

CS-UY 4563: Lecture 21

Auto-encoders, Principal Component Analysis

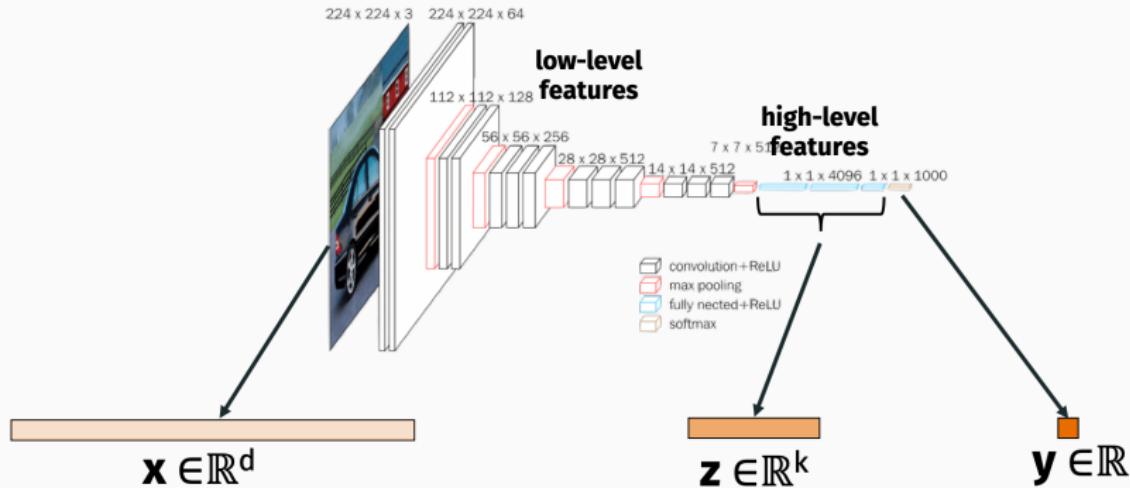
NYU Tandon School of Engineering, Prof. Christopher Musco

COURSE LOGISTICS

- Next weeks should be focused on project work! Final report due **5/11**.
- I am still working through proposals. If you feel blocked/need my input to move forward on project, please email or come to office hours.
- Each group will give a **5 minute presentation** in class on **5/6** or **5/11**. Link for signing up for a slot is on the course webpage.
- Details on expectations for presentation will be released soon.

TRANSFER LEARNING

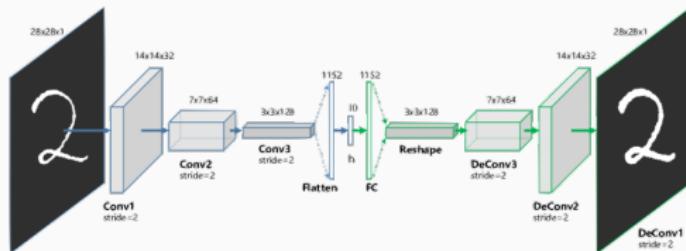
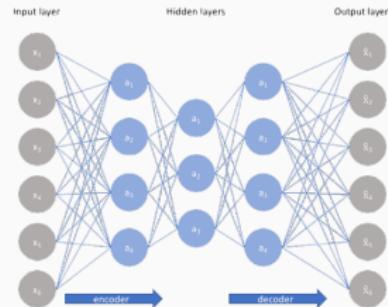
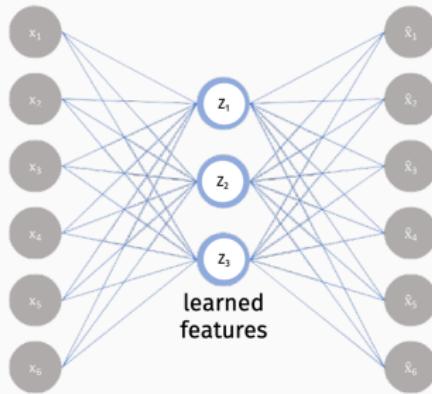
Machine learning algorithms like neural networks learn high level features.



These features are useful for other tasks that the network was not trained specifically to solve.

AUTOENCODER

Idea behind autoencoders: If you have limited labeled data, make the inputs the targets. Learn to reconstruct input data and extract high-level features along the way.

