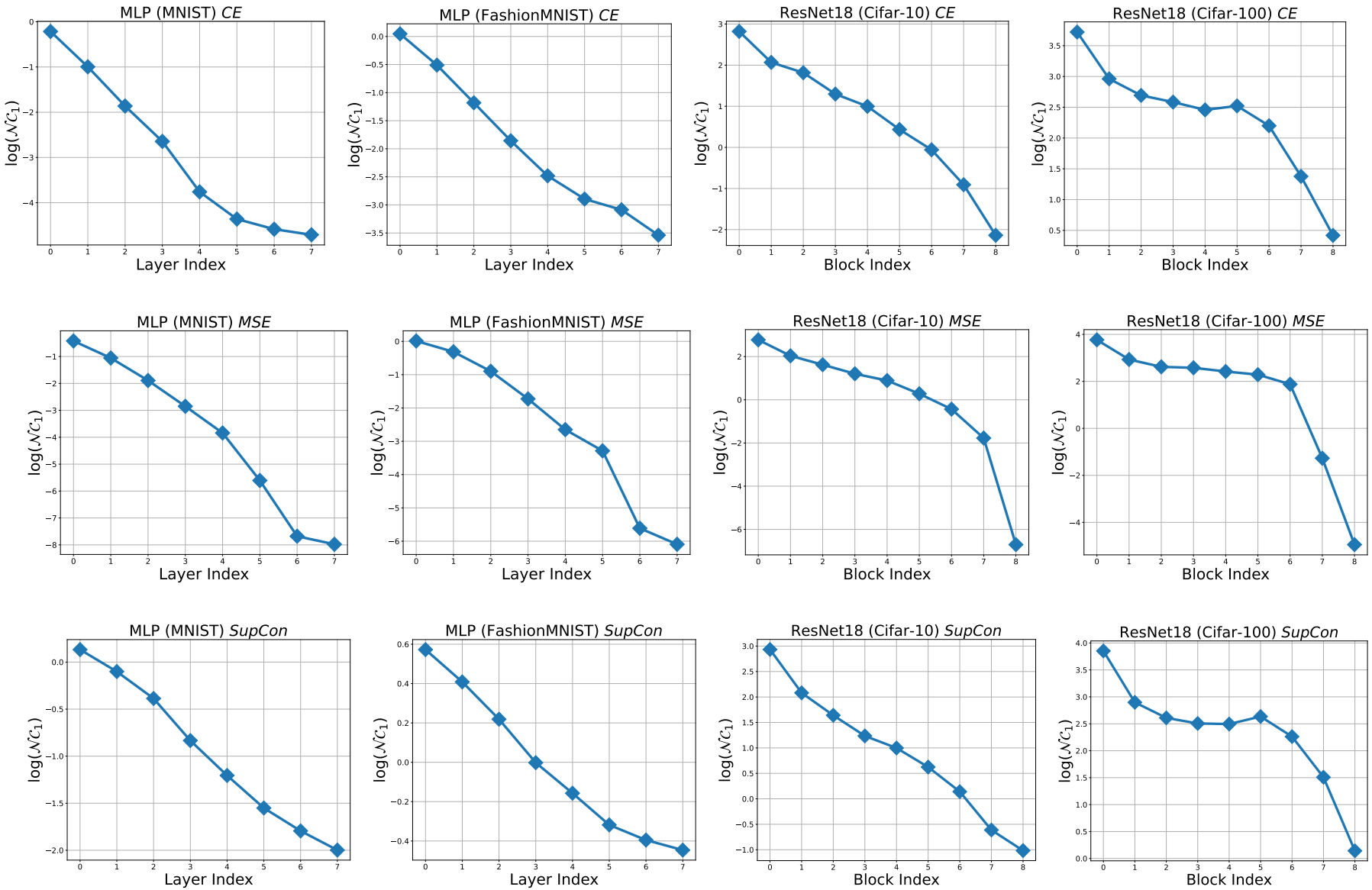








Prevalence of Progressive Separation in Deep Learning



Towards Understanding Progressive Collapse?

Given linearly separable data $\boldsymbol{X}$, we measure a certain metric of data separation of each layer's output for linear and nonlinear deep networks:

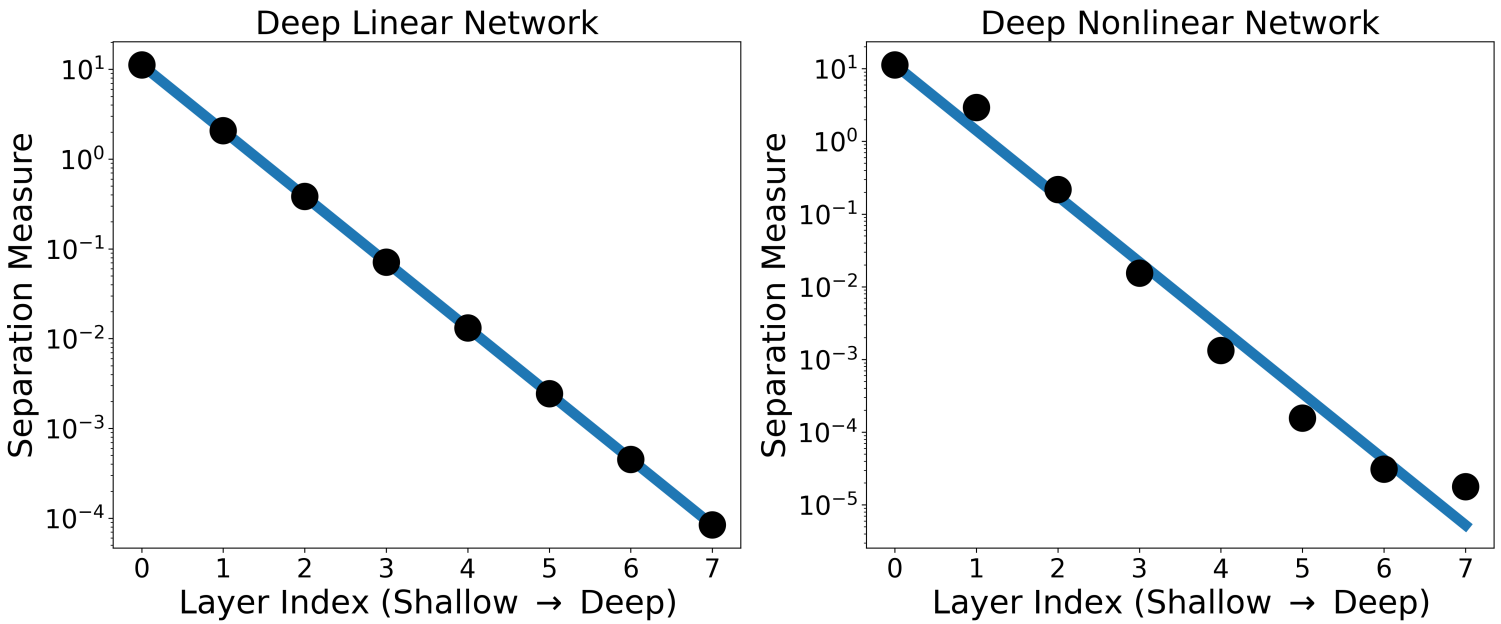
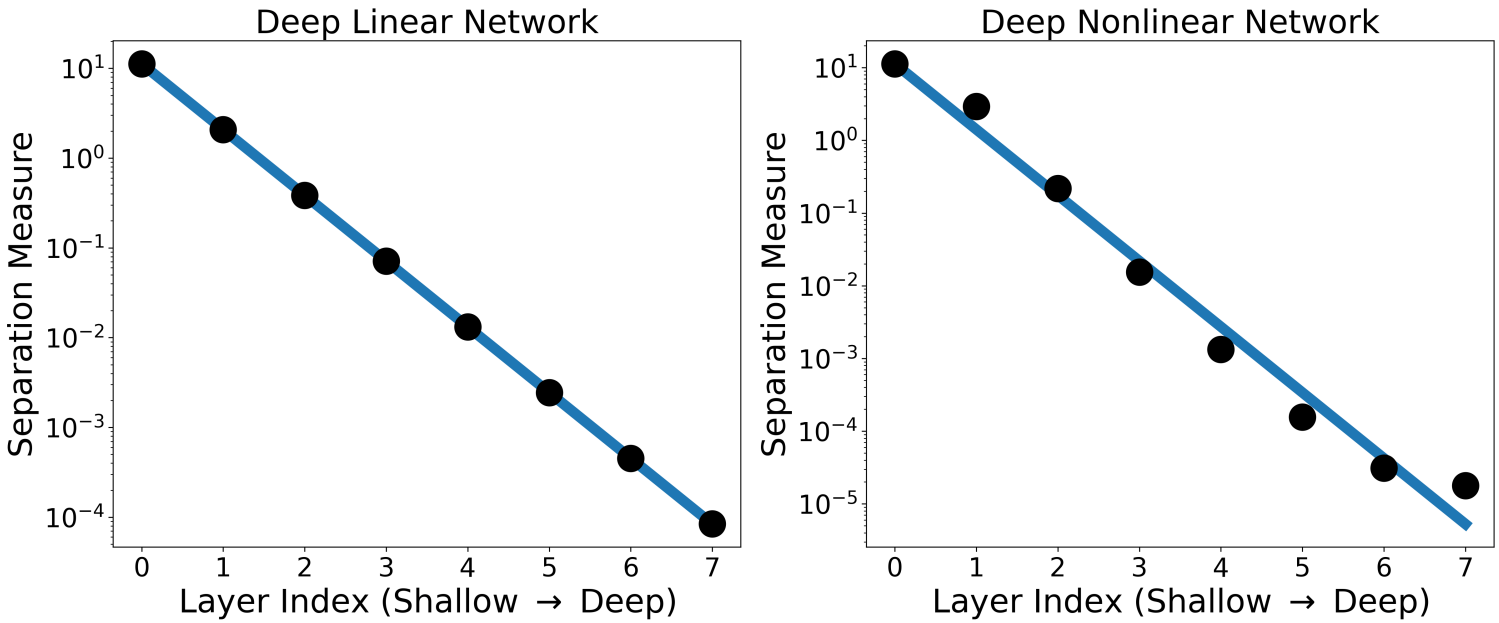


Figure: **Progressive collapse with linear decay[2] on deep linear and nonlinear networks.** The x-axis denotes the layer index and the y-axis denotes the separation measure.

[2] He & Su, A Law of Progressive Separation for Deep Learning, 2022.

Towards Understanding Progressive Collapse?

Given linearly separable data $\boldsymbol{X}$, we measure a certain metric of data separation of each layer's output on linear and nonlinear deep networks:

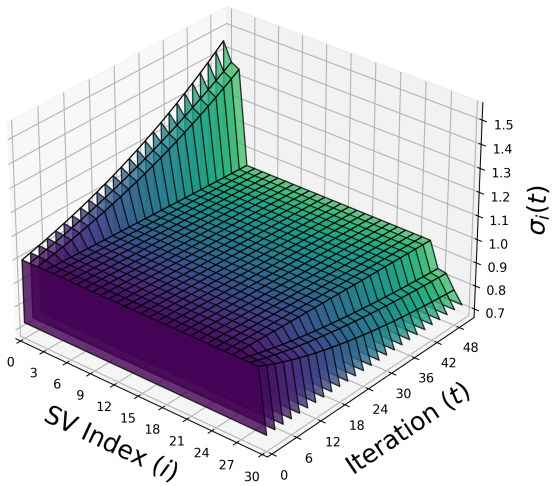


Even training overparameterized deep linear networks could have **infinite many solutions**, why do such benign structures happen?

