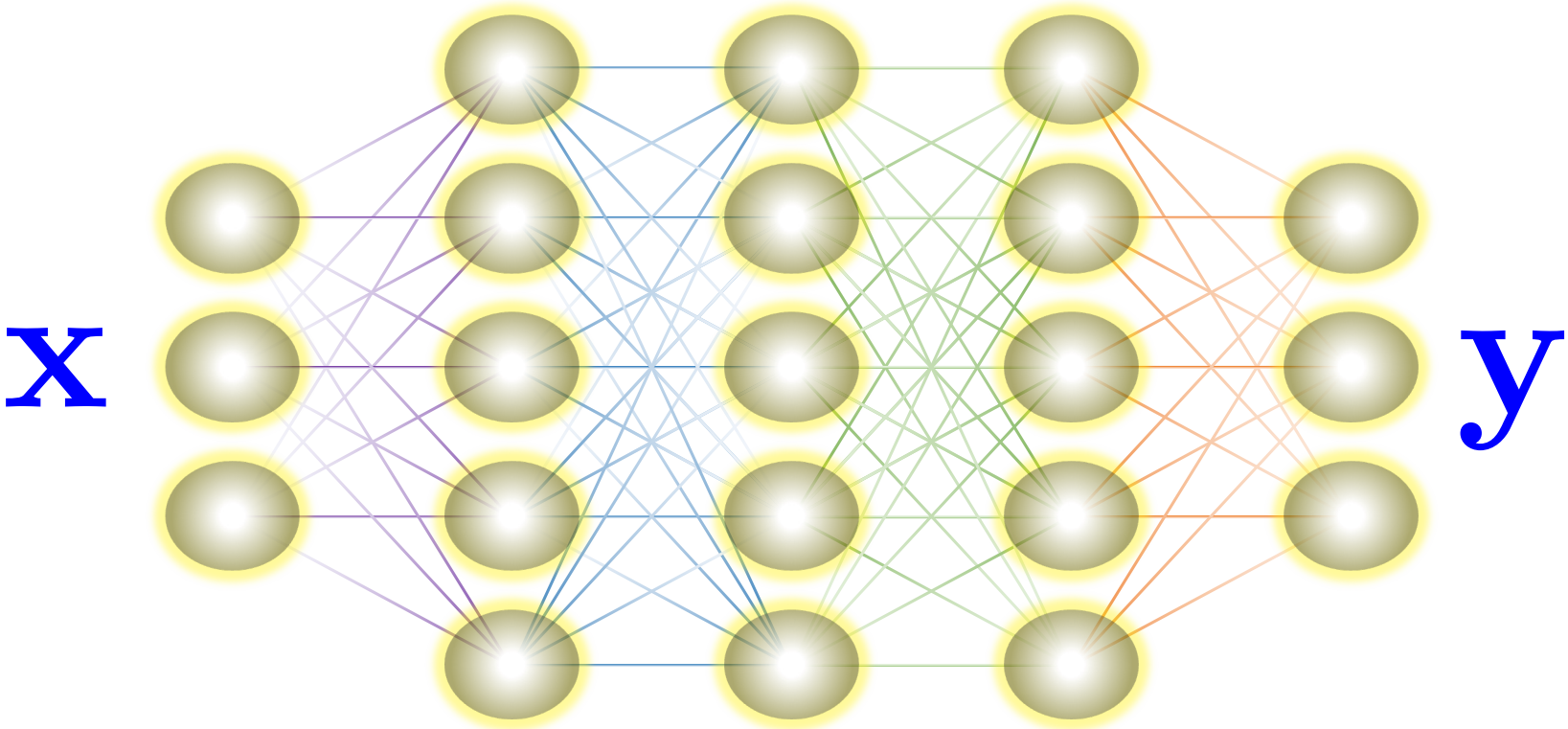
Optimization for Learning

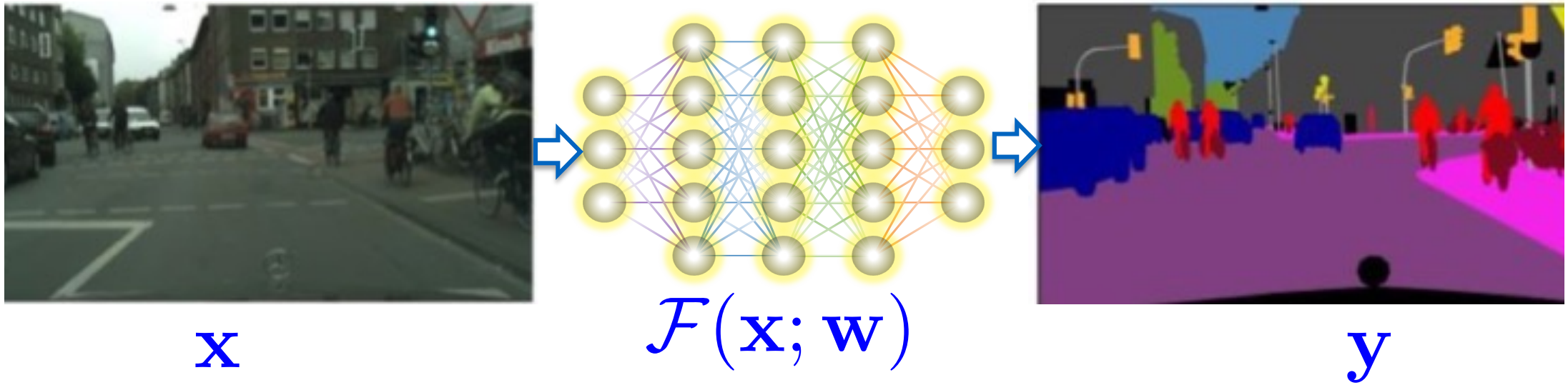
Instructor - Simon Lucey

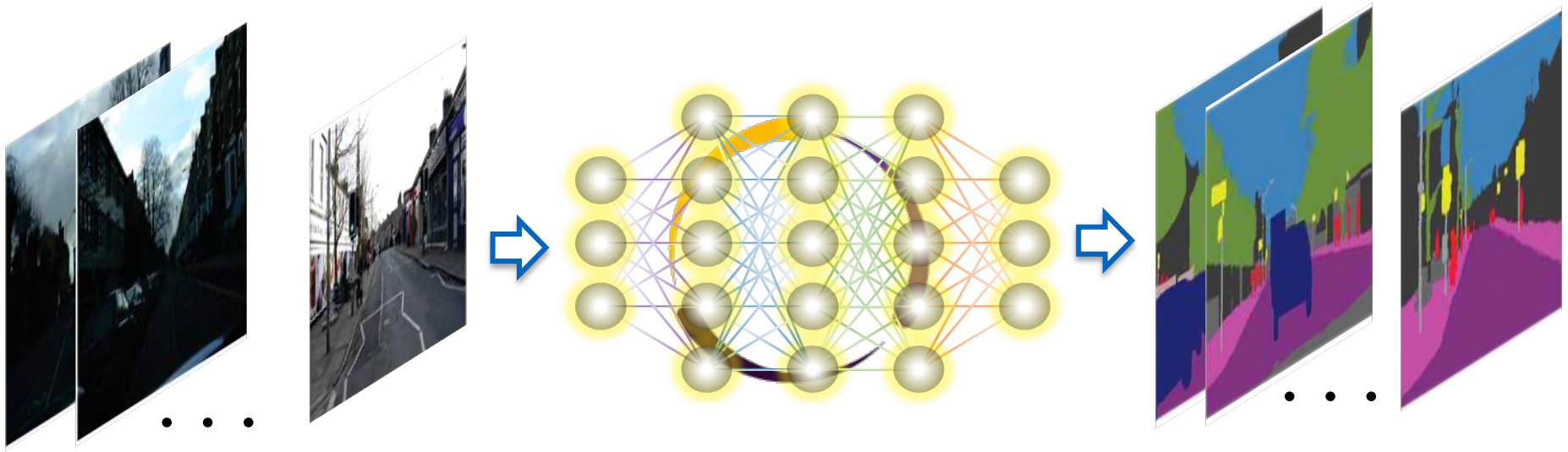
Maths of AI - 2024

THE UNIVERSITY of ADELAIDE

AUSTRALIAN INSTITUTE FOR MACHINE LEARNING


x
y


$\mathbf{x}$
$\mathcal{F}(\mathbf{x};\mathbf{w})$
$\mathbf{y}$



$[\mathbf{x}_1, \ldots, \mathbf{x}_N]$

$[\mathbf{y}_1, \ldots, \mathbf{y}_N]$

$$N \propto 10^6 - 10^9$$

Optimization and Learning

$$\min_{\mathbf{w}} \sum_{n=0}^{N-1} \mathcal{L}[y_n, \mathcal{F}(\mathbf{x}_n; \mathbf{w})]$$