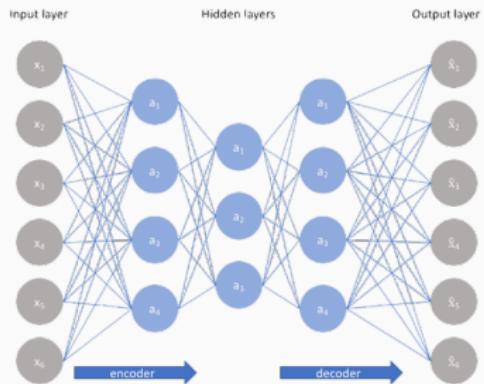


AUTOENCODER

Encoder: $e : \mathbb{R}^d \rightarrow \mathbb{R}^k$

Decoder: $d : \mathbb{R}^k \rightarrow \mathbb{R}^d$

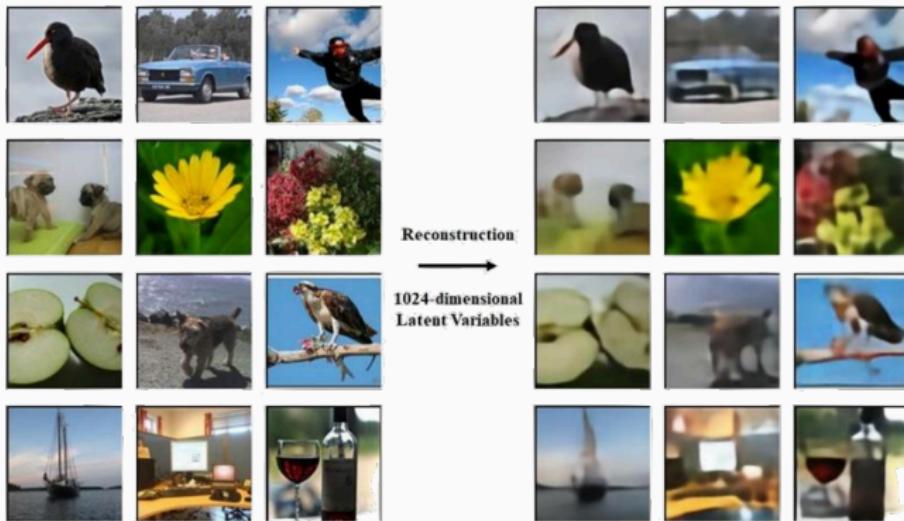
$$f(\vec{x}) = e(d(x))$$



The number of learned features k is typically $\ll d$.

AUTOENCODER RECONSTRUCTION

Example image reconstructions from autoencoder:



<https://www.biorxiv.org/content/10.1101/214247v1.full.pdf>

Input parameters: $d = 49152$.

Bottleneck “latent” parameters: $k = 1024$.

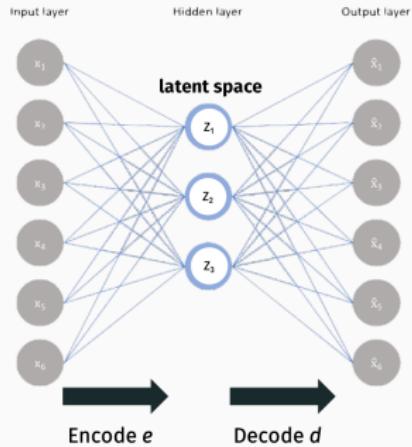
AUTOENCODERS FOR FEATURE EXTRACTION

Autoencoders also have many other applications besides feature extraction.

- Learned image compression.
- Denoising and in-painting.
- Image synthesis.

AUTOENCODERS FOR DATA COMPRESSION

Due to their bottleneck design, autoencoders perform **dimensionality reduction** and thus data compression.



Given input \vec{x} , we can completely recover $f(\vec{x})$ from $\vec{z} = e(\vec{x})$. \vec{z} typically has many fewer dimensions than \vec{x} and for a typical $f(\vec{x})$ will closely approximate \vec{x} .

AUTOENCODERS FOR IMAGE COMPRESSION

The best lossy compression algorithms are tailor made for specific types of data:

- JPEG 2000 for images
- MP3 for digital audio.
- MPEG-4 for video.