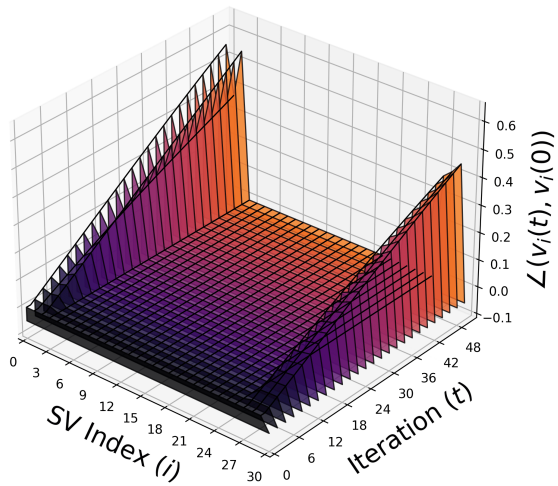
Outline

1. Neural Collapse, Transfer Learning, & Intermediate Layers
2. Law of Parsimony in Gradient Dynamics
3. Progressive Feature Separation in Deep Neural Networks
4. Efficient Deep Matrix Completion
5. Conclusion

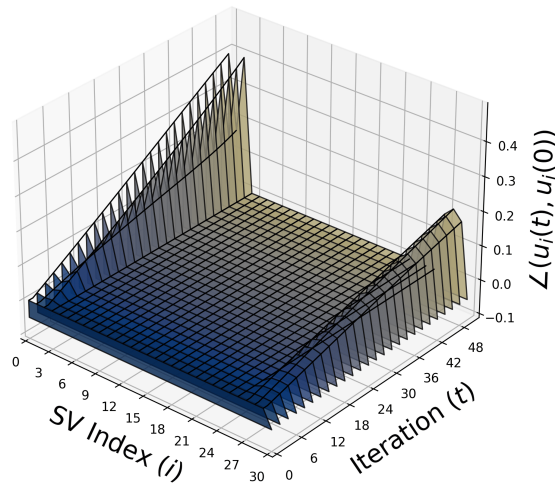
Main Message



Singular Values



Right Singular Vectors

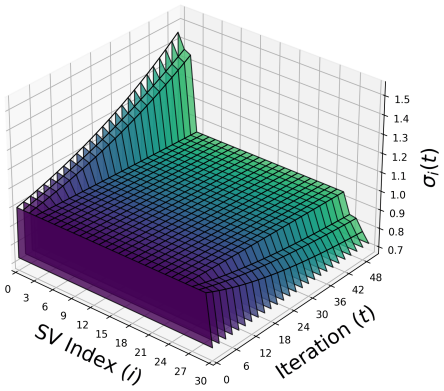


Left Singular Vectors

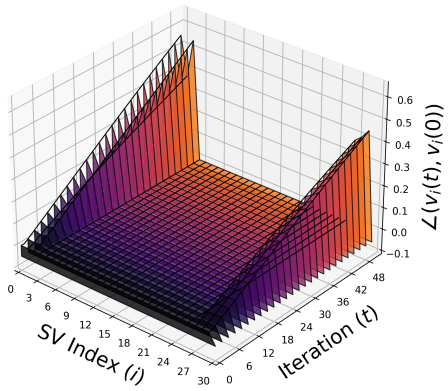
Figure: **Evolution of SVD of the weight matrix $\boldsymbol{W}_1(t) = \boldsymbol{U}_1(t)\boldsymbol{\Sigma}_1(t)\boldsymbol{V}_1(t)^\top$.**

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

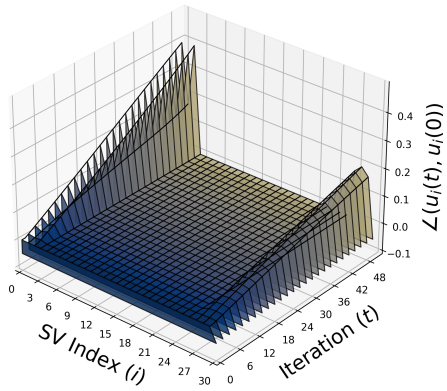
Main Message



Singular Values



Right Singular Vectors



Left Singular Vectors

SCAN ME

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

Setup on 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

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

Setup on 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

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

- **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 $\boldsymbol{\Theta} = \{\boldsymbol{W}_l\}_{l=1}^L$.

Setup on 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

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

- **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 $\boldsymbol{X}\boldsymbol{X}^\top = \boldsymbol{I}_{d_x}$.[3]
- The cross correlation matrix $\boldsymbol{Y}\boldsymbol{X}^\top$ has certain low-dimensional structures (e.g., low-rank or wide matrix).

[3] 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:

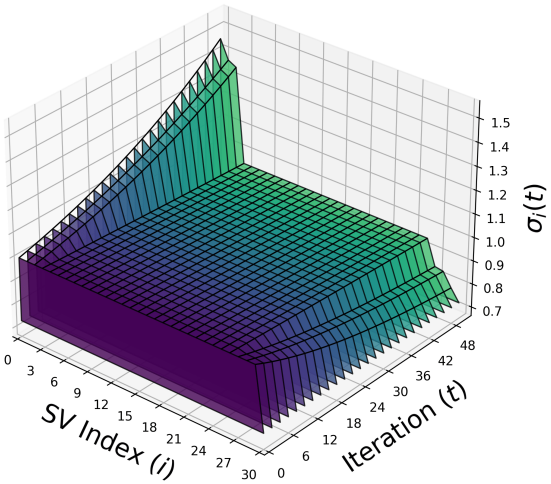
- The weight matrices are square except the last layer, i.e., $d_x = d_1 = d_2 = \cdots = d_{L-1} = d$ for some $d \in \mathbb{N}_+$.
- The input data is *whitened* in the sense that $\boldsymbol{X}\boldsymbol{X}^\top = \boldsymbol{I}_{d_x}$.[3]
- The cross correlation matrix $\boldsymbol{Y}\boldsymbol{X}^\top$ has certain low-dimensional structures (e.g., low-rank or wide matrix).

Under the simplified settings, would GD possess any parsimonious structures during training?

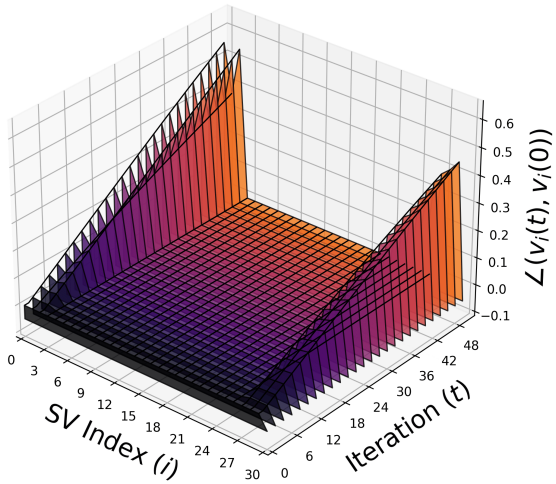
[3] For any full rank $\boldsymbol{X} \in \mathbb{R}^{d_x \times N}$ with $N \geq d_x$, whitened data can always be obtained with a data pre-processing step such as preconditioning.

The Evolution of Singular Spaces in GD Iterates for DLNs

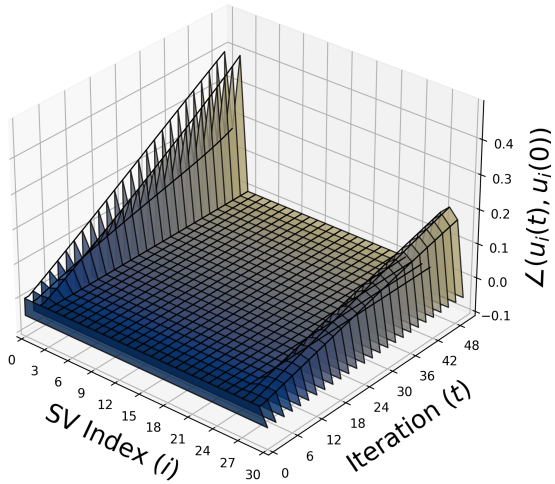
We train a $L = 3$ layer DLN with $d_x = d_y = 30$ and $r := \mathrm{rank}(\boldsymbol{Y}) = 3$.



Singular Values



Right Singular Vectors

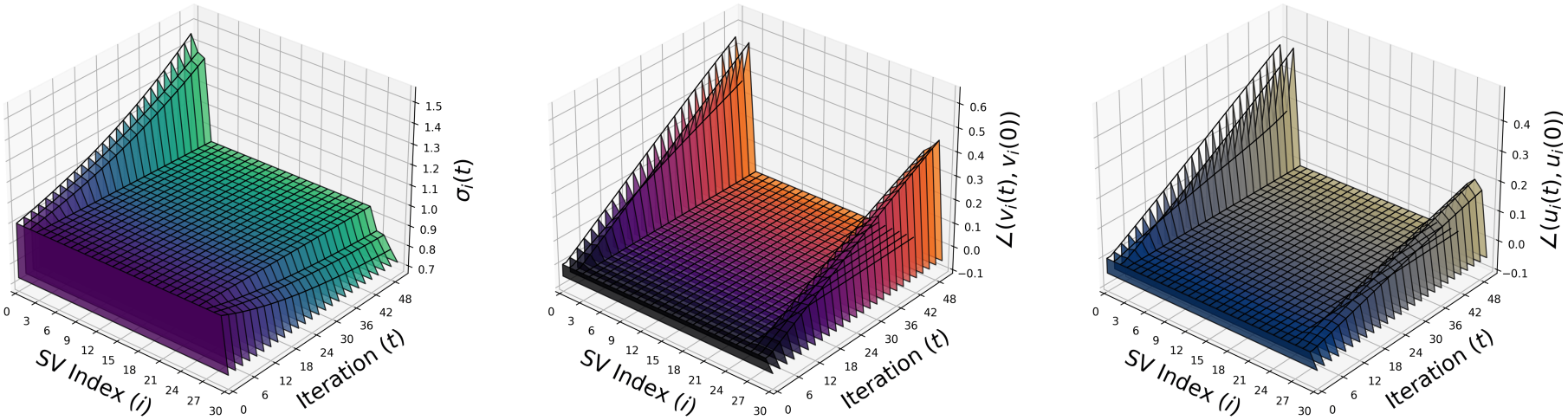


Left Singular Vectors

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

The Evolution of Singular Spaces in GD Iterates for DLNs

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



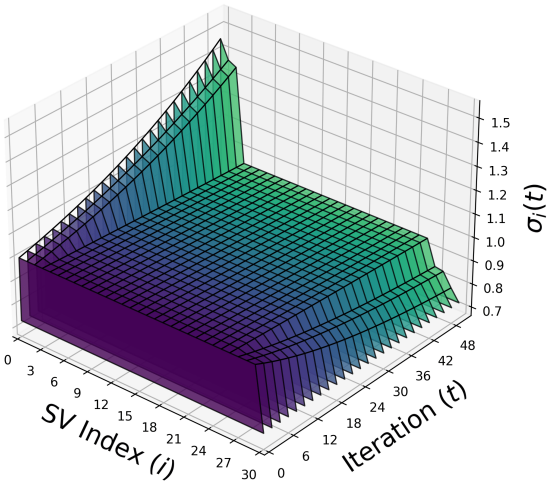
Singular Values | Right Singular Vectors | Left Singular Vectors

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

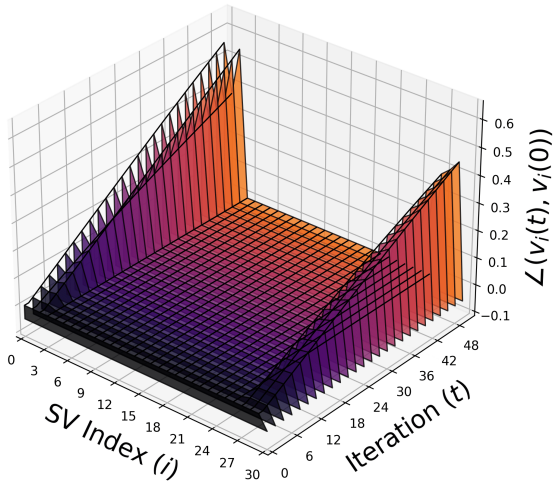
- **Left:** the evolution of singular values $\sigma_{1i}(t)$ throughout training $t \geq 0$;

The Evolution of Singular Spaces in GD Iterates for DLNs

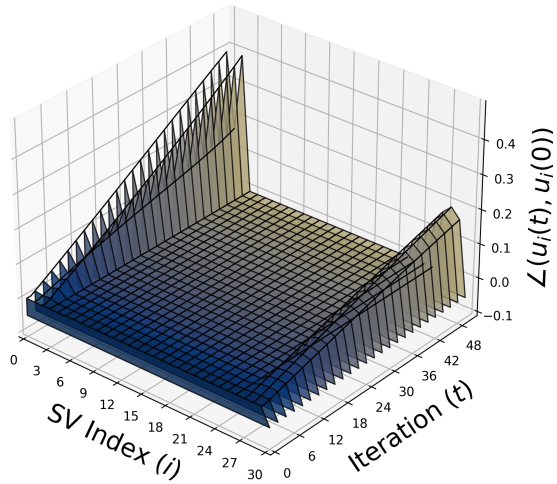
We train a $L = 3$ layer DLN with $d_x = d_y = 30$ and $r := \mathrm{rank}(\boldsymbol{Y}) = 3$.