What is the simplest network and loss?

$$\mathcal{L}[y, x] = ||y - x||_2^2$$ ← "least-squares loss"

$$\mathcal{F}(\mathbf{x}; \mathbf{w}) = \mathbf{x}^T \mathbf{w}$$ ← "linear network"

Today

- **Regularized Least-Squares**
- Gradient Descent
- Optimizing Deep Networks

Least-Squares Linear Model

$$\min_{\mathbf{w}} \frac{1}{2} ||\underset{(N \times 1)}{\mathbf{y}} - \underset{(M \times 1)}{\mathbf{X}\mathbf{w}}||_2^2$$

$$\mathbf{y} = [y_0, \ldots, y_{N-1}]^T$$

$$\mathbf{X} = [\mathbf{x}_0, \ldots, \mathbf{x}_{N-1}]^T$$

Least-Squares Linear Model

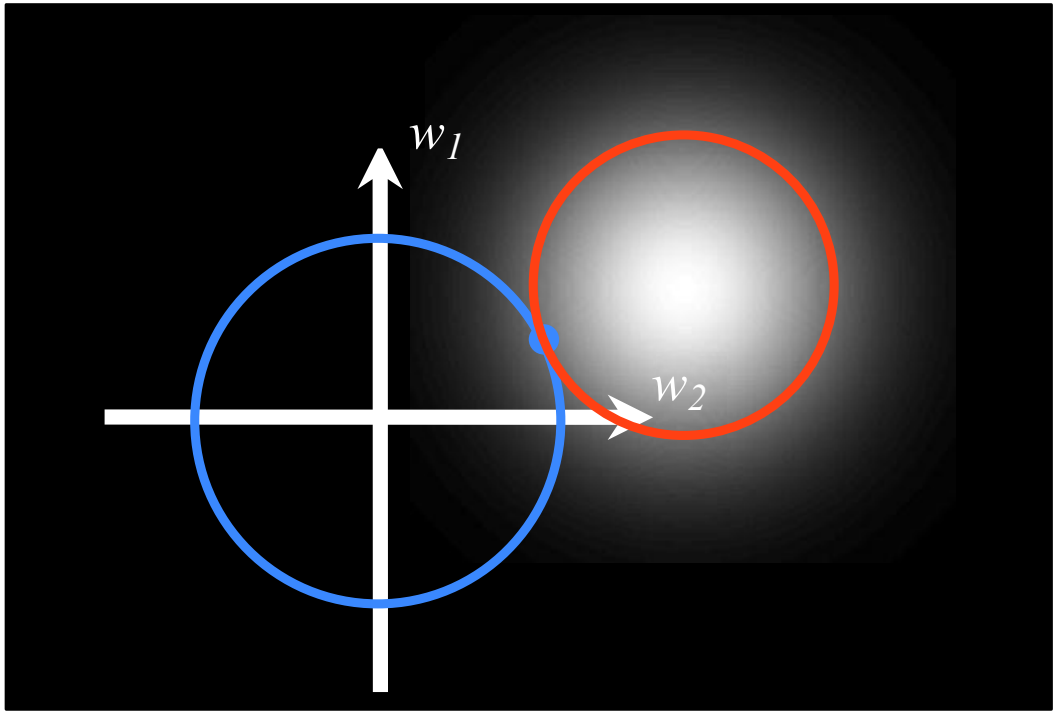
$$\min_{\mathbf{w}} \frac{1}{2}||\mathbf{y} - \mathbf{X}\mathbf{w}||_2^2 + \Omega(\mathbf{w})$$

$$\mathbf{w} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$ "closed form"

What if $(\mathbf{X}^T\mathbf{X})$ is non-singular?

Ridge Regression

$$\min_{\mathbf{w}} \frac{1}{2}||\mathbf{y} - \mathbf{X}\mathbf{w}||_2^2 \text{ s.t. } ||\mathbf{w}||_2^2 \leq \epsilon$$



Ridge Regression – Closed Form

$$\min_{\mathbf{w}} \frac{1}{2}||\mathbf{y} - \mathbf{X}\mathbf{w}||_2^2 + \frac{\lambda}{2}||\mathbf{w}||_2^2$$

$$\mathbf{w} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y}$$ "closed form solution"

$$\mathbf{X} = \mathbf{U}\mathrm{diag}(\mathbf{s})\mathbf{V}^T$$ "SVD"

$$\mathbf{U}\mathbf{U}^T = \mathbf{I} \quad \mathbf{V}\mathbf{V}^T = \mathbf{I} \qquad \mathbf{s} \geq \mathbf{0}$$

Ridge Shrinkage

$$\min_{\mathbf{w}} \frac{1}{2}||\mathbf{y} - \mathbf{X}\mathbf{w}||_2^2 + \frac{\lambda}{2}||\mathbf{w}||_2^2$$

$$\mathbf{w} = \mathbf{V}\mathrm{diag}(\hat{\mathbf{s}}^{-1})\mathbf{U}^T\mathbf{y}$$ "closed form solution"

$$\hat{s}_{\mathrm{rdg}}^{-1} = \frac{s}{s^2 + \lambda}$$ "ridge shrinkage"

Drawback: attenuates all principal components!!

PCA Shrinkage

$$\mathbf{X} = \mathbf{U}\mathrm{diag}(\mathbf{s})\mathbf{V}^T$$ "SVD"

$$\mathbf{w} = \mathbf{V}\mathrm{diag}(\hat{\mathbf{s}}^{-1})\mathbf{U}^T\mathbf{y}$$ "closed form solution"

$$\hat{s}^{-1}_{\mathrm{pca}} = m_{\mathrm{pca}}(s, \kappa) \cdot s^{-1}$$ "masked shrinkage"

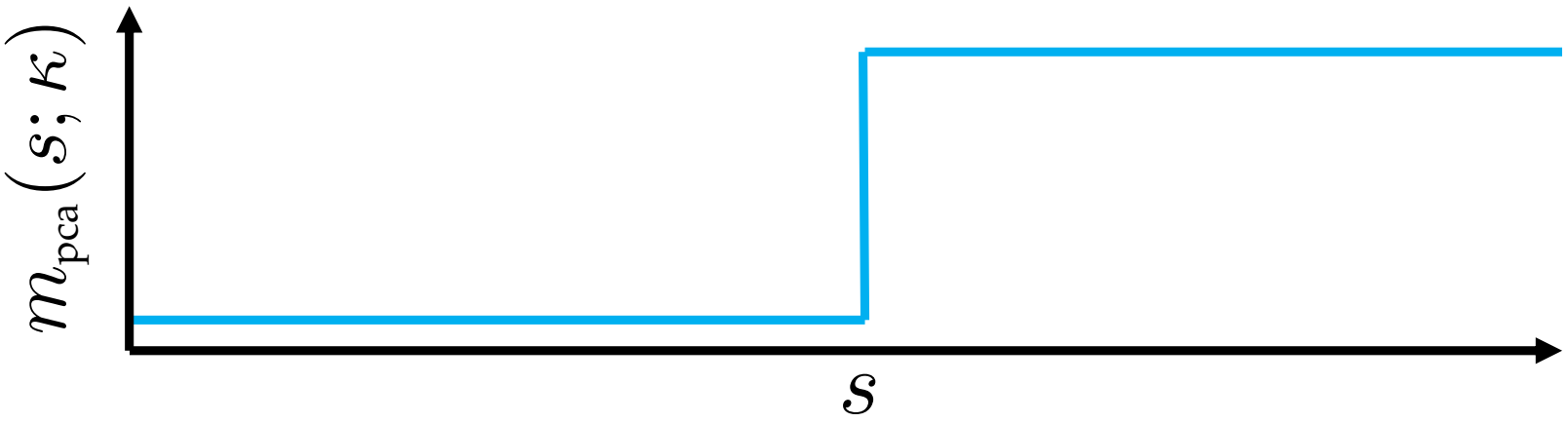
$$m_{\mathrm{pca}}(s; \kappa) = \begin{cases} 1, & \text{if } s \geq \kappa \\ 0, & \text{otherwise.} \end{cases}$$ "PCA mask"