All of these algorithms take advantage of specific structure in these data sets. E.g. JPEG assumes images are locally “smooth”.

AUTOENCODERS FOR IMAGE COMPRESSION

With enough input data, autoencoders can be trained to find this structure on their own.

"End-to-end optimized image compression", Ballé, Laparra, Simoncelli

Need to be careful about how you choose loss function, design the network, etc. but can lead to much better image compression than "hand-designed" algorithms like JPEG.

AUTOENCODERS FOR DATA RESTORATION

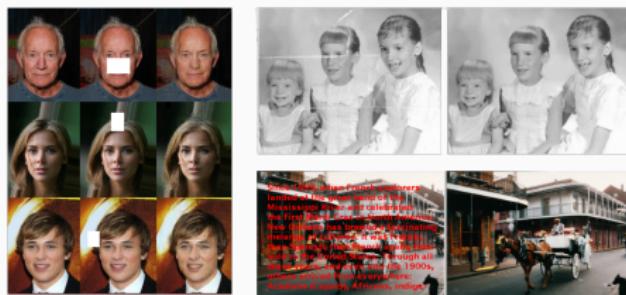
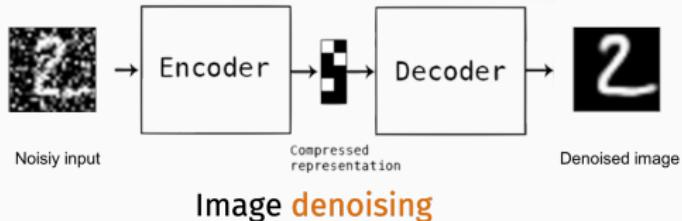


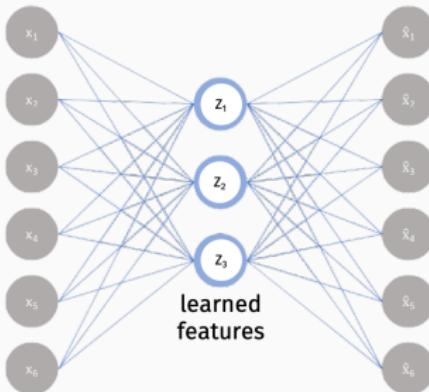
Image **inpainting**

Train autoencoder on uncorrupted data. Pass corrupted data \vec{x} through autoencoder and return $f(\vec{x})$ as repaired result.¹

¹Works much better if trained on corrupted data. More on this later.

AUTOENCODERS LEARN COMPRESSED REPRESENTATIONS

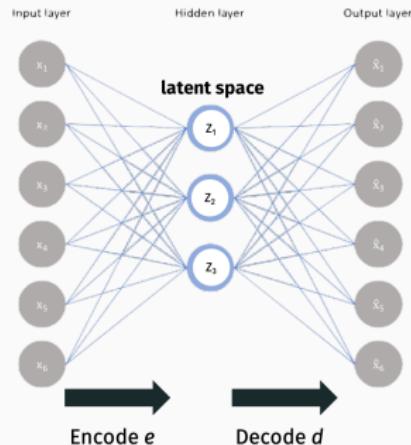
Why does this work?



Definitions:

- Let \mathcal{A} be our original data space. E.g. $\mathcal{A} = \mathbb{R}^d$ for some dimension d .
- Let \mathcal{S} be the set of all data examples which could be the output of our autoencoder f . We have that $\mathcal{S} \subset \mathcal{A}$. Formally,
$$\mathcal{S} = \{\vec{y} \in \mathbb{R}^d : \vec{y} = f(\vec{x}) \text{ for some } \vec{x} \in \mathbb{R}^d\}.$$

AUTOENCODERS LEARN COMPRESSED REPRESENTATIONS



Consider $128 \times 128 \times 3$ images with pixels values in $0, 1, \dots, 255$. How many unique images are there in \mathcal{A} ?

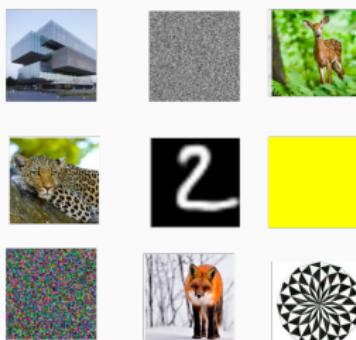
Suppose \vec{z} holds k values between in $0, .1, .2, \dots, 1$. Roughly how many unique images are there in \mathcal{S} ?