Singular Values



Right Singular Vectors



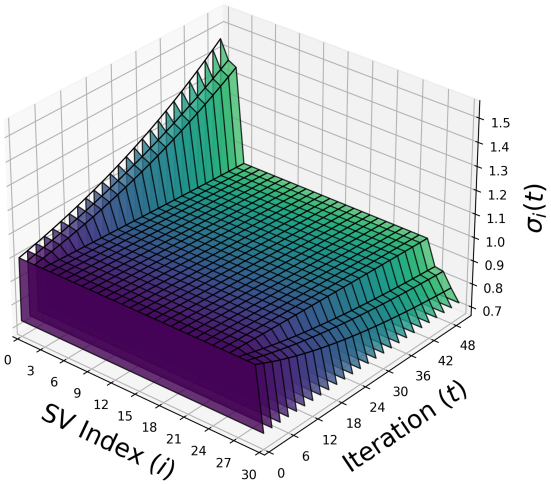
Left Singular Vectors

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

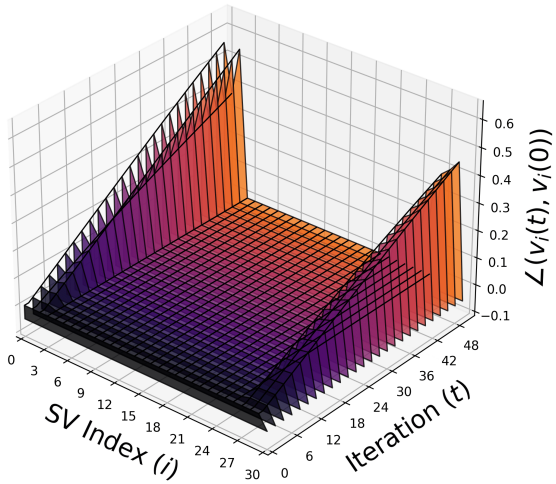
- **Left:** the evolution of singular values $\sigma_{1i}(t)$ throughout training $t \geq 0$;
- **Middle:** the evolution of $\angle(\boldsymbol{v}_{1i}(t), \boldsymbol{v}_{1i}(0))$ throughout training $t \geq 0$;

The Evolution of Singular Spaces in GD Iterates for DLNs

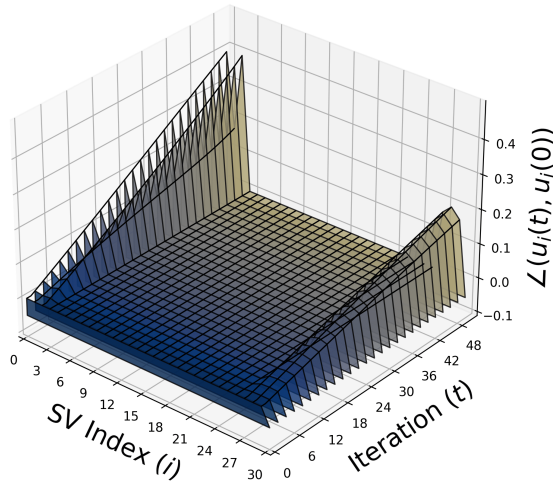
We train a $L = 3$ layer DLN with $d_x = d_y = 30$ and $r := \mathrm{rank}(\boldsymbol{Y}) = 3$.



Singular Values



Right Singular Vectors

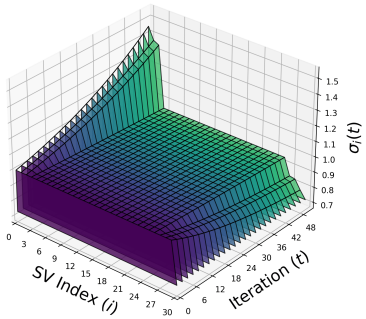


Left Singular Vectors

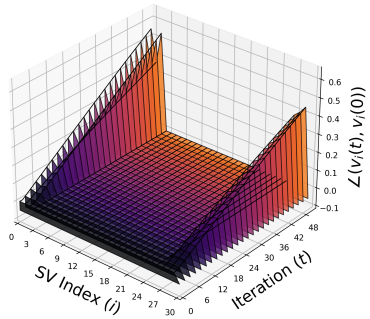
Figure: **Evolution of SVD of the weight matrix $\boldsymbol{W}_1(t) = \boldsymbol{U}_1(t)\boldsymbol{\Sigma}_1(t)\boldsymbol{V}_1(t)^\top$.**

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

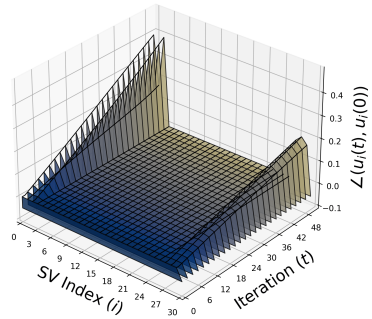
Layer 1



Singular Values

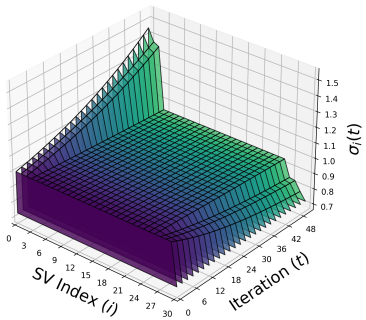


Right Singular Vectors

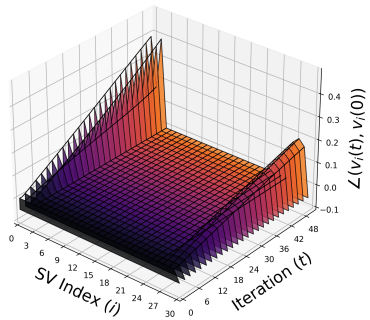


Left Singular Vectors

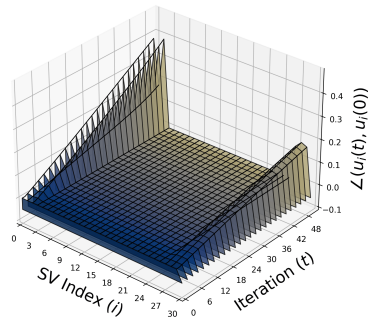
Layer 2



Singular Values

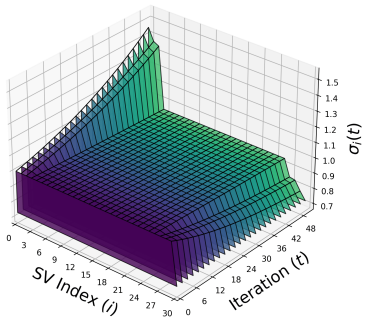


Right Singular Vectors

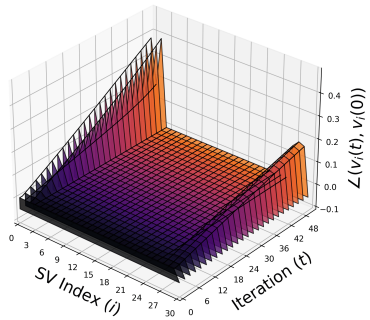


Left Singular Vectors

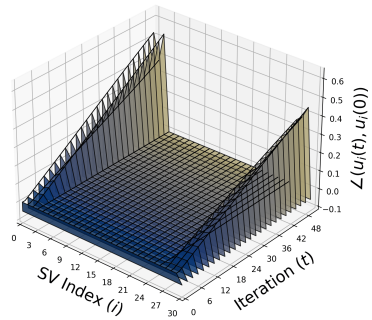
Layer 3



Singular Values

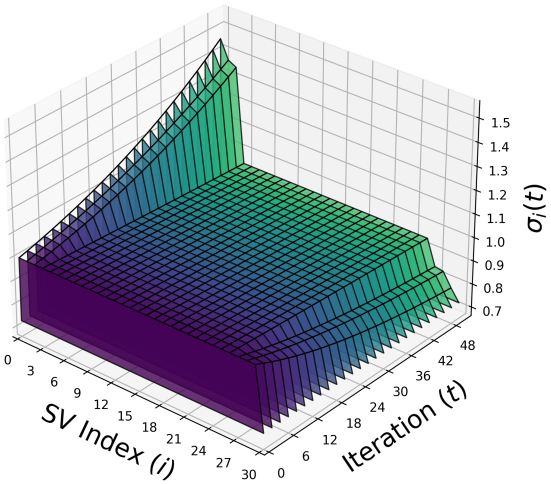


Right Singular Vectors

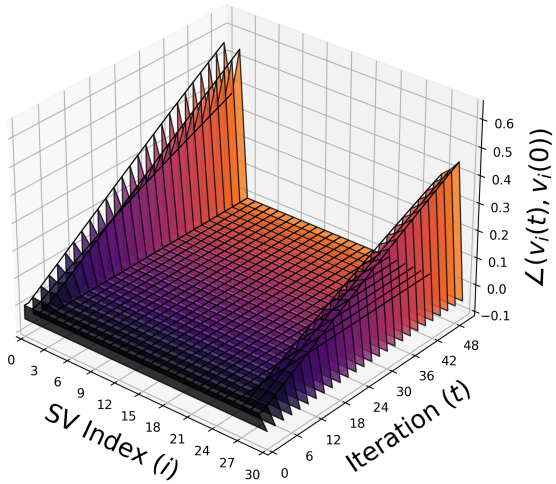


Left Singular Vectors

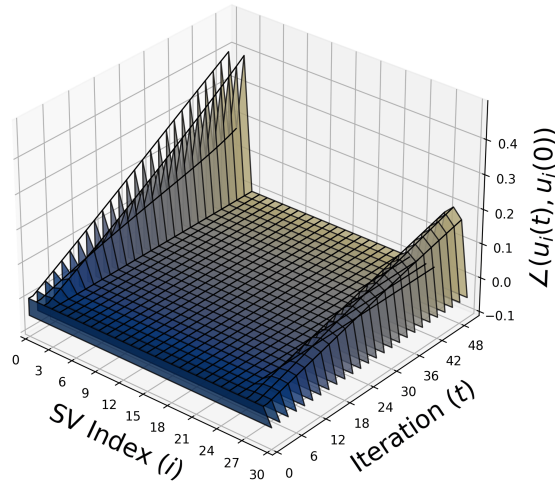
The Evolution of Singular Spaces in GD Iterates for DLNs



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 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 $f_{\boldsymbol{\Theta}}(\cdot)$ 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 $\boldsymbol{Y}\boldsymbol{X}^\top \in \mathbb{R}^{d_y \times d_x}$ with $d_y = r$ and $m := d_x - 2d_y > 0$. 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 $\widetilde{\boldsymbol{W}}_l(0) = \varepsilon \boldsymbol{I}_{2r}$.

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 $\boldsymbol{Y}\boldsymbol{X}^\top \in \mathbb{R}^{d_y \times d_x}$ with $d_y = r$ and $m := d_x - 2d_y > 0$. 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 $\boldsymbol{Y}\boldsymbol{X}^\top \in \mathbb{R}^{d_y \times d_x}$ is of rank $r \in \mathbb{N}_+$ with $d_y = d_x$, and $m = d_x - 2r > 0$. 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

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

[4] 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 $2r$!

[4] 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

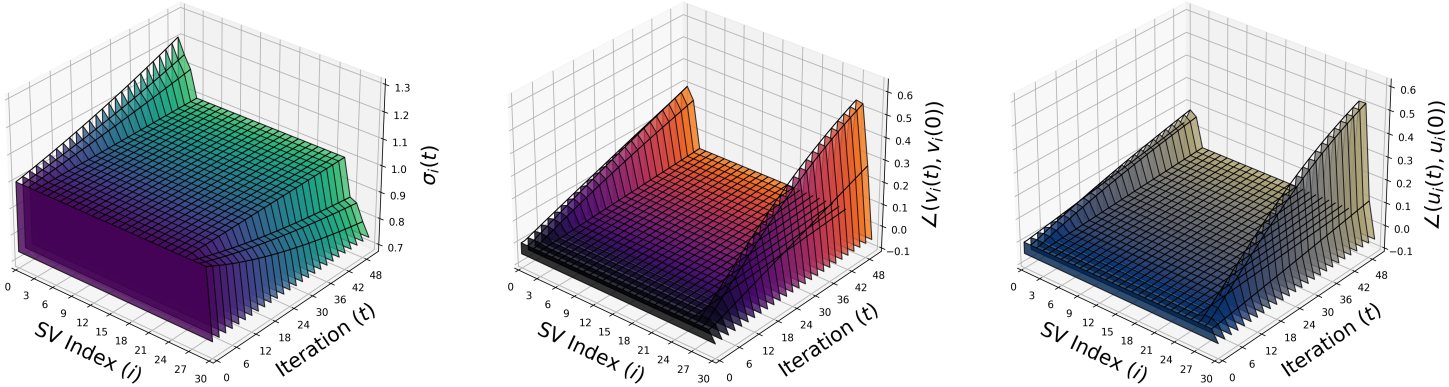
$$\boldsymbol{W}_l(t) = \begin{bmatrix} \boldsymbol{U}_{l,1}\widetilde{\boldsymbol{U}}_l(t) & \boldsymbol{U}_{l,2} \end{bmatrix} \begin{bmatrix} \widetilde{\boldsymbol{\Sigma}}_l(t) & \boldsymbol{0} \\ \boldsymbol{0} & \rho(t)\boldsymbol{I}_m \end{bmatrix} \begin{bmatrix} \boldsymbol{V}_{l,1}\widetilde{\boldsymbol{V}}_l(t)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

- **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.**[4] 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 $2r$).