PCA Shrinkage

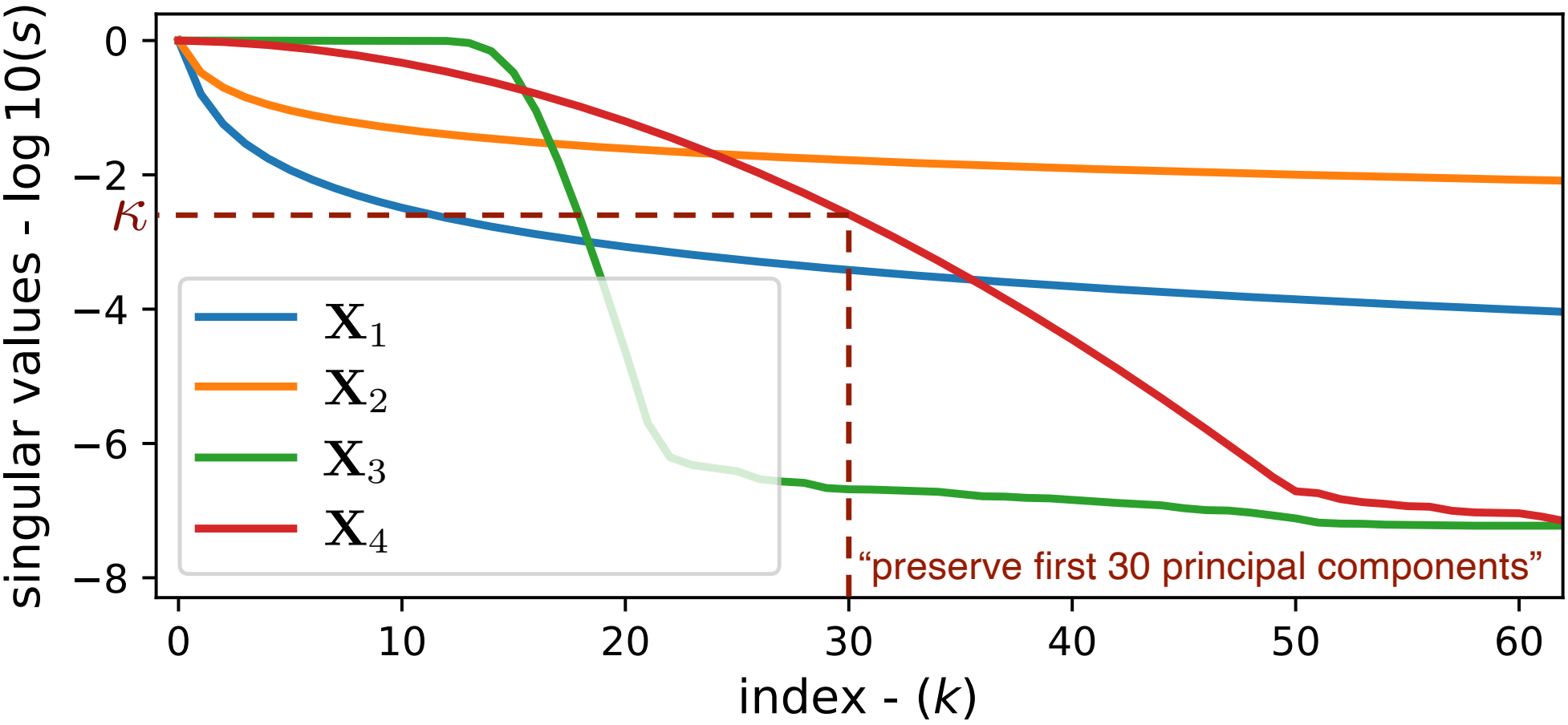
$$\mathbf{X} = \mathbf{U}\mathrm{diag}(\mathbf{s})\mathbf{V}^T$$ "SVD"

$$\mathbf{w} = \mathbf{V}\mathrm{diag}(\hat{\mathbf{s}}^{-1})\mathbf{U}^T\mathbf{y}$$ "closed form solution"

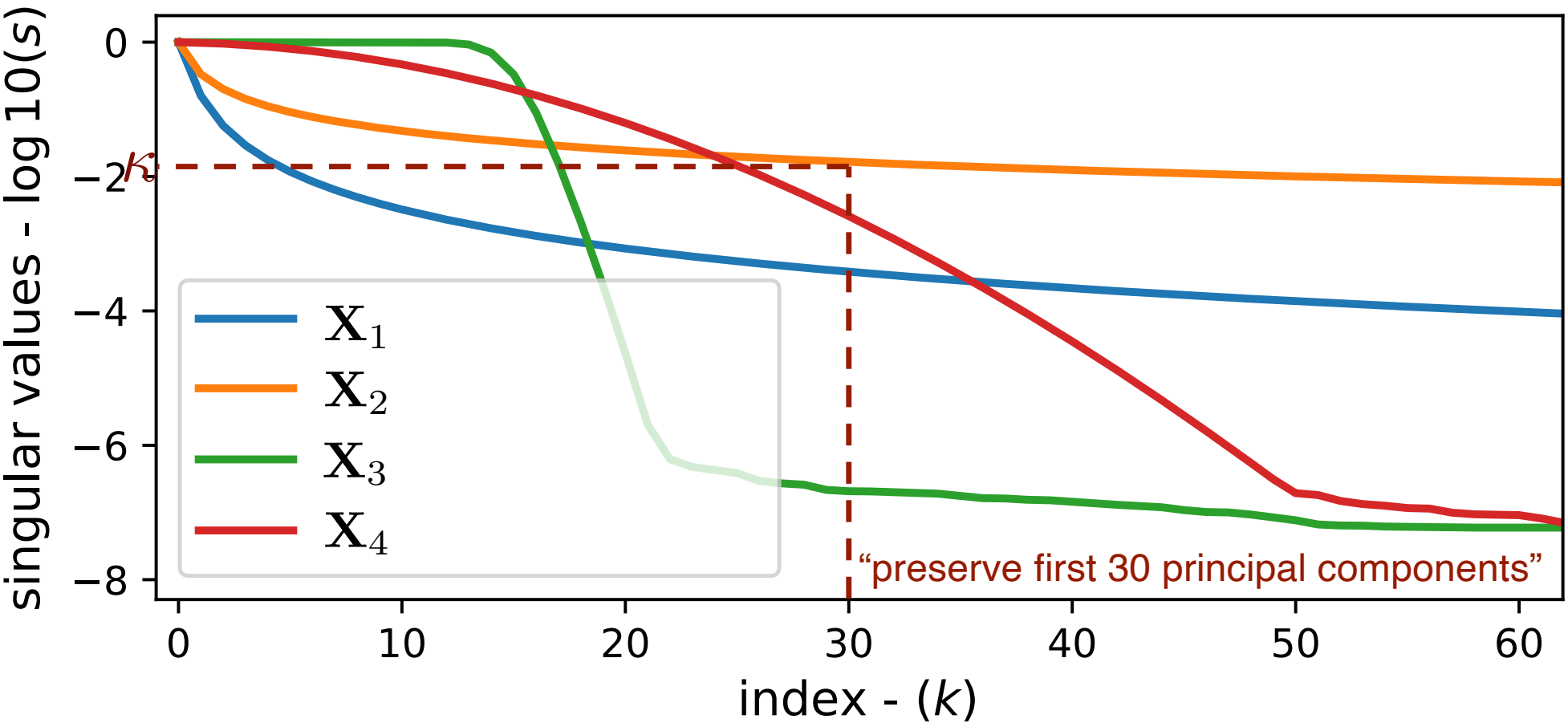
$$\hat{s}_{\mathrm{pca}}^{-1} = m_{\mathrm{pca}}(s, \kappa) \cdot s^{-1}$$ "masked shrinkage"