AUTOENCODERS LEARN COMPRESSED REPRESENTATIONS

So, any autoencoder can only represent a tiny fraction of all possible images. This is a good thing.



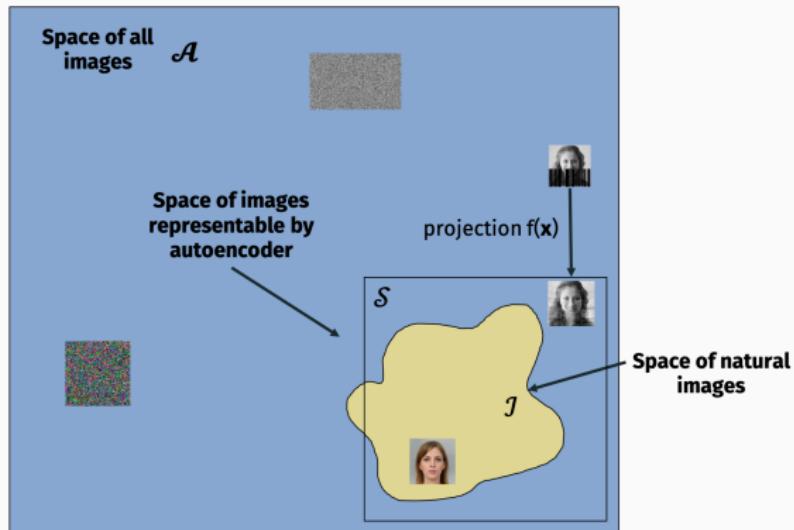
Training data



Which images are likely in \mathcal{S} ?

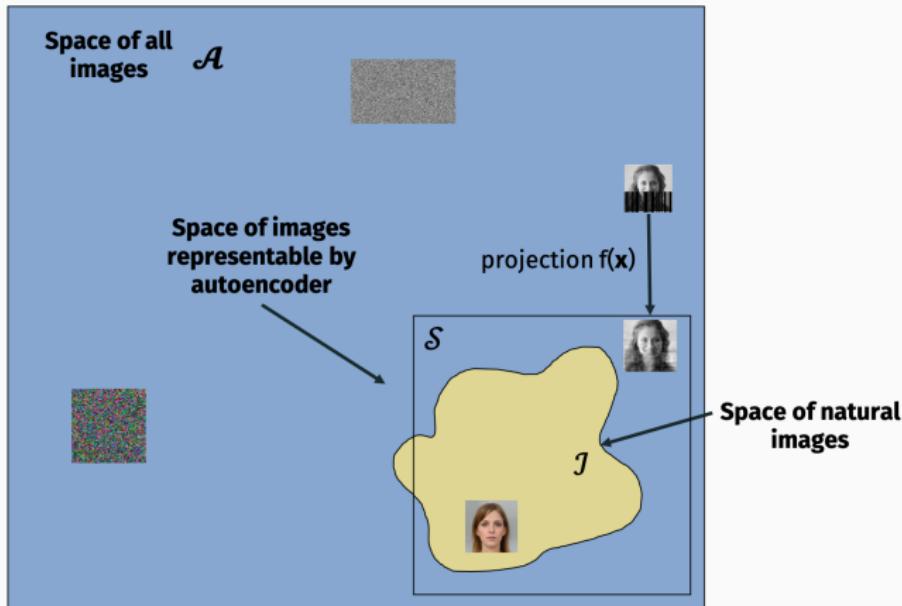
AUTOENCODERS LEARN COMPRESSED REPRESENTATIONS

$$\mathcal{S} = \{\vec{y} \in \mathbb{R}^d : \vec{y} = f(\vec{x}) \text{ for some } \vec{x} \in \mathbb{R}^d\}$$



For a good (accurate, small bottleneck) autoencoder, \mathcal{S} will closely approximate \mathcal{I} . Both will be much smaller than \mathcal{A} .