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

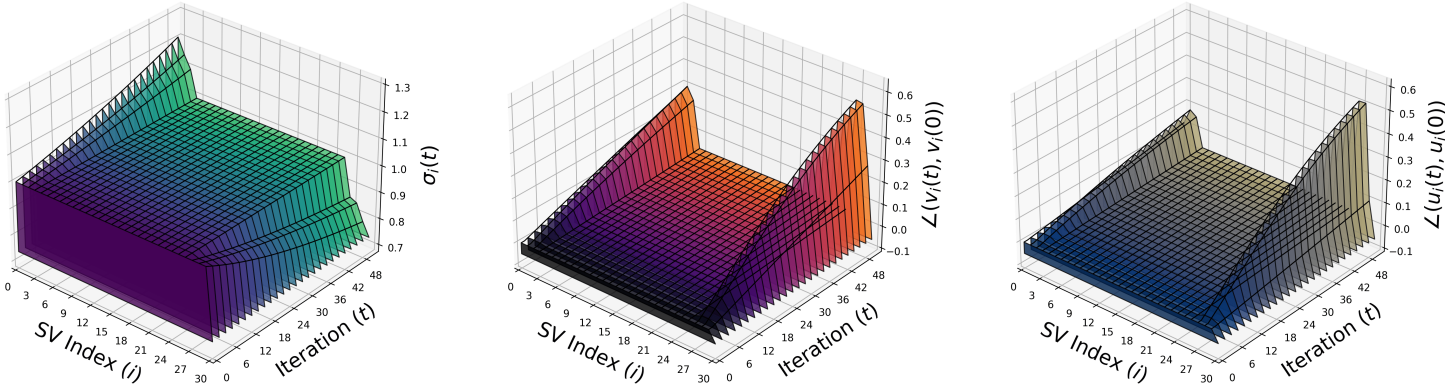
The Evolution of Singular Spaces in More Generic Settings



Singular Values | Right Singular Vectors | Left Singular Vectors

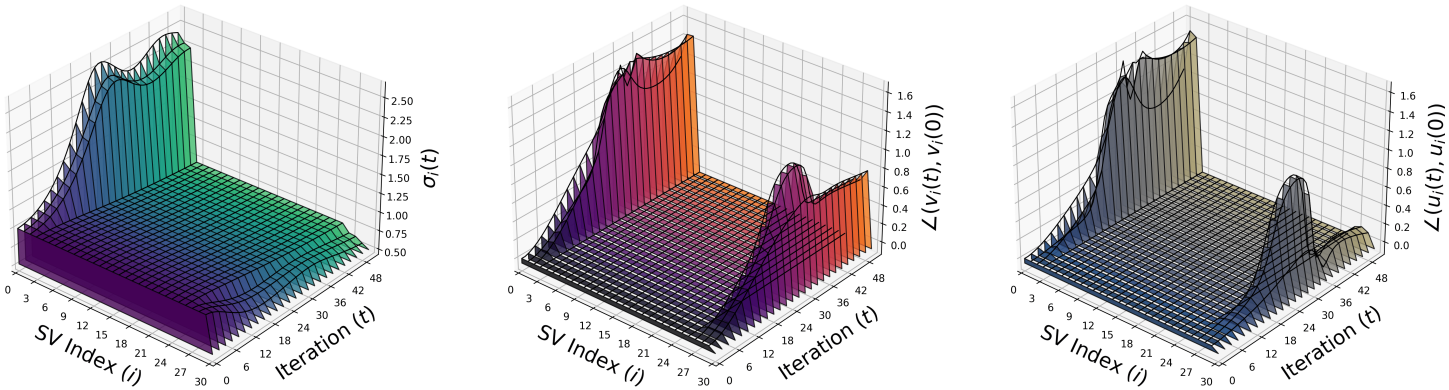
Figure: **Evolution of SVD of weight matrices without whitened data.**

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.**



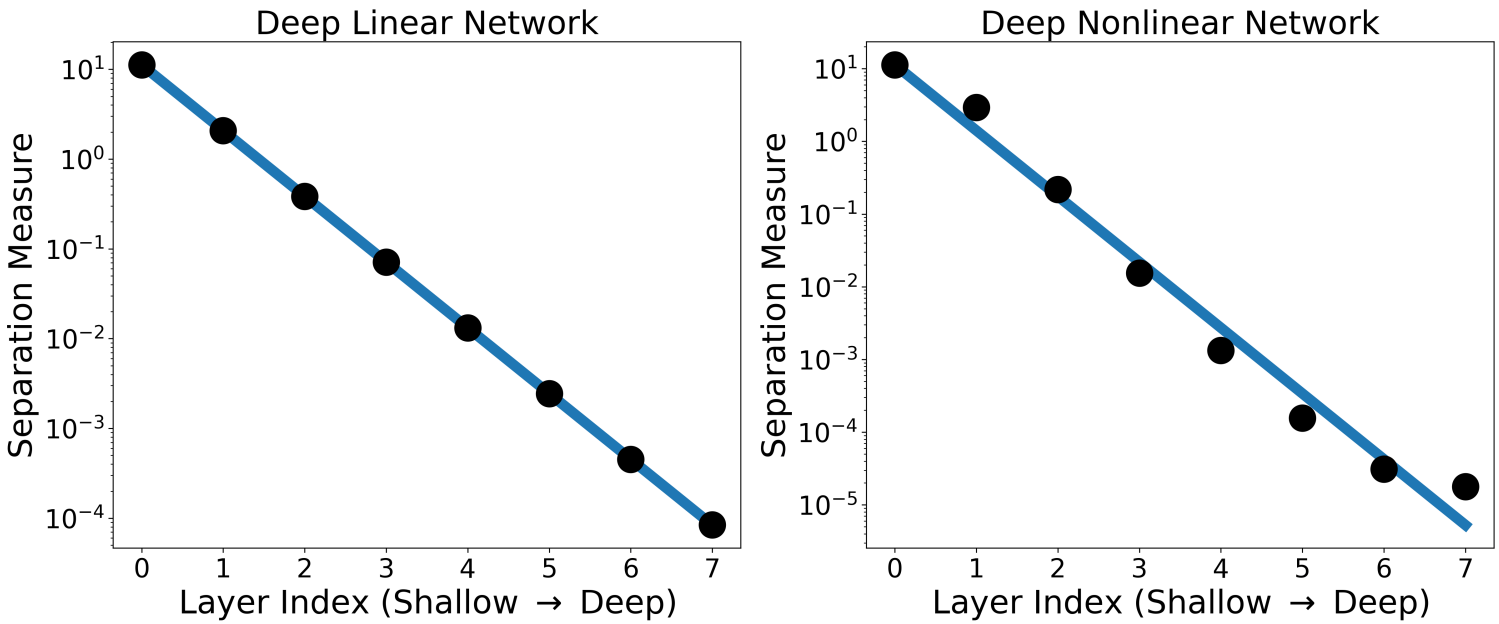
Singular Values | Right Singular Vectors | Left Singular Vectors

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

Outline

1. Neural Collapse, Transfer Learning, & Intermediate Layers
2. Law of Parsimony in Gradient Dynamics
3. Progressive Feature Separation in Deep Neural Networks
4. Efficient Deep Matrix Completion
5. Conclusion

Main Message



The law of parsimony in GD explains progressive feature separation.

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:**

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

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:**

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

- **Measure of Data Separation:** we replace $\mathcal{NC}_1^l = \text{trace}(\boldsymbol{\Sigma}_W^l \boldsymbol{\Sigma}_B^{l\dagger})$ with a simpler measure

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

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:**

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

- **Measure of Data Separation:** we replace $\mathcal{NC}_1^l = \text{trace}(\boldsymbol{\Sigma}_W^l \boldsymbol{\Sigma}_B^{l\dagger})$ with a simpler measure

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

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

Progressive Feature Separation with Linear Decay Rate

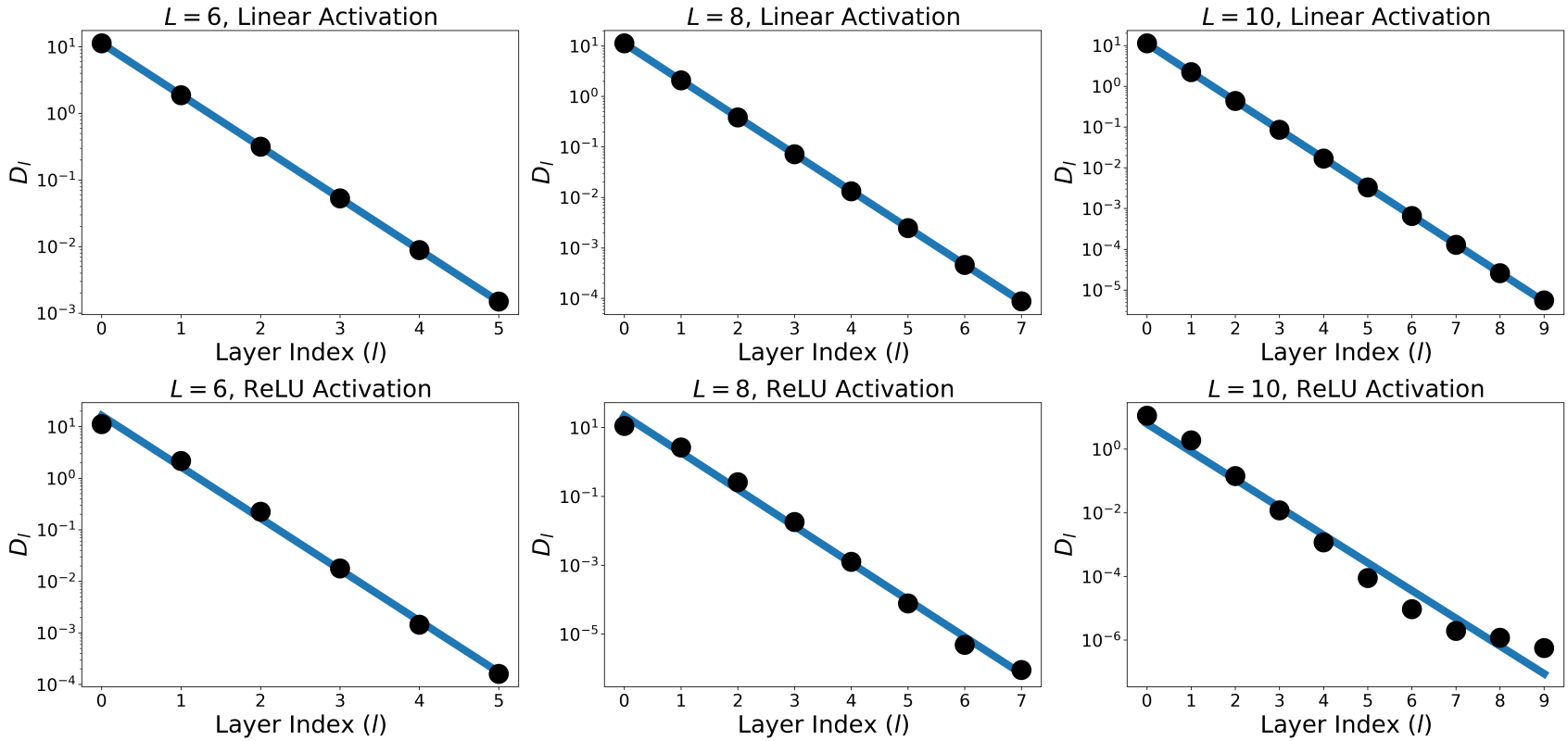


Figure: **Linear decay of feature separation in trained deep networks.**

Progressive Feature Separation with Linear Decay Rate

Theorem (Wang et al.'23)

Suppose we train a L-layer DLN with parameters $\boldsymbol{\Theta} = \{\boldsymbol{W}_l\}_{l=1}^{L}$ via GD with orthogonal initialization of ε-scaling, where input $\boldsymbol{X} \in \mathbb{R}^{d\times N}$ is orthogonal and square and $d_l = d > 2K$. If $\boldsymbol{\Theta}$ satisfies the following:

- **Global Optimality:** $\boldsymbol{W}_{L:1}\boldsymbol{X} = \boldsymbol{Y}$.
- **Balancedness:** *For all weights*

$$\boldsymbol{W}_{l+1}^\top \boldsymbol{W}_{l+1} = \boldsymbol{W}_l \boldsymbol{W}_l^\top, \forall l \in [L-2],$$
$$\|\boldsymbol{W}_L^\top \boldsymbol{W}_L - \boldsymbol{W}_{L-1}\boldsymbol{W}_{L-1}^\top\|_F \le \varepsilon^2\sqrt{d-K}.$$

Progressive Feature Separation with Linear Decay Rate

Theorem (Wang et al.'23)

Suppose we train a L-layer DLN with parameters $\boldsymbol{\Theta} = \{\boldsymbol{W}_l\}_{l=1}^L$ via GD with orthogonal initialization of ε-scaling, where input $\boldsymbol{X} \in \mathbb{R}^{d\times N}$ is orthogonal and square and $d_l = d > 2K$. If $\boldsymbol{\Theta}$ satisfies the following:

- **Global Optimality:** $\boldsymbol{W}_{L:1}\boldsymbol{X} = \boldsymbol{Y}$.
- **Balancedness:** *For all weights*

$$\boldsymbol{W}_{l+1}^\top \boldsymbol{W}_{l+1} = \boldsymbol{W}_l \boldsymbol{W}_l^\top, \forall l \in [L-2],$$
$$\|\boldsymbol{W}_L^\top \boldsymbol{W}_L - \boldsymbol{W}_{L-1}\boldsymbol{W}_{L-1}^\top\|_F \le \varepsilon^2\sqrt{d-K}.$$