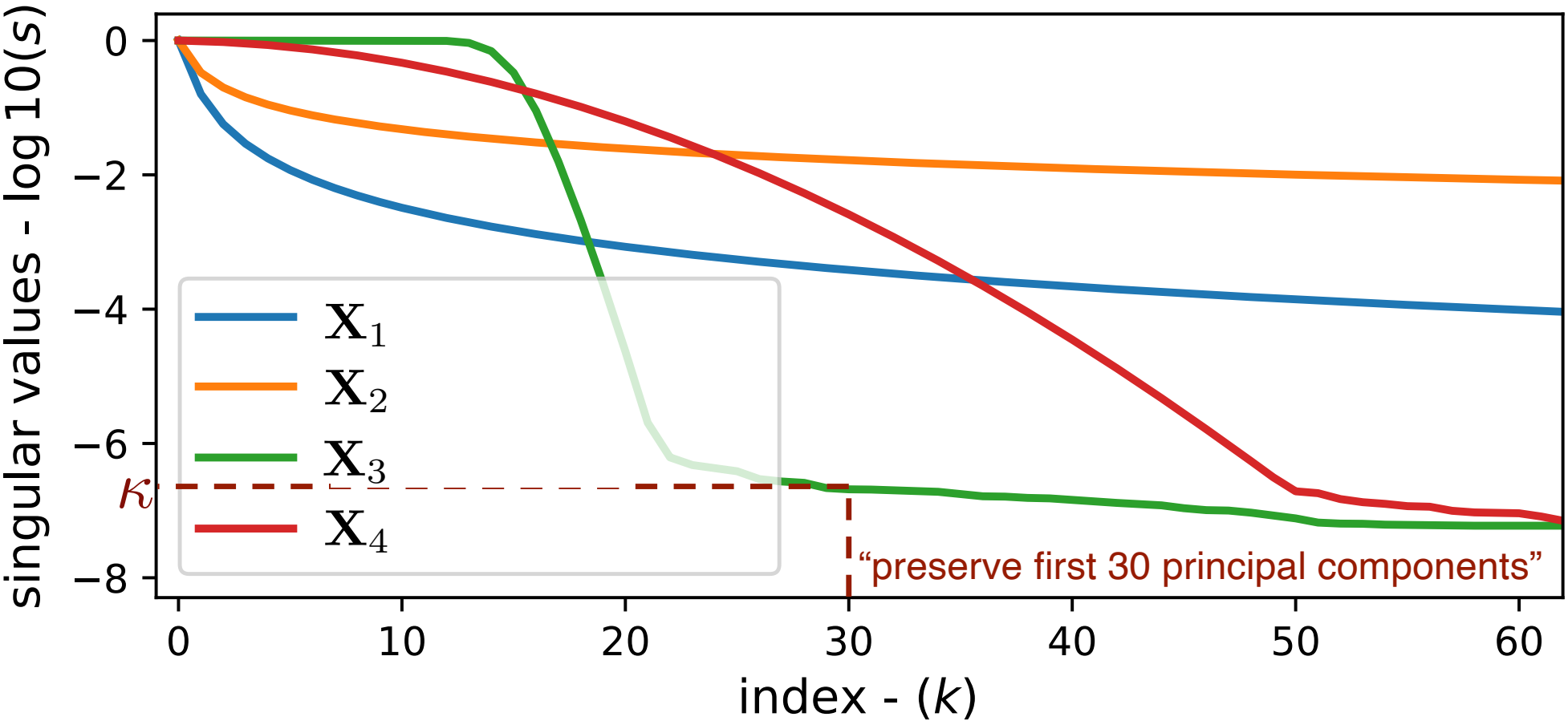
PCA Shrinkage



PCA Shrinkage



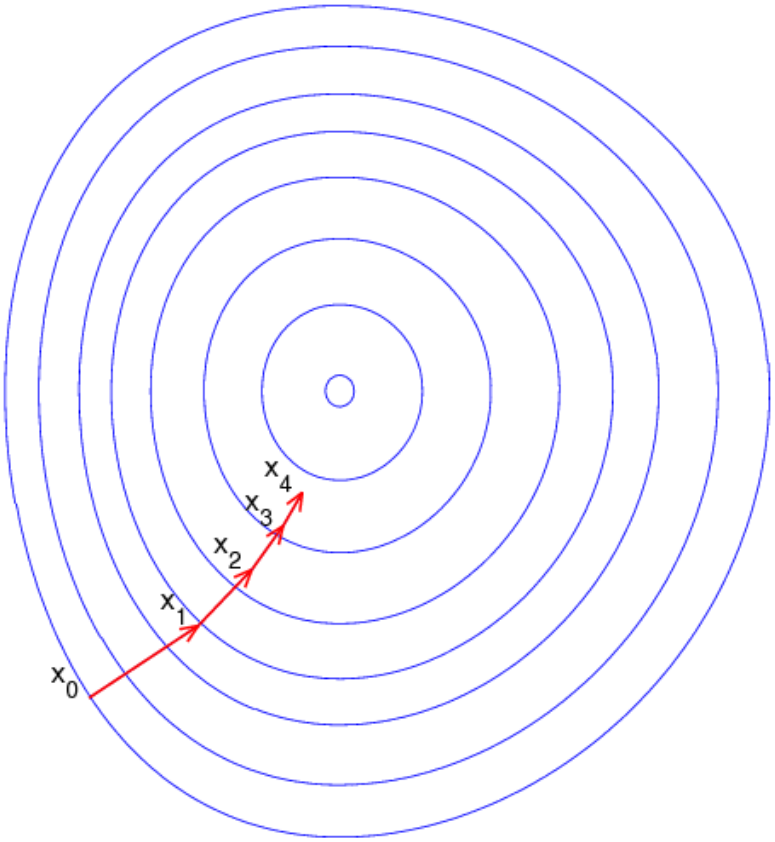
PCA Shrinkage



Today

- Regularized Least-Squares
- **Gradient Descent**
- Optimizing Deep Networks

Gradient Descent



Idea: just go downhill

Pros:

- Simple
- Always improves on current guess

Cons:

- Slow, (does not use higher order information)
- Requires objective to be differentiable.
- Heuristics on learning rate and number of iterations

Gradient Descent

$$\mathbf{w} \rightarrow \mathbf{w} - \alpha \cdot \frac{\partial \mathcal{L}(\mathbf{w})}{\partial \mathbf{w}^T}$$

"learning rate"

$$\mathcal{L}(\mathbf{w}) = \frac{1}{2}||\mathbf{y} - \mathbf{X}\mathbf{w}||_2^2$$

"loss function"

$$\frac{\partial \mathcal{L}(\mathbf{w})}{\partial \mathbf{w}^T} = -\mathbf{X}^T[\mathbf{y} - \mathbf{X}\mathbf{w}]$$

"partial derivative"

Does gradient descent implicitly regularize?

Closed form - Gradient Descent

If we start with $\mathbf{w}_0 = \mathbf{0}$

$$\mathbf{w}_1 = \alpha \mathbf{X}^T \mathbf{y}$$

$$\mathbf{w}_2 = (\mathbf{I} - \alpha \mathbf{X}^T \mathbf{X}) \alpha \mathbf{X}^T \mathbf{y} + \alpha \mathbf{X}^T \mathbf{y}$$

$$\mathbf{w}_3 = (\mathbf{I} - \alpha \mathbf{X}^T \mathbf{X})^2 \alpha \mathbf{X}^T \mathbf{y} + (\mathbf{I} - \alpha \mathbf{X}^T \mathbf{X}) \alpha \mathbf{X}^T \mathbf{y} + \alpha \mathbf{X}^T \mathbf{y}$$

at the q-th iteration generalizes to,

$$\mathbf{w}_q = \sum_{k=0}^{q-1} (\mathbf{I} - \alpha \mathbf{X}^T \mathbf{X})^{k-1} \alpha \mathbf{X}^T \mathbf{y}$$

Closed form - Gradient Descent

Using the Neumann series identity,

$$\sum_{k=0}^{q-1} z^k = (1 - z^q)(1 - z)^{-1}$$

Closed form solution,

$$\mathbf{w}_q = [\mathbf{I} - (\mathbf{I} - \alpha \mathbf{X}^T \mathbf{X})^q](\alpha \mathbf{X}^T \mathbf{X})^{-1} \alpha \mathbf{X}^T \mathbf{y}$$

https://www.cs.cornell.edu/courses/cs6241/2021sp/lec/2021-02-18.pdf

Gradient Descent Shrinkage

$$\mathbf{X} = \mathbf{U}\mathrm{diag}(\mathbf{s})\mathbf{V}^T$$ "SVD"

$$\mathbf{w} = \mathbf{V}\mathrm{diag}(\hat{\mathbf{s}}^{-1})\mathbf{U}^T\mathbf{y}$$ "closed form solution"

$$\hat{s}_{\mathrm{gd}}^{-1} = m_{\mathrm{gd}}(s;\alpha,q)\cdot s^{-1}$$ "masked shrinkage"

$$m_{\mathrm{gd}}(s;\alpha,q) = [1-(1-\alpha\cdot s^2)^q]$$ "GD mask"