AUTOENCODERS LEARN COMPRESSED REPRESENTATIONS

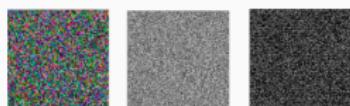


$f(\vec{x})$ projects an image \vec{x} closer to the space of natural images.

AUTOENCODERS FOR DATA GENERATION

Suppose we want to generate a random natural image. How might we do that?

- **Option 1:** Draw each pixel in \vec{x} value uniformly at random.
Draws a random image from \mathcal{A} .



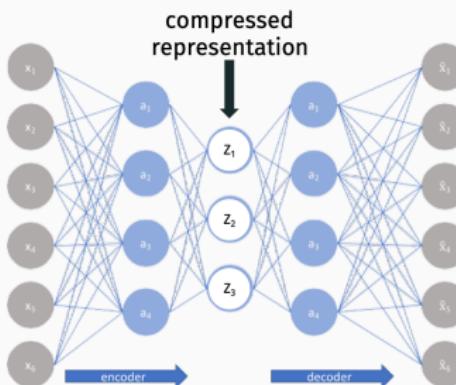
- **Option 2:** Draws \vec{x} randomly image from \mathcal{S} .



How do we randomly select an image from \mathcal{S} ?

AUTOENCODERS FOR DATA GENERATION

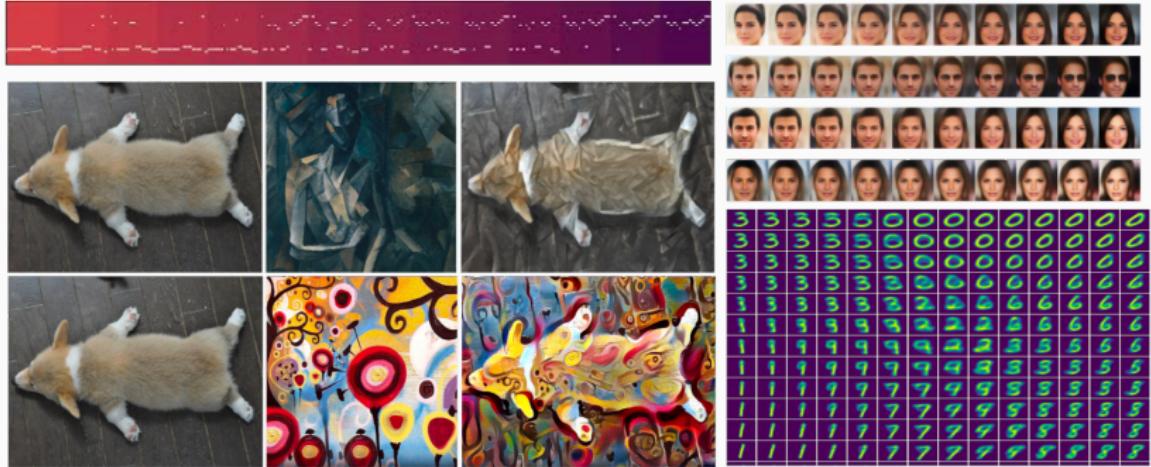
How do we randomly select an image \vec{x} from \mathcal{S} ?



Randomly select code \vec{z} , then set $\vec{x} = e(\vec{z})$.²

²Lots of details to think about here. In reality, people use “variational autoencoders” (VAEs), which are a natural modification of AEs.

AUTOENCODERS FOR DATA GENERATION



Generative models are a growing area of machine learning, driven by a lot of interesting new ideas. Generative Adversarial Networks in particular are now a major competitor with variational autoencoders.

PRINCIPAL COMPONENT ANALYSIS

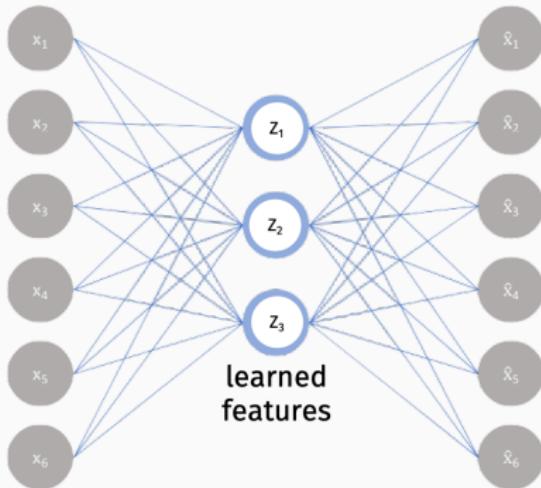
Remainder of lecture: Deeper dive into understanding a simple, but powerful autoencoder architecture. Specifically we will learn about **principal component analysis (PCA)** as a type of autoencoder.