- **Unchanged Spectrum:** *There exists an index set $\mathcal{A} \subseteq [d]$ with $|\mathcal{A}| = d - 2K$ such that for all $l \in [L-1]$ that $\sigma_i(\boldsymbol{W}_l) = \varepsilon,\ \forall i \in \mathcal{A}$.*

Progressive Feature Separation with Linear Decay Rate

Theorem (Wang et al.'23)

Suppose we train a L-layer DLN with parameters $\boldsymbol{\Theta} = \{\boldsymbol{W}_l\}_{l=1}^{L}$ via GD with orthogonal initialization of ε-scaling, where input $\boldsymbol{X} \in \mathbb{R}^{d\times N}$ is orthogonal and square and $d_l = d > 2K$. If $\boldsymbol{\Theta}$ satisfies the following:

- **Global Optimality:** $\boldsymbol{W}_{L:1}\boldsymbol{X} = \boldsymbol{Y}$.
- **Balancedness:** *For all weights*

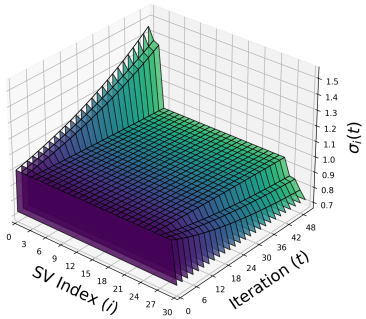
$$\boldsymbol{W}_{l+1}^\top \boldsymbol{W}_{l+1} = \boldsymbol{W}_l \boldsymbol{W}_l^\top, \forall l \in [L-2],$$
$$\|\boldsymbol{W}_L^\top \boldsymbol{W}_L - \boldsymbol{W}_{L-1}\boldsymbol{W}_{L-1}^\top\|_F \le \varepsilon^2\sqrt{d-K}.$$

- **Unchanged Spectrum:** *There exists an index set $\mathcal{A} \subseteq [d]$ with $|\mathcal{A}| = d - 2K$ such that for all $l \in [L-1]$ that $\sigma_i(\boldsymbol{W}_l) = \varepsilon,\ \forall i \in \mathcal{A}$.*

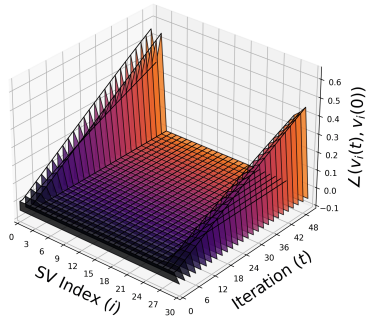
Then, it holds for all $l = 0, 1, \ldots, L-2$ that

$$D_{l+1}/D_l \le 2(\sqrt{K}+1)\varepsilon^2.$$

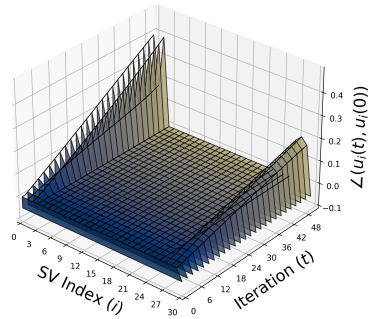
Layer 1



Singular Values

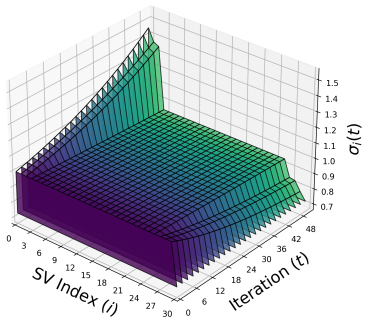


Right Singular Vectors

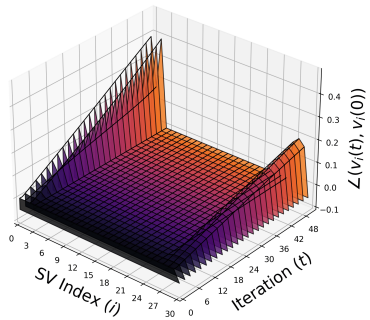


Left Singular Vectors

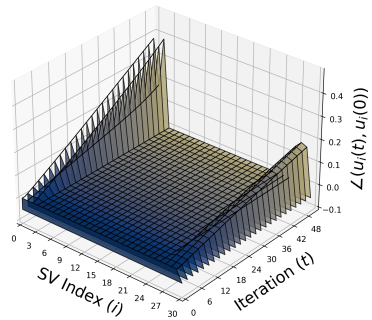
Layer 2



Singular Values

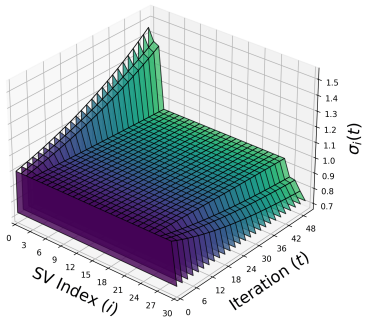


Right Singular Vectors

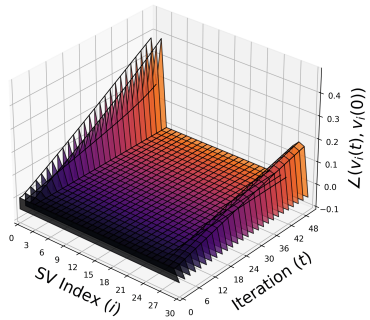


Left Singular Vectors

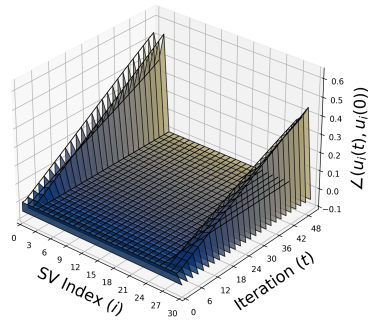
Layer 3



Singular Values



Right Singular Vectors



Left Singular Vectors

Effects of Initialization Scale ε

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

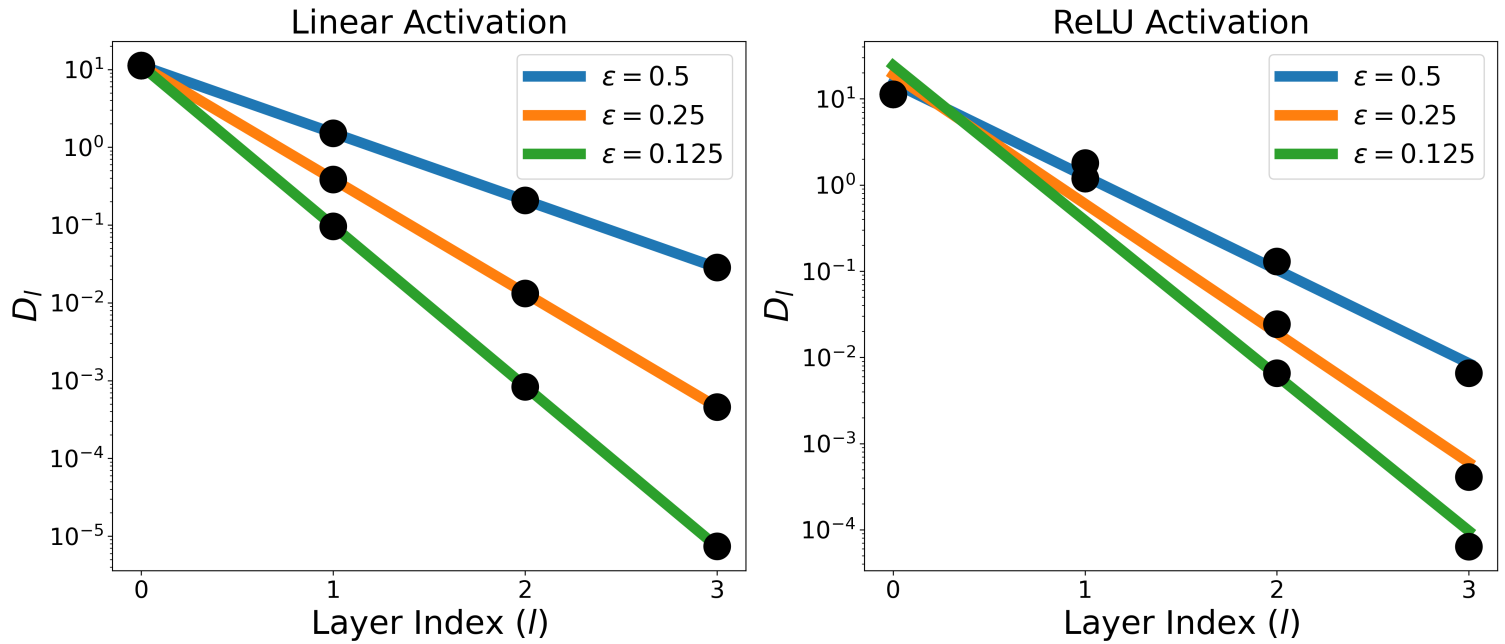


Figure: **Linear decay of feature separation measure D_l in trained deep networks with varying initialization scale ε.**

Tradeoffs Between Decay Rate and Convergence

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

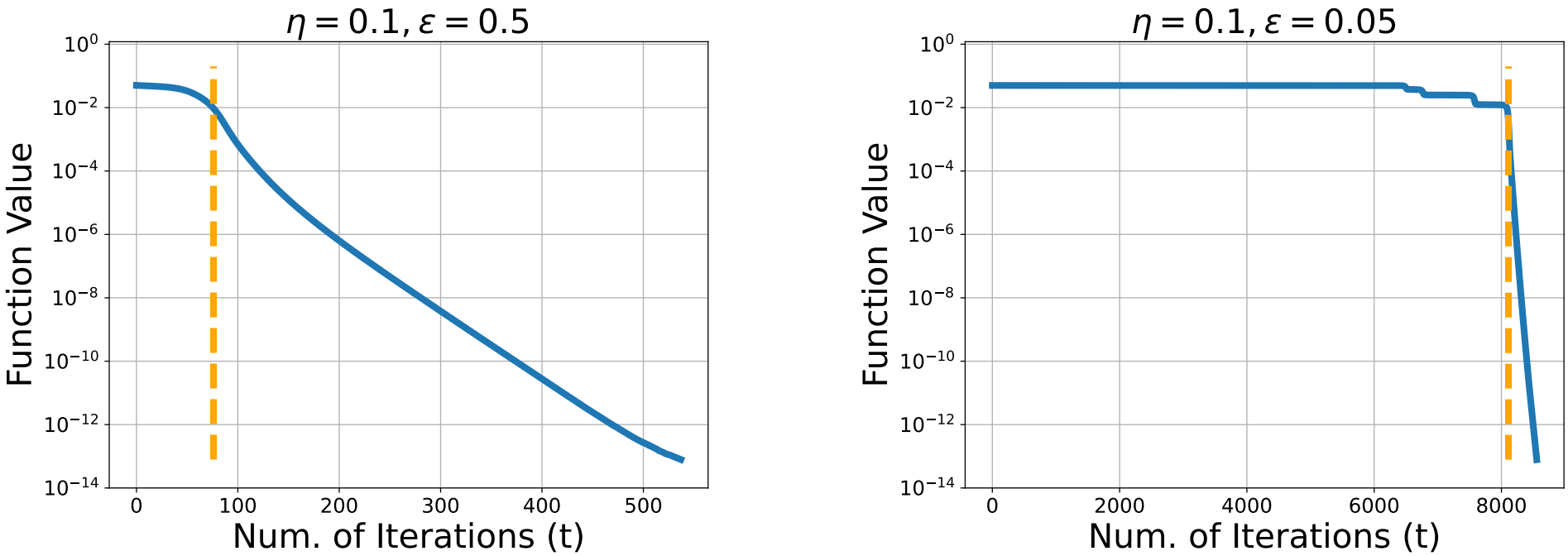


Figure: **The dynamics of GD for DLNs with learning rate $\eta = 0.1$.**

Is the Orthogonal Initialization Critical?