Choosing Learning Rate and Iterations

Inspecting the masking function,

$$m_{\mathrm{gd}}(s;\alpha,q) = [1-(1-\alpha\cdot s^2)^q]$$

we can see that,

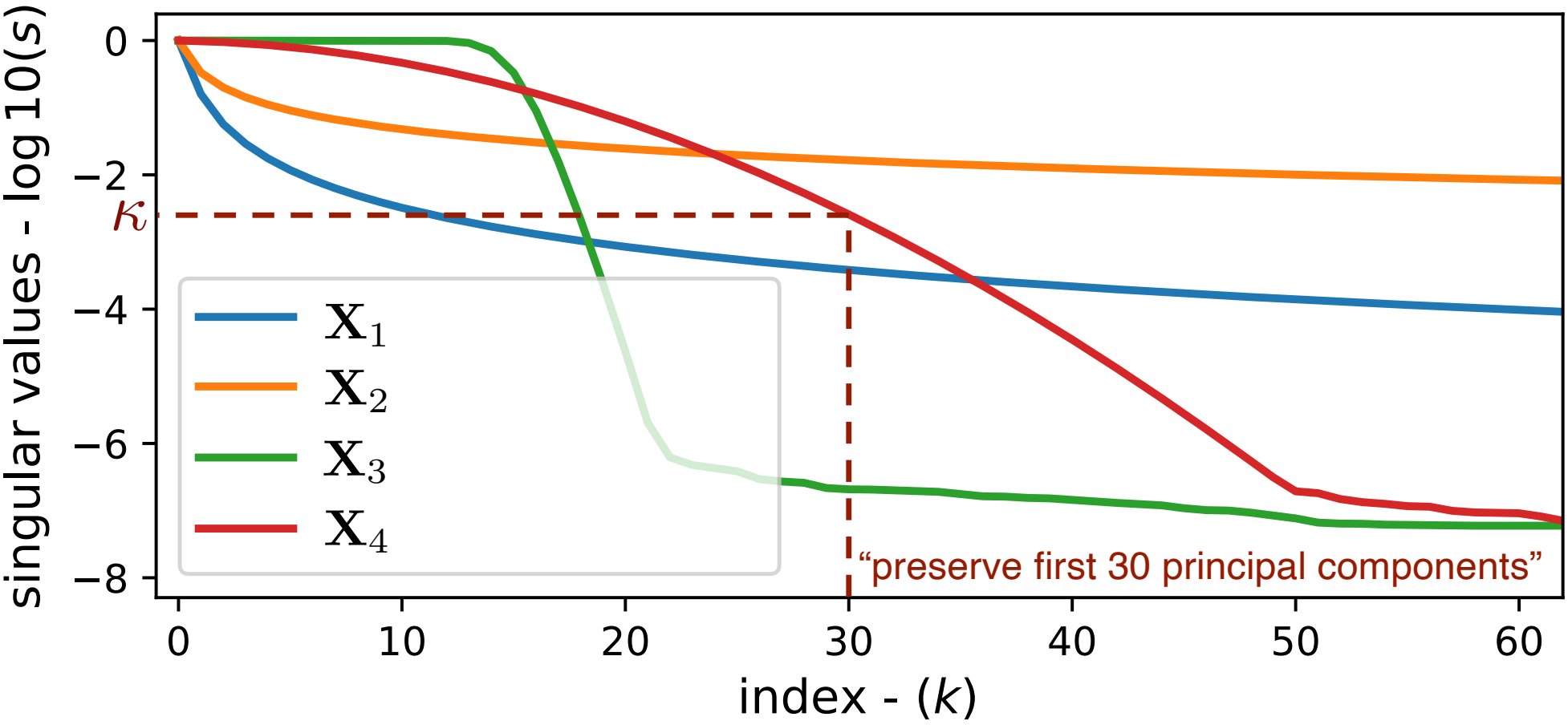
$$\alpha \leq 2\cdot s_{\max}^{-2}$$

for stable efficient performance,

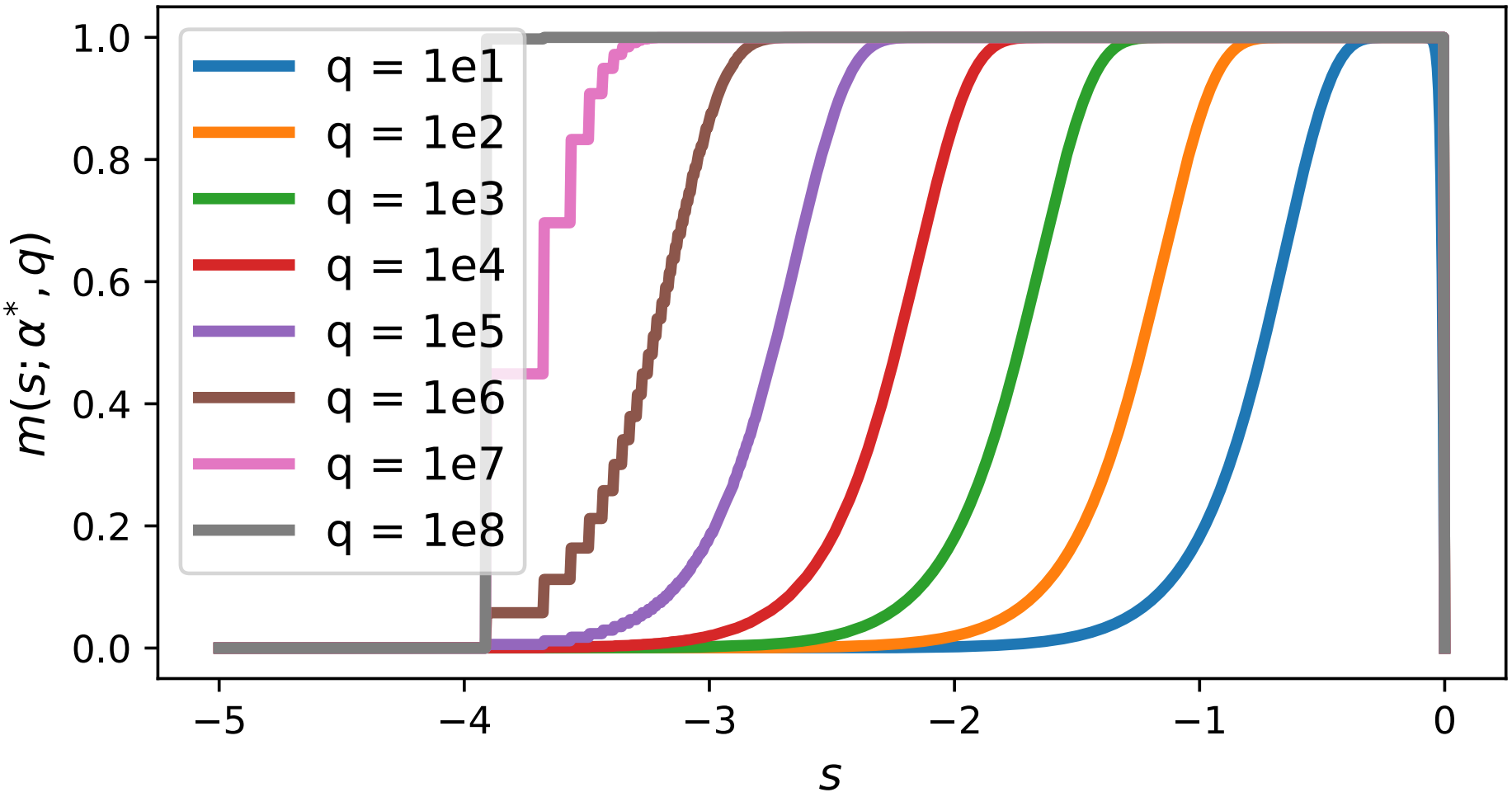
$$\alpha^* = s_{\max}^{-2} \quad \text{and} \quad q^* \approx \frac{s_{\max}^2}{\kappa^2}$$

"remember from PCA shrinkage"