PCA is the “linear regression” of unsupervised learning: often the go-to baseline method for feature extraction and dimensionality reduction.

Very important outside machine learning as well.

PRINCIPAL COMPONENT ANALYSIS

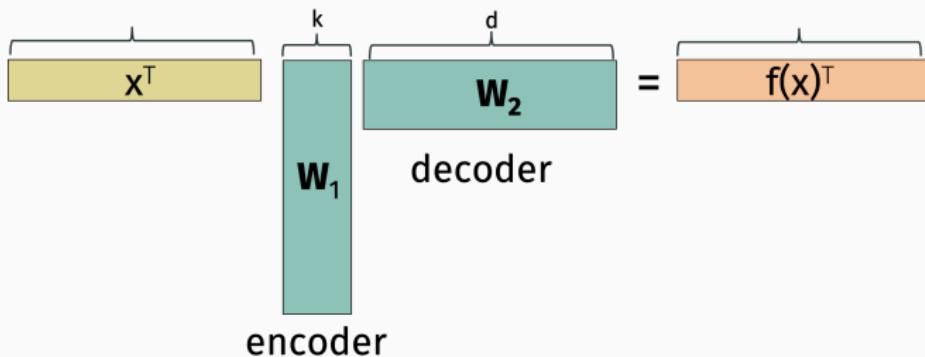
Consider the simplest possible autoencoder:



- One hidden layer. No non-linearity. No biases.
- Latent space of dimension k .
- Weight matrices are $W_1 \in \mathbb{R}^{d \times k}$ and $W_2 \in \mathbb{R}^{k \times d}$.

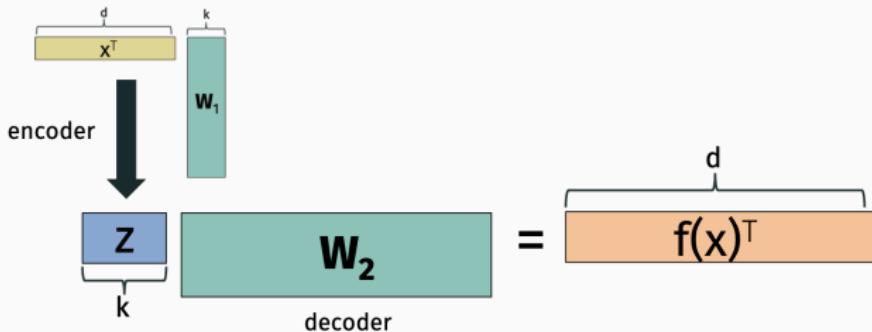
PRINCIPAL COMPONENT ANALYSIS

Given input $\vec{x} \in \mathbb{R}^d$, what is $f(\vec{x})$ expressed in linear algebraic terms?