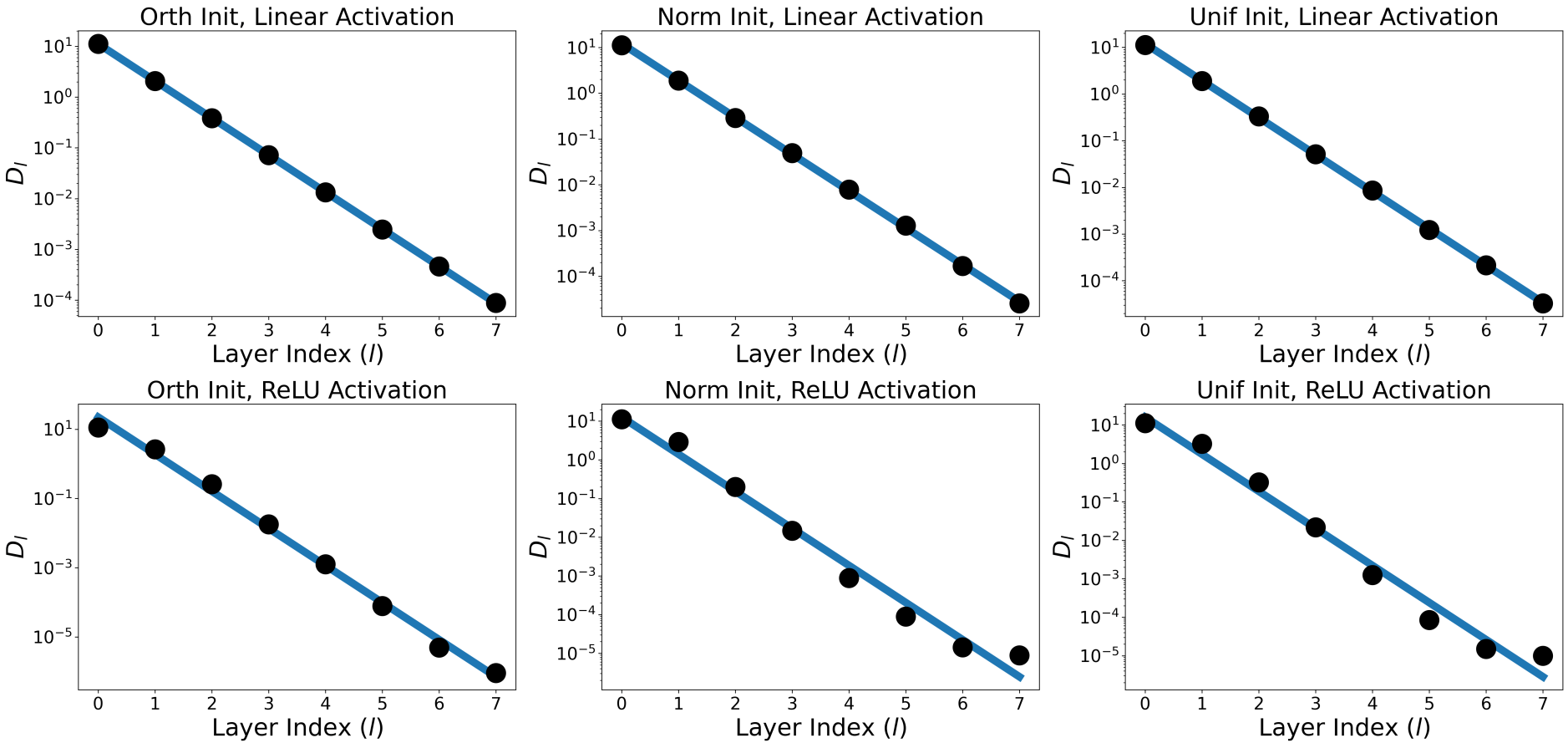


Figure: **Linear decay of feature separation in trained DLNs with different initialization types (left to right: Orth., Norm, Unif).**

Outline

1. Neural Collapse, Transfer Learning, & Intermediate Layers
2. Law of Parsimony in Gradient Dynamics
3. Progressive Feature Separation in Deep Neural Networks
4. Efficient Deep Matrix Completion
5. Conclusion

Main Message

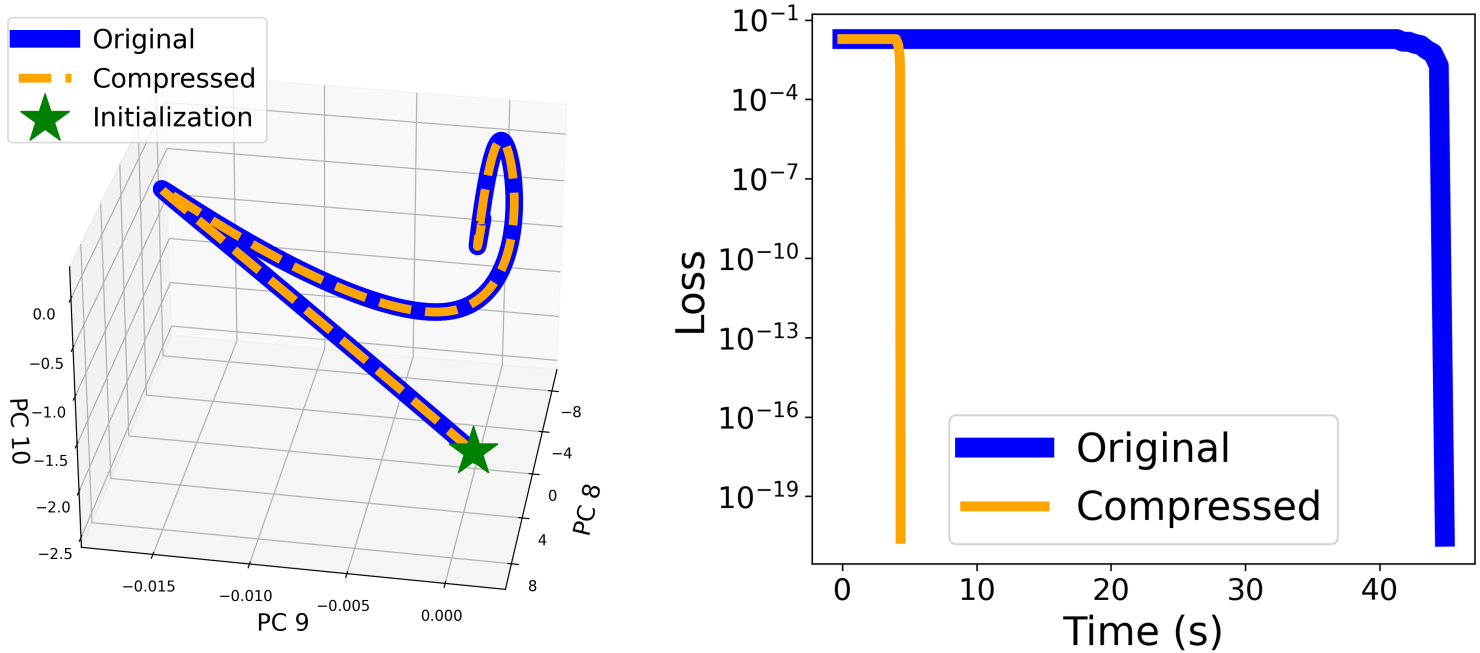


Figure: **Efficient training of deep linear networks.**

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

Problem Setup for Deep Matrix Completion

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

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

Problem Setup for Deep Matrix Completion

Consider recovering $\boldsymbol{\Phi} \in \mathbb{R}^{d\times d}$ with $r := \mathrm{rank}(\boldsymbol{\Phi}) \ll d$ 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.

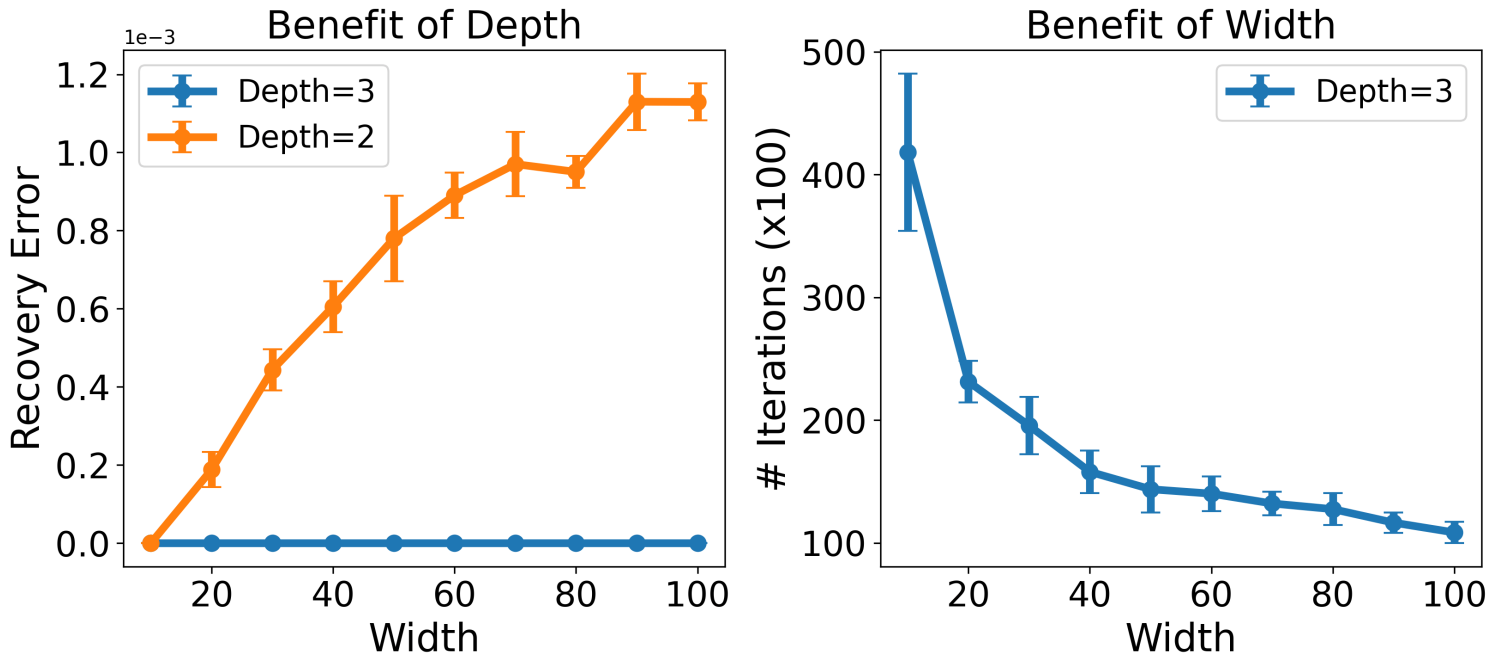
Problem Setup for Deep Matrix Completion

Consider recovering $\boldsymbol{\Phi} \in \mathbb{R}^{d\times d}$ with $r := \mathrm{rank}(\boldsymbol{\Phi}) \ll d$ 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.
- If the network depth $L = 2$, it reduces to the Burer-Monteiro factorization formulation.

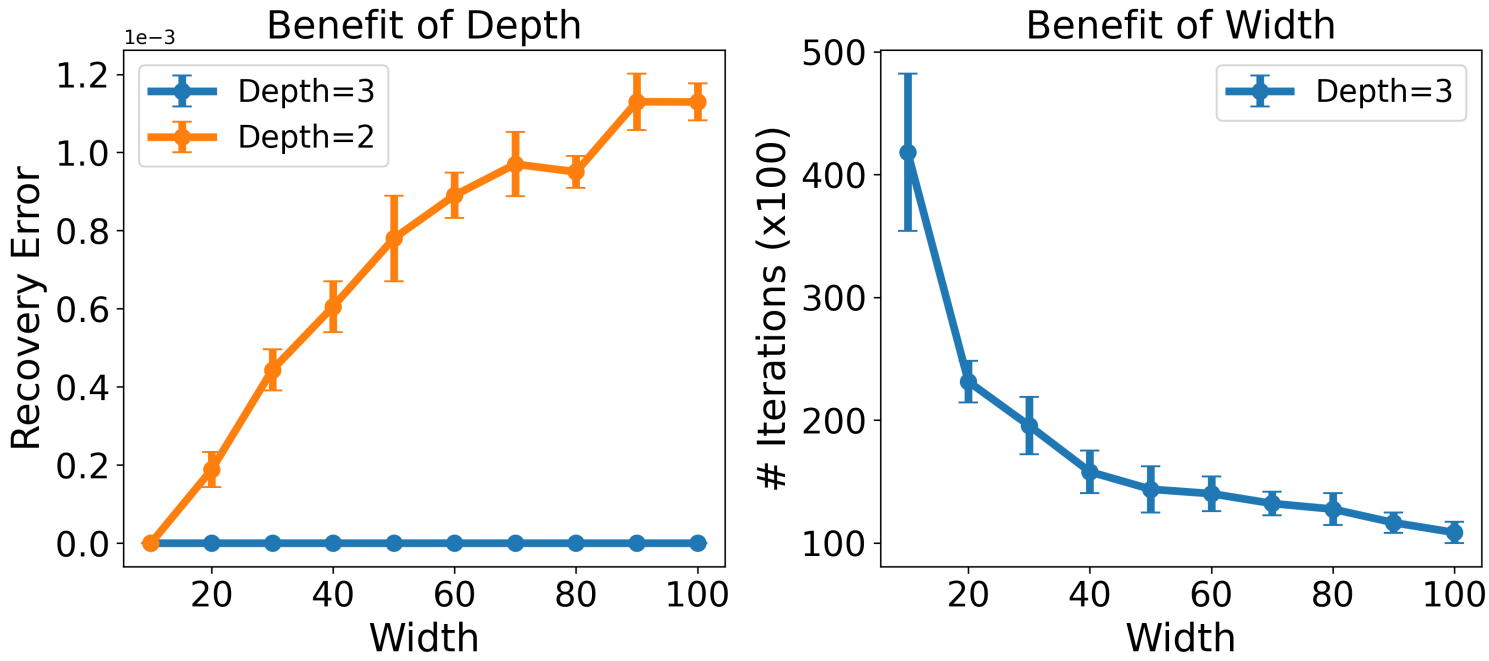
Why Deep Matrix Factorization and Overparameterization?