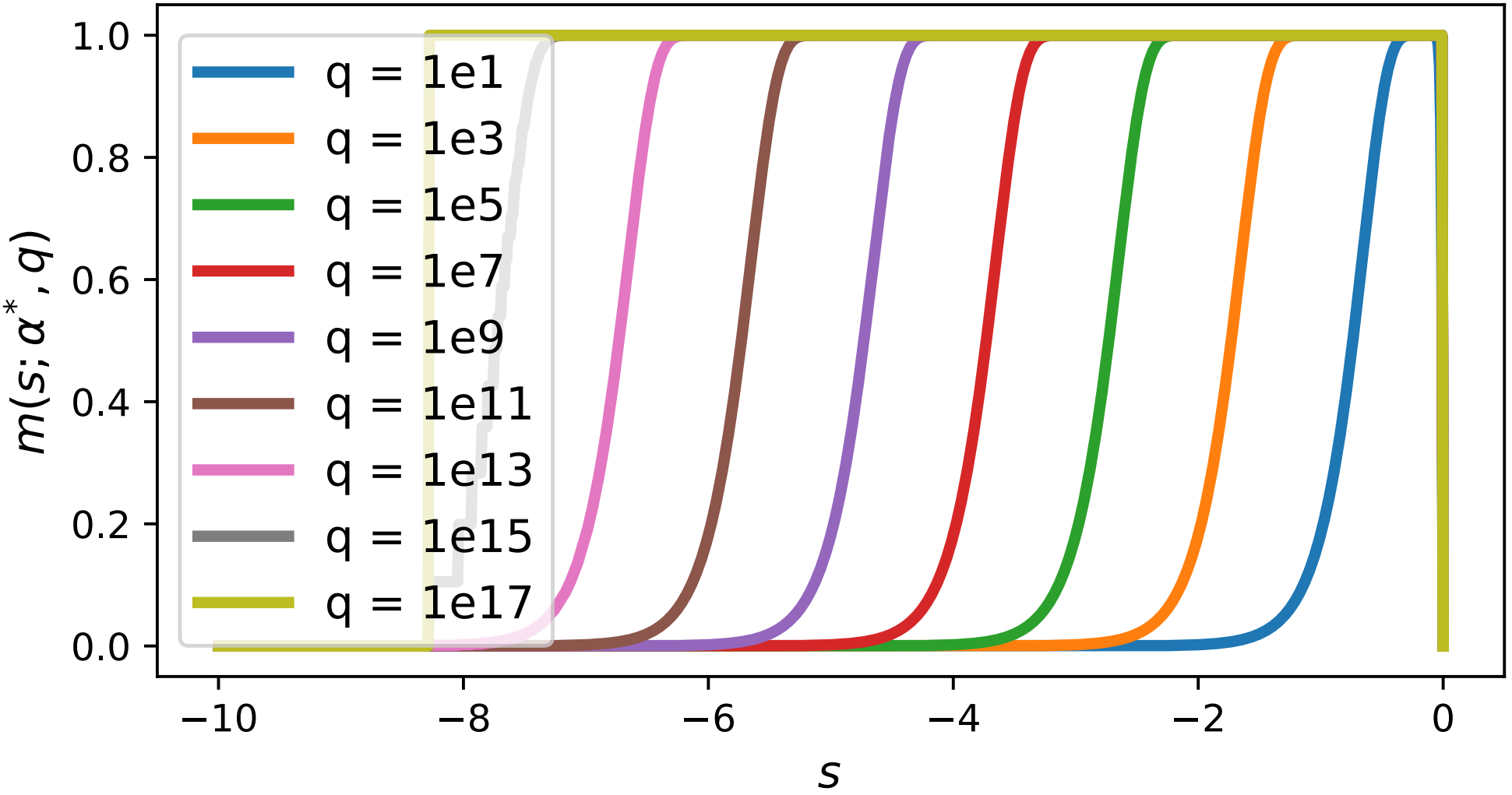
Remember - PCA Shrinkage



Gradient Descent – Shrinkage (Float 32 bit)



Gradient Descent – Shrinkage (Float 64 bit)



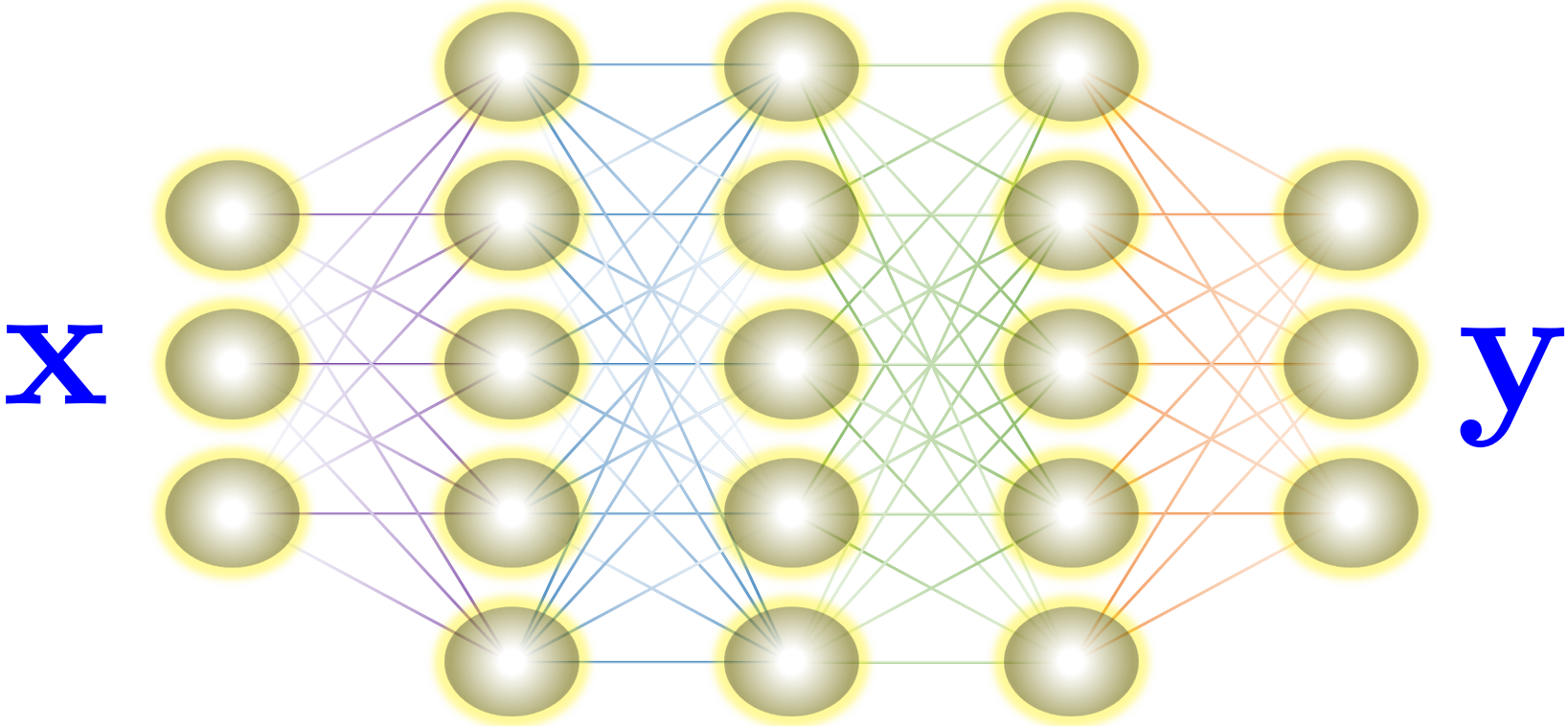
Gradient Descent Shrinkage - Review

- Gradient descent on linear models can be reinterpreted as a type of implicit shrinkage regularizer.
- Unlike explicit PCA shrinkage, it does not require any costly SVD step making it ideal for large scale problems.
- Number of iterations and learning rate approximately dictate the number of principal components to preserve,
- Floating point precision provides an upper bound on the number of principal components that can be preserved.

Today

- Regularized Least-Squares
- Gradient Descent
- **Optimizing Deep Networks**

Optimizing Deep Networks



$$\mathcal{F}(\mathbf{x}; \mathbf{w})$$

Linearizing Deep Networks

$$\min_{\mathbf{w}} \frac{1}{2} ||\mathbf{y} - \mathcal{F}(\mathbf{X}; \mathbf{w})||_F^2$$

$$\mathcal{F}(\mathbf{X}; \mathbf{w}) \approx \mathcal{F}(\mathbf{X}; \mathbf{w}_0) + \frac{\partial \mathcal{F}(\mathbf{X}; \mathbf{w}_0)}{\partial \mathbf{w}^T} [\mathbf{w} - \mathbf{w}_0]$$

"network Jacobian"

Could we apply the same trick?