$$f(\vec{x})^T = \vec{x}^T W_1 W_2$$

PRINCIPAL COMPONENT ANALYSIS

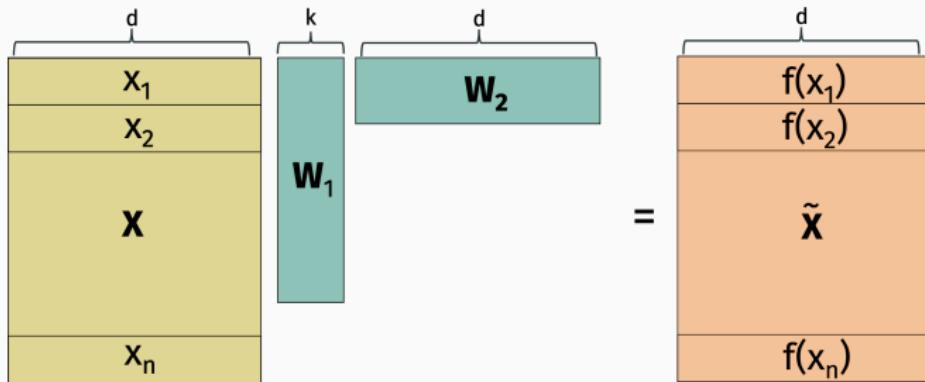


$$\text{Encoder: } e(\vec{x}) = \vec{x}^\top \mathbf{W}_1.$$

$$\text{Decoder: } d(\vec{z}) = \vec{z} \mathbf{W}_2$$

PRINCIPAL COMPONENT ANALYSIS

Given training data set $\vec{x}_1, \dots, \vec{x}_n$, let X denote our data matrix.
Let $\tilde{X} = XW_1W_2$.



Goal of training autoencoder: Learn weights (i.e. learn matrices W_1W_2) so that \tilde{X} is as close to X as possible.

FROBENIUS NORM

Natural squared autoencoder loss: Minimize $L(\mathbf{X}, \tilde{\mathbf{X}})$ where:

$$\begin{aligned} L(\mathbf{X}, \tilde{\mathbf{X}}) &= \sum_{i=1}^n \|\vec{x}_i - f(\vec{x}_i)\|_2^2 \\ &= \sum_{i=1}^n \sum_{j=1}^d (\vec{x}_i[j] - f(\vec{x}_i)[j])^2 \\ &= \|\mathbf{X} - \tilde{\mathbf{X}}\|_F^2 \end{aligned}$$

Recall that for a matrix \mathbf{M} , $\|\mathbf{M}\|_F^2$ is called the Frobenius norm.

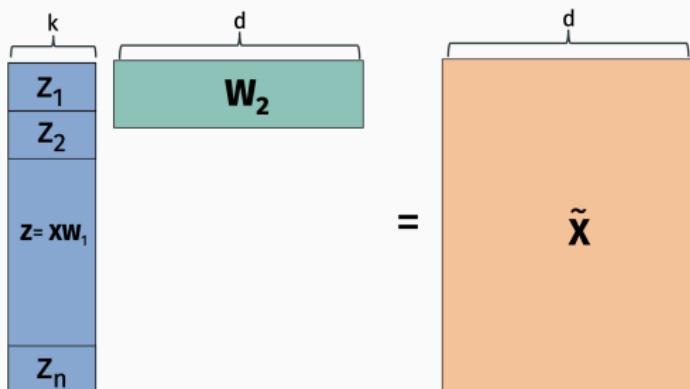
$$\|\mathbf{M}\|_F^2 = \sum_{i,j} \mathbf{M}_{i,j}^2.$$

Question: How should we find $\mathbf{W}_1, \mathbf{W}_2$ to minimize
 $\|\mathbf{X} - \tilde{\mathbf{X}}\|_F^2 = \|\mathbf{X} - \mathbf{X}\mathbf{W}_1\mathbf{W}_2\|_F^2?$

LOW-RANK APPROXIMATION

Recall:

- The columns of a matrix with column rank k can all be written as linear combinations of just k columns.
- The rows of a matrix with row rank k can all be written as linear combinations of k rows.
- Column rank = row rank = **rank**.



\tilde{X} is a **low-rank matrix** since it has rank k for $k \ll d$.

LOW-RANK APPROXIMATION

Principal component analysis is the task of finding W_1, W_2 , which amounts to finding a rank k matrix \tilde{X} which approximates the data matrix X as closely as possible.

In general, X will have rank d .

SINGULAR VALUE DECOMPOSITION

Any matrix X can be written:

$$\begin{matrix} d \\ \text{ } \\ \text{ } \\ \text{ } \\ n \end{matrix} \quad X = \begin{matrix} \text{left singular vectors} \\ \text{ } \\ \text{ } \\ \text{ } \\ \text{ } \end{matrix} \quad \begin{matrix} \sigma_1 & & & \\ & \Sigma & & \\ & & \sigma_2 & \\ & & & \ddots \\ & & & & \sigma_{d-1} \\ & & & & & \sigma_d \end{matrix} \quad \begin{matrix} \text{singular values} \\ \text{ } \\ \text{ } \\ \text{ } \\ \text{right singular vectors} \end{matrix}$$

Where $U^T U = I$, $V^T V = I$, and $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_d \geq 0$. I.e. U and V are orthogonal matrices.