- **Benefits of Depth (Left):** Improved sample complexity[5] and less prone to overfitting.

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

Why Deep Matrix Factorization and Overparameterization?



- **Benefits of Depth (Left):** Improved sample complexity[5] and less prone to overfitting.
- **Benefits of Width (Right):** Increasing the width of the network results in accelerated convergence in terms of iterations.

[5] 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

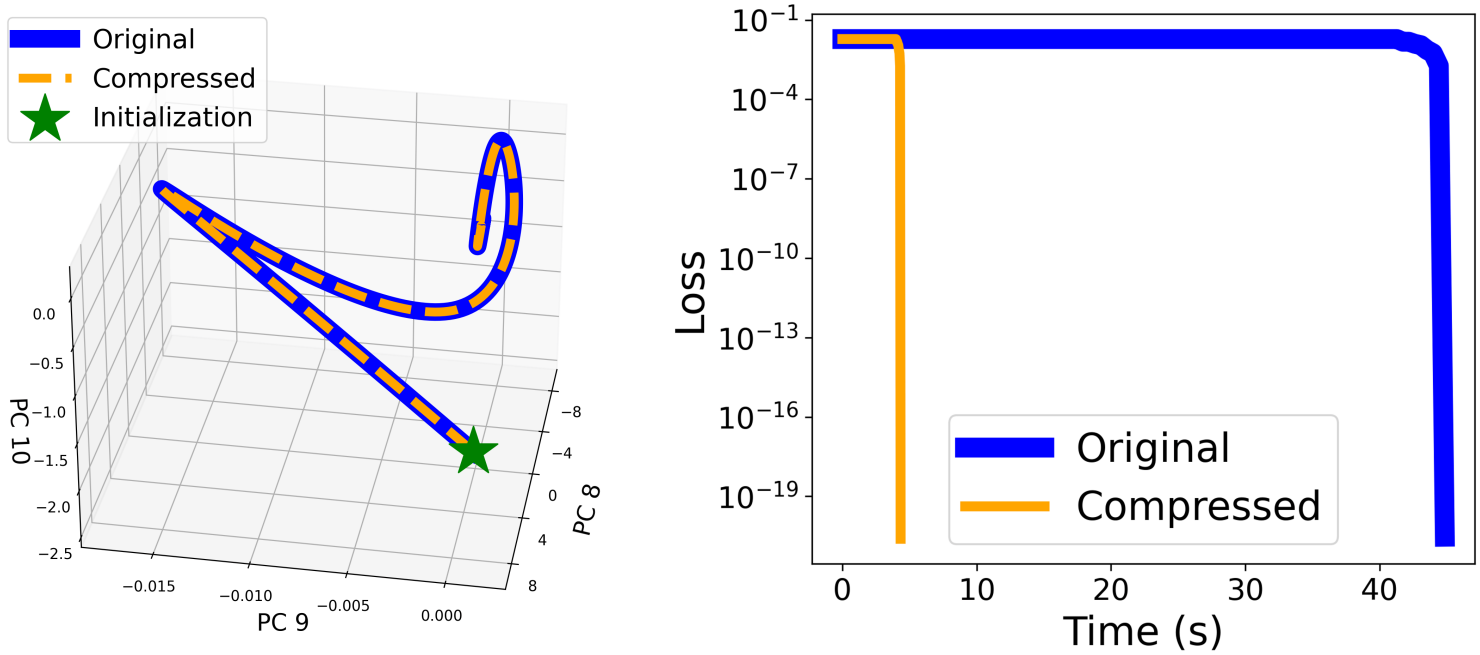


Figure: **Efficient training of deep linear networks.**

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:

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

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

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

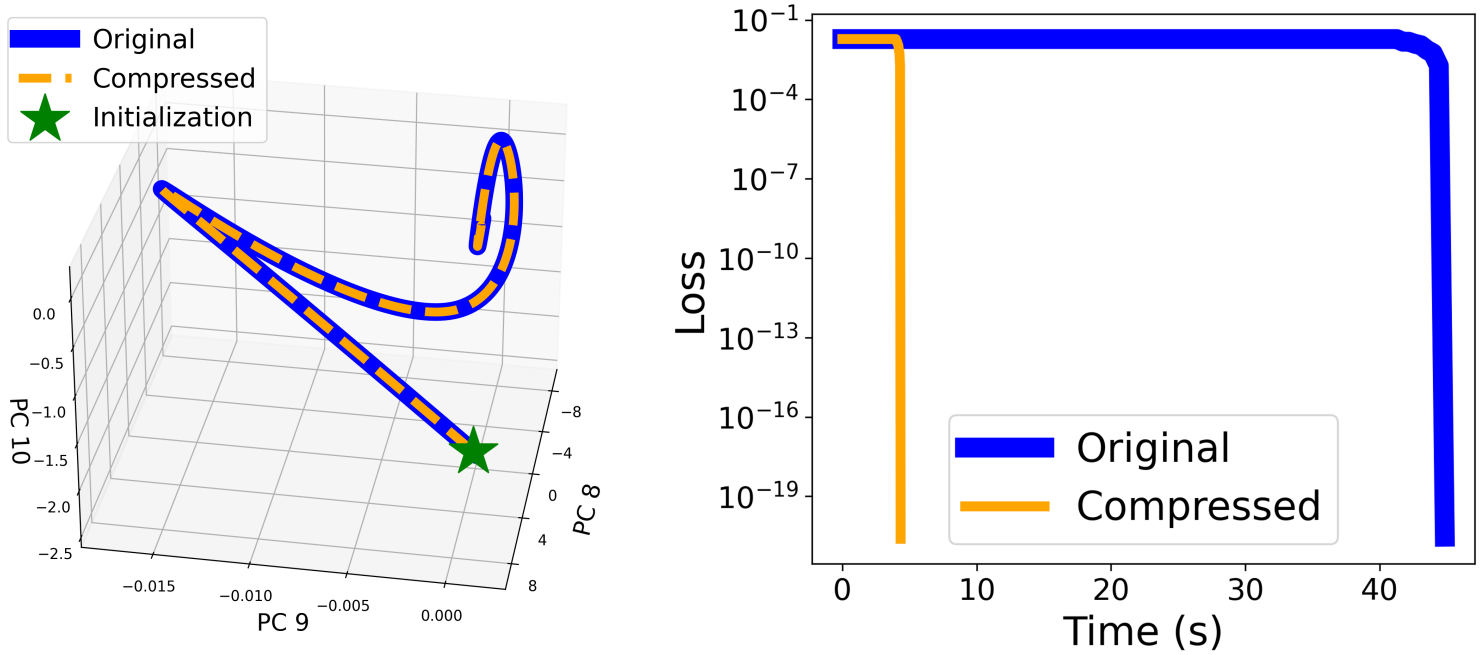
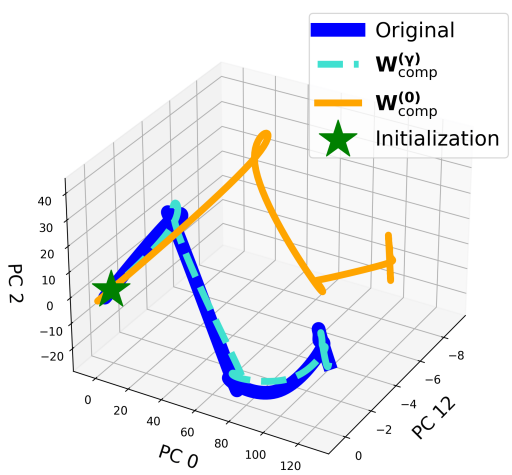
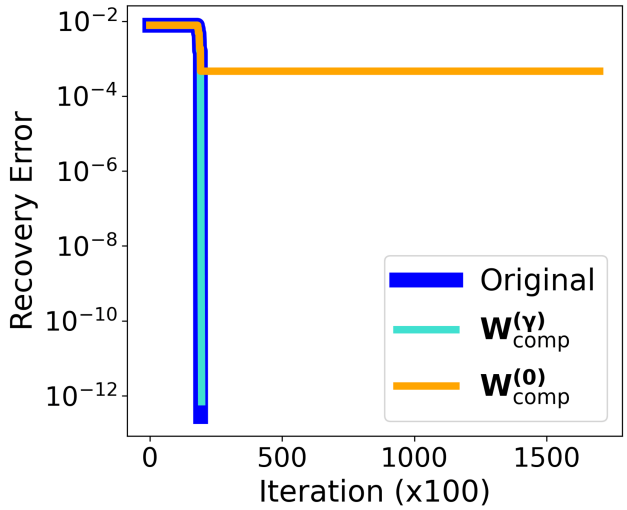


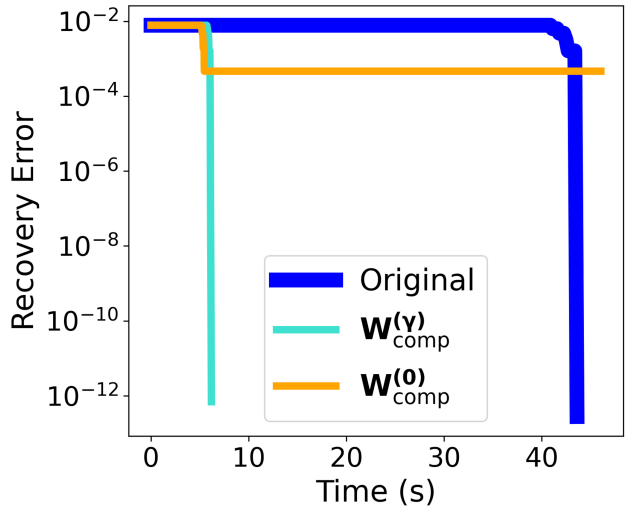
Figure: **Efficient training of deep linear networks.**

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

From Deep Matrix Factorization to Completion?

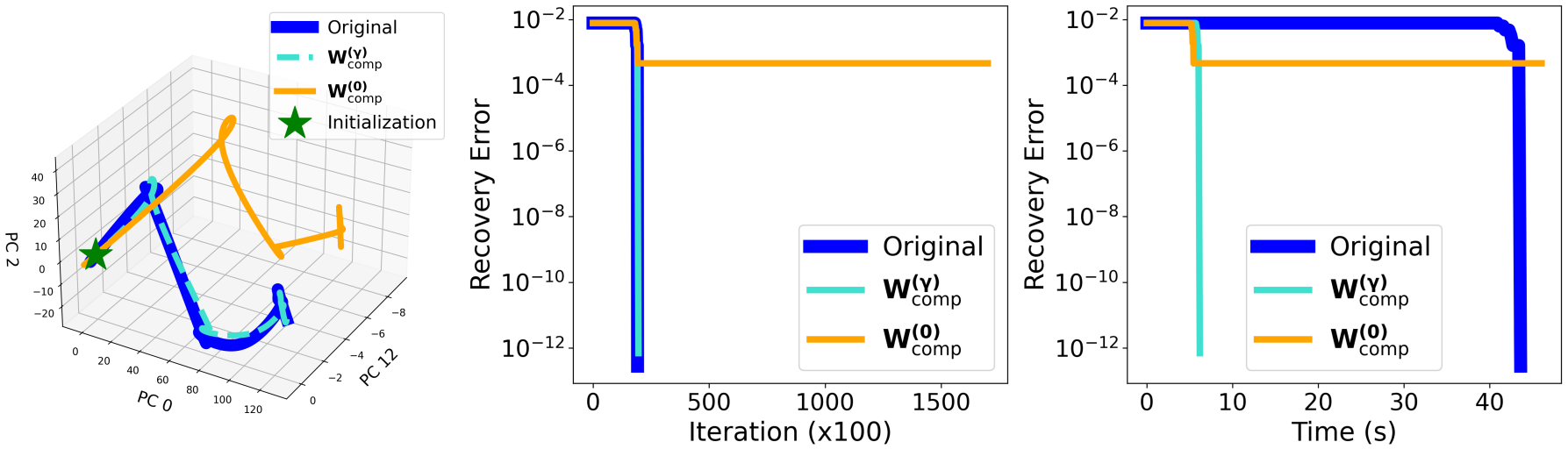






- However, directly applying our approach from deep matrix factorization to completion does not work well...

From Deep Matrix Factorization to Completion?