Reminder - Gradient Descent Shrinkage

$$\nabla \mathcal{F}(\mathbf{X}; \mathbf{w}) = \mathbf{U}\mathrm{diag}(\mathbf{s})\mathbf{V}^T$$ "SVD"

$$\mathbf{w} = \mathbf{V}\mathrm{diag}(\hat{\mathbf{s}}^{-1})\mathbf{U}^T\mathbf{y}$$ "closed form solution"

$$\hat{s}_{\mathrm{gd}}^{-1} = m_{\mathrm{gd}}(s; \alpha, q) \cdot s^{-1}$$ "masked shrinkage"

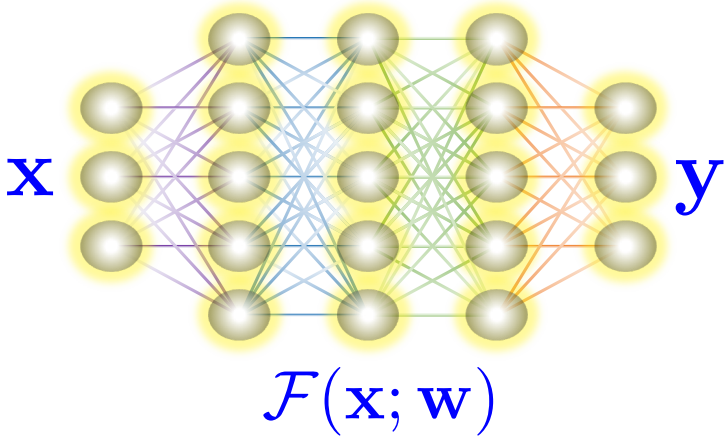
$$m_{\mathrm{gd}}(s; \alpha, q) = [1 - (1 - \alpha \cdot s^2)^q]$$ "GD mask"

Linear Network - Fixed Jacobian

$$\frac{\partial \mathcal{F}(\mathbf{X};\mathbf{w})}{\partial \mathbf{w}^T} = \text{constant}$$

Deep Network - Dynamic Jacobian

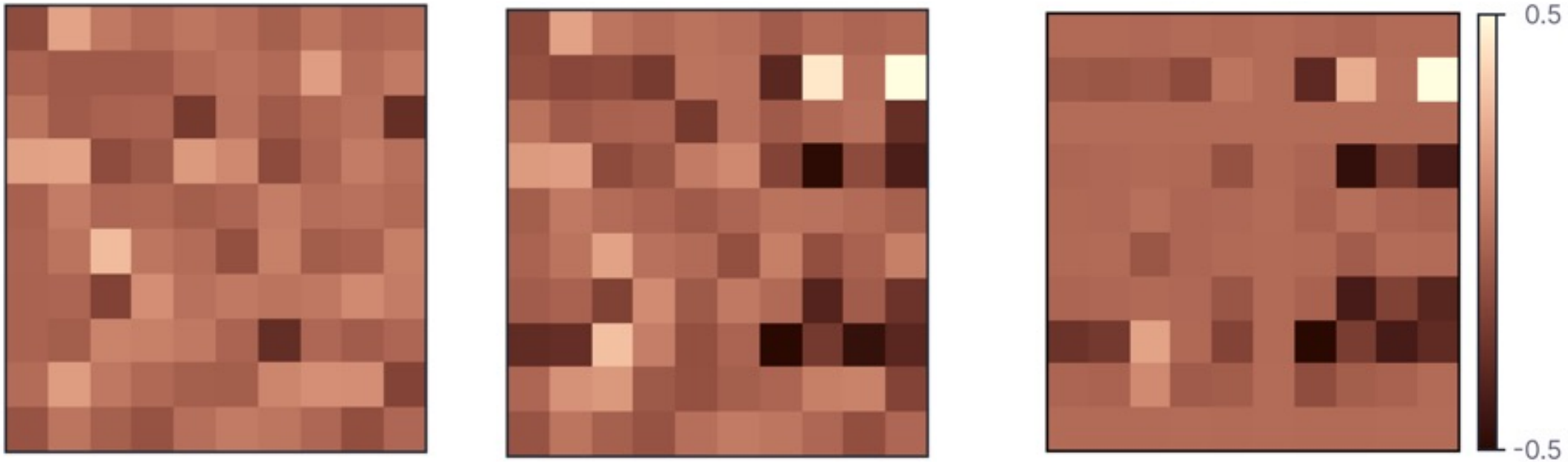
$$\frac{\partial \mathcal{F}(\mathbf{X};\mathbf{w})}{\partial \mathbf{w}^T} \neq \text{constant}$$



Solution – Overparametrization?

e.g. 2 layer MLP, varying width

width = 10



$\mathbf{w}_0$ $\mathbf{w}_\infty$ $\mathbf{w}_0 - \mathbf{w}_\infty$

https://www.borealisai.com/research-blogs/the-neural-tangent-kernel/

Solution – Overparameterization?

e.g. 2 layer MLP, varying width

width = 1000

$\mathbf{w}_0 - \mathbf{w}_\infty$

arXiv:2407.14679v1 [cs.CL] 19 Jul 2024

Compact Language Models via Pruning and Knowledge Distillation

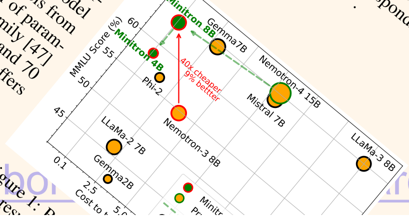
Saurav Muralidharan* Sharath Turuvekere Sreenivas* Raviraj Joshi
Marcin Chochowski Mostofa Patwary Mohammad Shoeybi Bryan Catanzaro
Jan Kautz Pavlo Molchanov
NVIDIA
{sauravm,sharatht,ravirajj,mchochowski,mpatwary,mshoeybi,
bcatanzaro,jkautz,pmolchanov}@nvidia.com

Abstract

Large language models (LLMs) targeting different deployment scales and sizes are currently produced by training each variant from scratch; this is extremely compute-intensive. In this paper, we investigate if pruning an existing LLM and then re-training it with a fraction (<3%) of the original training data can be a suitable alternative to repeated, full retraining. To this end, we develop a set of practical and effective **compression best practices** for LLMs that combine depth, width, attention and MLP pruning with knowledge distillation-based retraining; we arrive at these best practices through a detailed empirical exploration of pruning strategies for each axis, methods to combine axes, distillation strategies, and search techniques for arriving at optimal compressed architectures. We use this guide to compress the Nemotron-4 family of LLMs by a factor of 2-4×, and compare their performance to similarly-sized models on a variety of language modeling tasks. Deriving 8B and 4B models from an already pretrained 15B model using our approach requires up to 40× fewer training tokens per model compared to training from scratch; this results in compute cost savings of 1.8× for training the full model family (15B, 8B, and 4B). MINITRON models exhibit up to a 16% improvement in MMLU scores compared to training from scratch, perform comparably to other community models such as Mistral 7B, Gemma 7B and Llama-3 8B, and outperform state-of-the-art compression techniques from the literature. We have open-sourced MINITRON model weights on Huggingface [2], with corresponding supplementary material including example code available on GitHub [3].