This is called the **singular value decomposition**.

Can be computed in $O(nd^2)$ time (faster with approximation algos).

ORTHOGONAL MATRICES

Let $\mathbf{u}_1, \dots, \mathbf{u}_n \in \mathbb{R}^n$ denote the columns of \mathbf{U} . I.e. the top left singular vectors of \mathbf{X} .

$$\mathbf{U}^T \quad \mathbf{U} = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\|\mathbf{u}_i\|_2^2 =$$

$$\mathbf{u}_i^T \mathbf{u}_j =$$

SINGULAR VALUE DECOMPOSITION

Can read off optimal low-rank approximations from the SVD:

$$\begin{matrix} d \\ \text{---} \\ n \end{matrix} \quad X_k = \begin{matrix} \text{left singular vectors} \\ U_k \end{matrix} \quad \begin{matrix} \text{singular values} \\ \Sigma_k \end{matrix} \quad \begin{matrix} \text{right singular vectors} \\ V_k^T \end{matrix}$$

Eckart–Young–Mirsky Theorem: For any $k \leq d$, $X_k = U_k \Sigma_k V_k^T$ is the optimal k rank approximation to X :

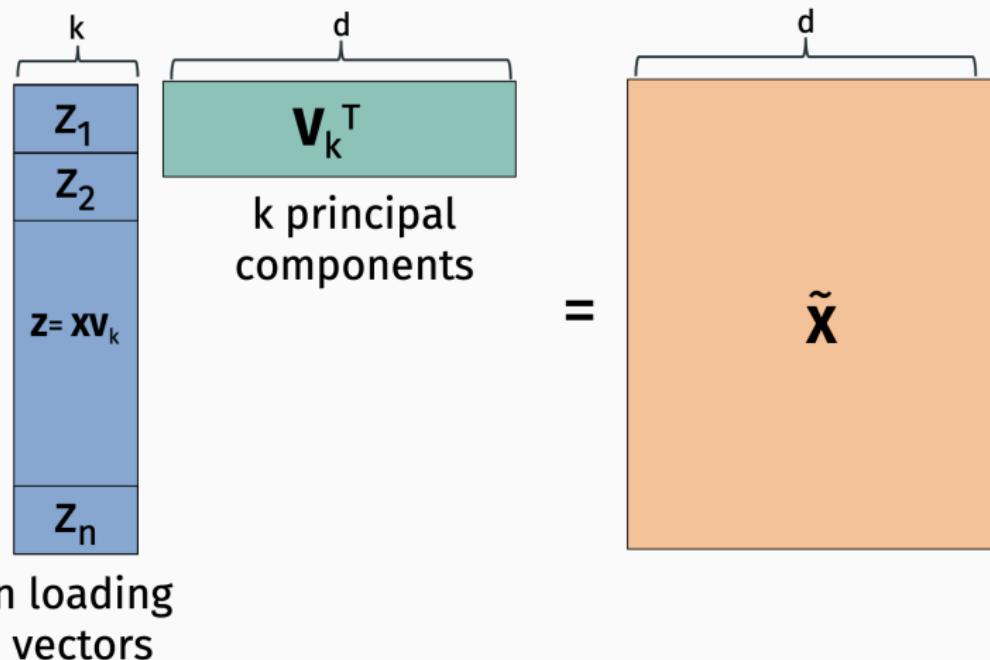
$$X_k = \underset{\tilde{X} \text{ with rank } \leq k}{\arg \min} \|X - \tilde{X}\|_F^2.$$

SINGULAR VALUE DECOMPOSITION

Claim: $\mathbf{X}_k = \mathbf{U}_k \boldsymbol{\Sigma}_k \mathbf{V}_k^T = \mathbf{X} \mathbf{V}_k \mathbf{V}_k^T$.

So for a model with k hidden variables, we obtain an optimal autoencoder by setting $\mathbf{W}_1 = \mathbf{V}_k$, $\mathbf{W}_2 = \mathbf{V}_k^T$. $f(\vec{x}) = \vec{x} \mathbf{V}_k \mathbf{V}_k^T$.

PRINCIPAL COMPONENT ANALYSIS



To be continued...