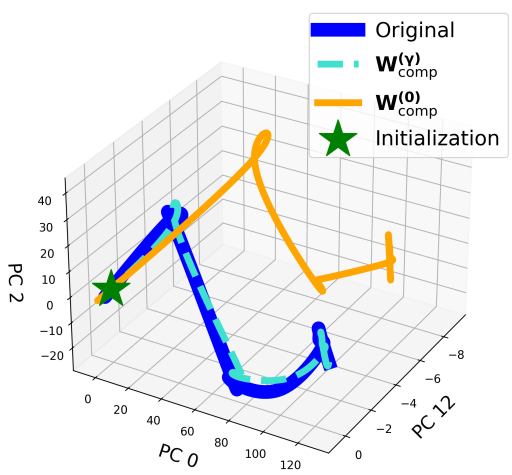


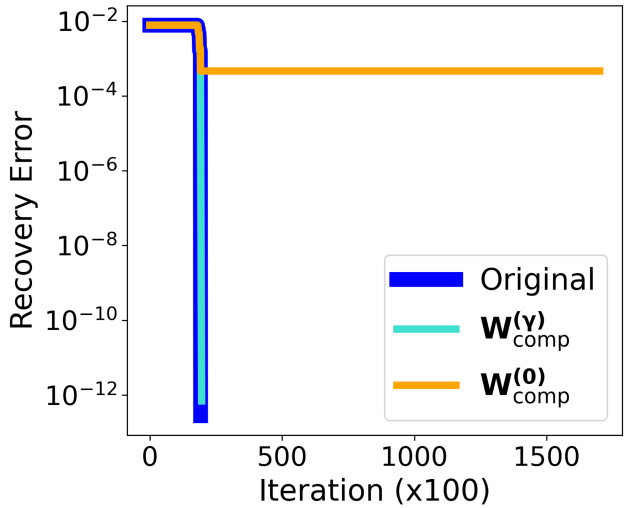
- However, directly applying our approach from deep matrix factorization to completion does not work well...
- This is due to the fact that the law of parsimony in GD:

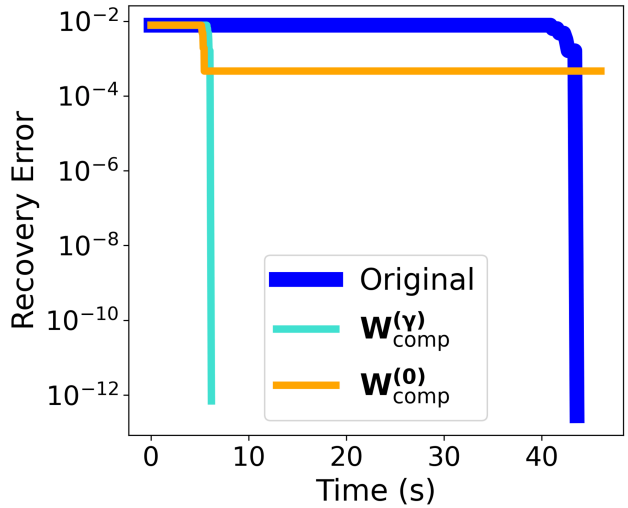
$$\boldsymbol{W}_{L:1}(t) \;\approx\; \boldsymbol{U}_{L,1}\widetilde{\boldsymbol{W}}_{L:1}(t)\boldsymbol{V}_{1,1}^{\top}, \quad \forall t \geq 0,$$

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

From Deep Matrix Factorization to Completion?





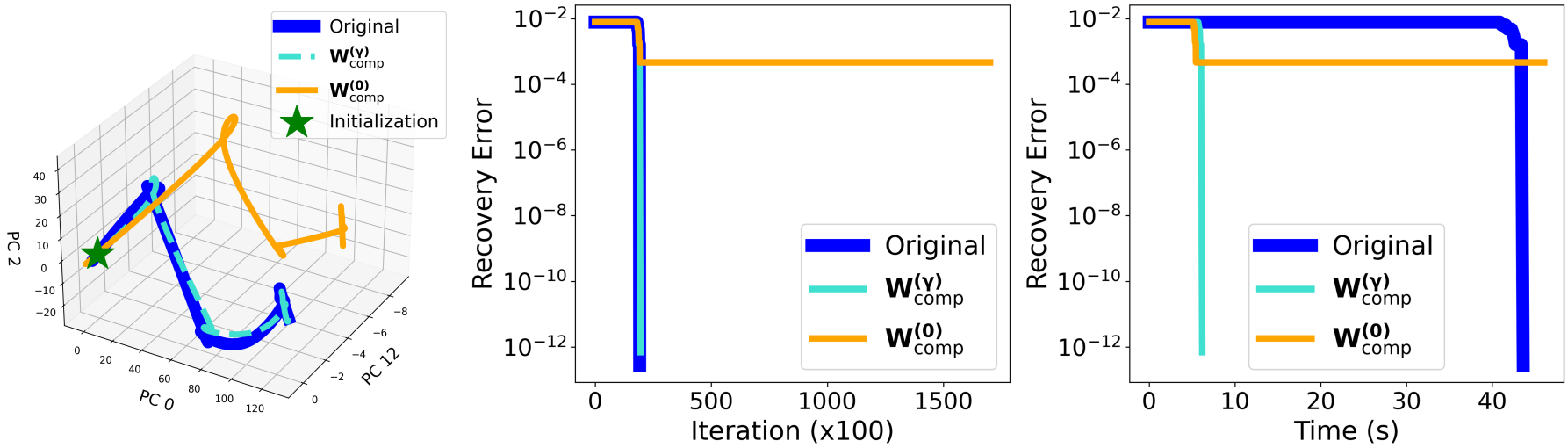


- **Remedy:** update both $\boldsymbol{V}_{1,1}(t)$ and $\boldsymbol{U}_{L,1}(t)$ factors via GD with a **discrepant** learning rate $\gamma\eta$ in the "compressed network":[6]

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

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

From Deep Matrix Factorization to Completion?



- **Remedy:** update both $\boldsymbol{V}_{1,1}(t)$ and $\boldsymbol{U}_{L,1}(t)$ factors via GD with a **discrepant** learning rate $\gamma\eta$ in the "compressed network":[6]

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

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

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

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 $d \gg \widehat{r}$?

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 $d \gg \widehat{r}$?

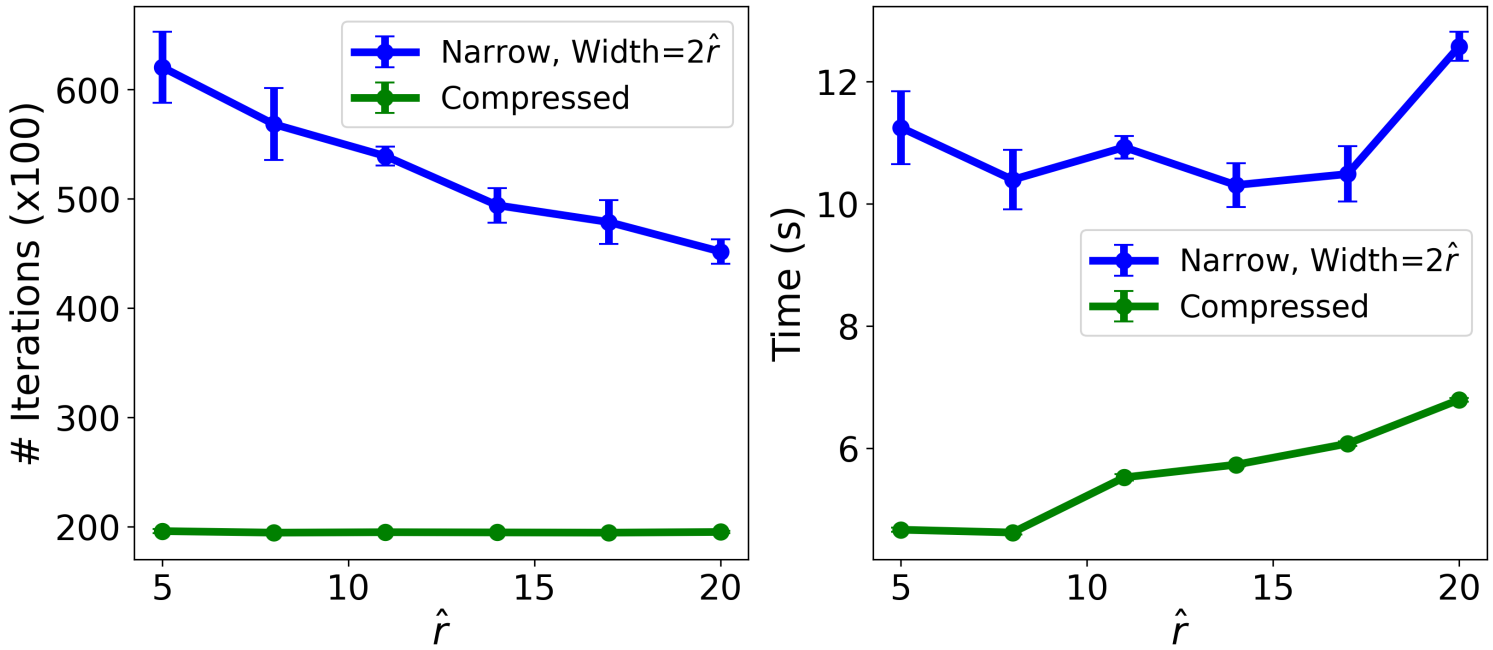


Figure: **Efficiency of compressed networks vs. narrow network.**

Compressed Networks vs. Narrow Networks?

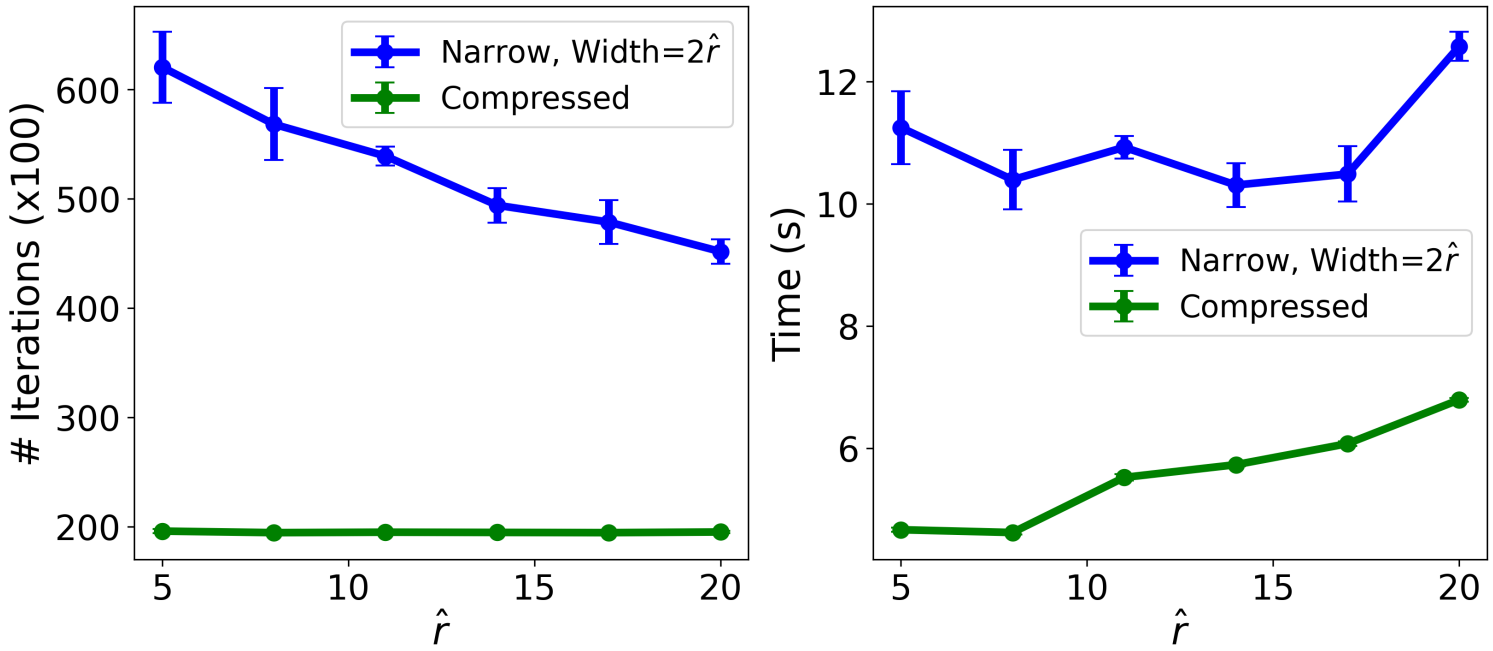


Figure: **Efficiency of compressed networks vs. narrow network.**

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

Outline

1. Neural Collapse, Transfer Learning, & Intermediate Layers
2. Law of Parsimony in Gradient Dynamics
3. Progressive Feature Separation in Deep Neural Networks
4. Efficient Deep Matrix Completion
5. Conclusion

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 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.

3 Wang, P., Yaras, C., Li, X., Hu, W., Zhu, Z., Balzano, L., Qu, Q. (2023). Understanding Hierarchical Representation Learning in Deep Networks via Neural Collapse. Working paper.

4 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.

Conclusion and Coming Attractions

Learning common deep networks for low-dim structure

- **Low-dimensional features**: understand low-dim. features (sparse and neural collapse (NC)) learned in deep classifiers trained with one-hot labeling based losses
- **Law-of-parsimony in GD**: efficient network compression & training, and understanding intermediate layers of deep networks

Thank You! Questions?

Call for Papers

- IEEE JSTSP Special Issue on Seeking Low-dimensionality in Deep Neural Networks (SLowDNN) Manuscript Due: **Nov. 30, 2023**.
- Conference on Parsimony and Learning (CPAL) January 2024, Hongkong, Manuscript Due: **Aug. 28, 2023**.