1 Introduction

Large language models (LLMs) now dominate real-world natural language processing and have demonstrated excellent proficiency in understanding difficult contexts [7, 40, 50, 47, 46]. To aid users targeting different deployment sizes and scales, model providers often train an entire family of models from scratch, each with a different size (number of parameters). For instance, the LLaMa-2 model family [47] includes three different variants with 7, 13, and 70 billion parameters, while the Pythia family [6] offers



*Equal contribution.
[2]https://huggingface.co/nvidia
[3]https://github.com/NVlabs/Minitron

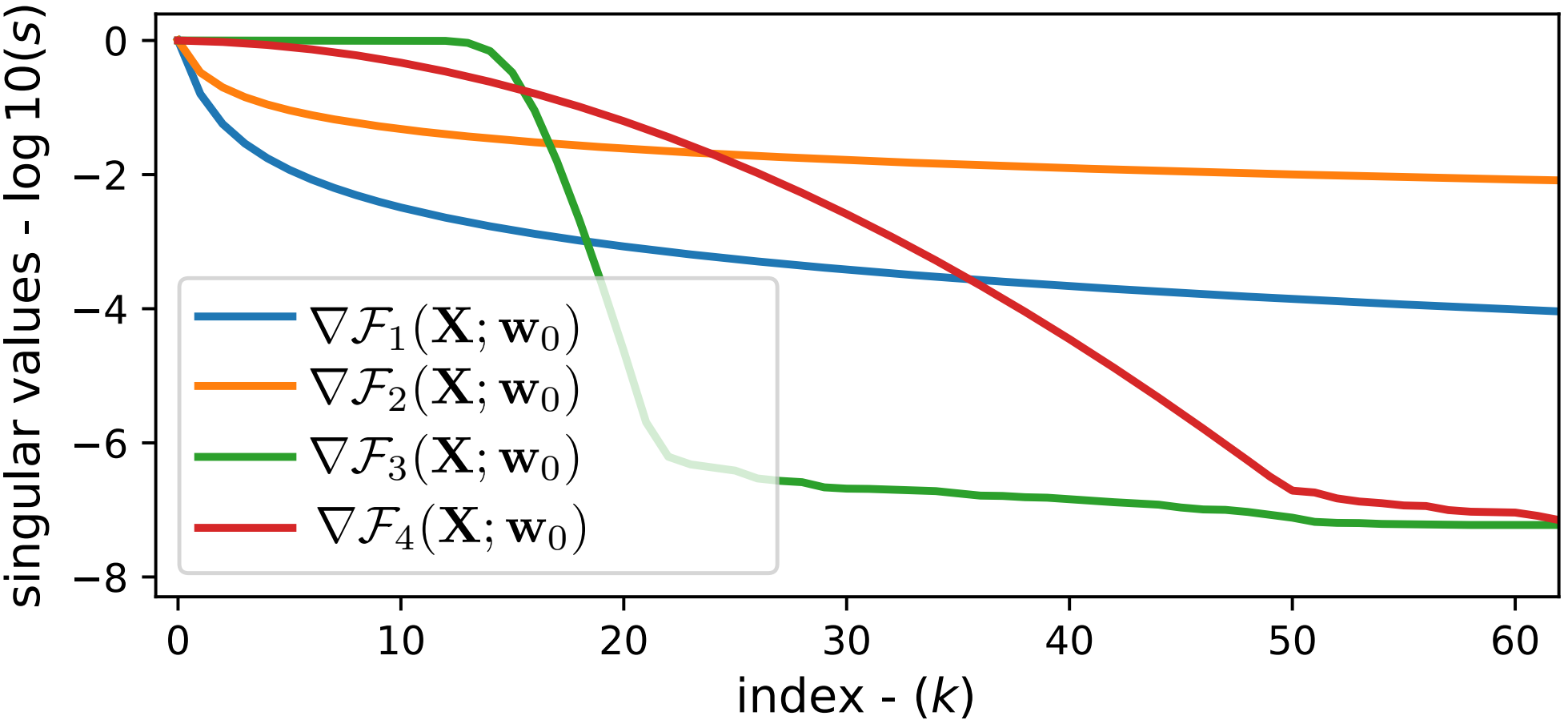
https://www.bo[illegible]rch-blogs/the-neural-tangent-kernel/

Neural Tangent Kernel

- It has been argued by Jacot et al. that a deep network's Jacobian will become fixed as the width of the network tends towards infinity.
- Allows for the closed form analysis, and better motivates numerous modern interventions including:-
  - Skip connections
  - Batch-norm
  - Layer-norm
  - Adam optimization.

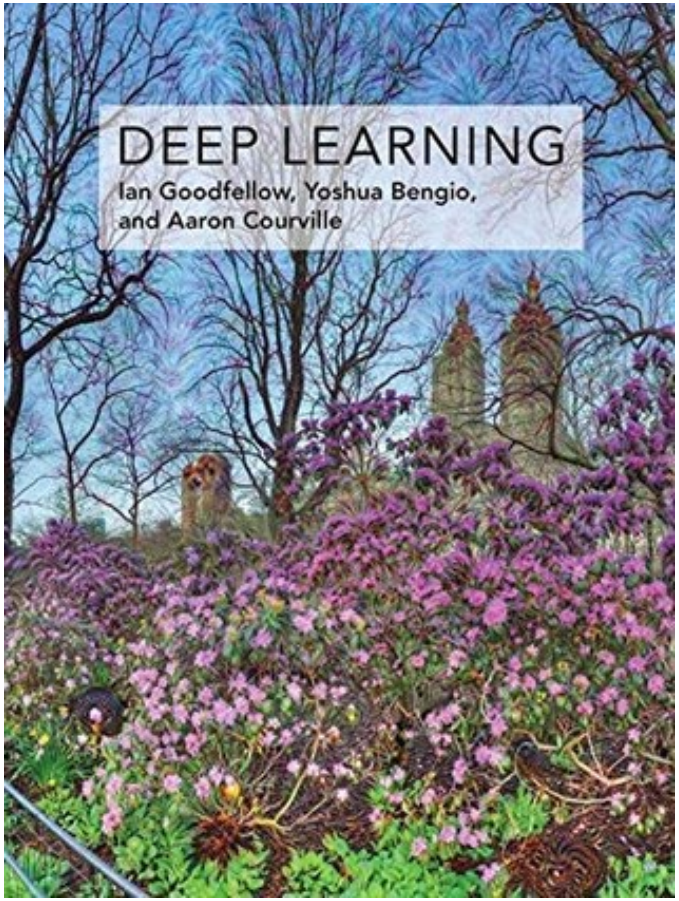
https://proceedings.neurips.cc/paper_files/paper/2018/file/5a4be1fa34e62bb8a6ec6b91d2462f5a-Paper.pdf

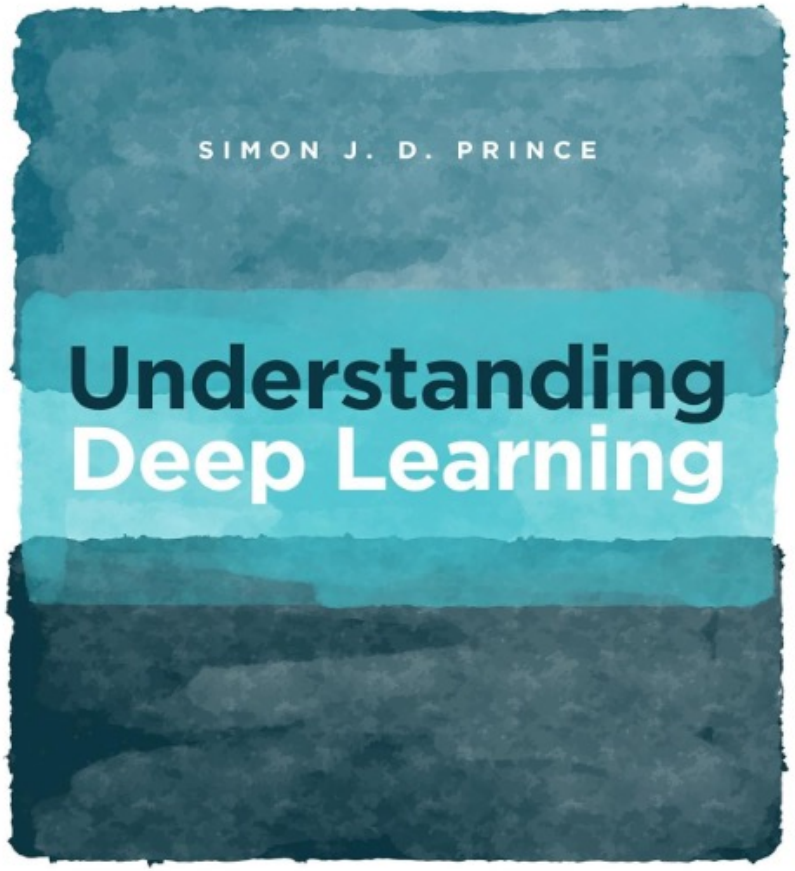
Problem - Ill-Conditioned Jacobian



https://www.borealisai.com/research-blogs/the-neural-tangent-kernel/

More to read…


DEEP LEARNING
Ian Goodfellow, Yoshua Bengio,
and Aaron Courville


SIMON J. D. PRINCE
Understanding
Deep Learning