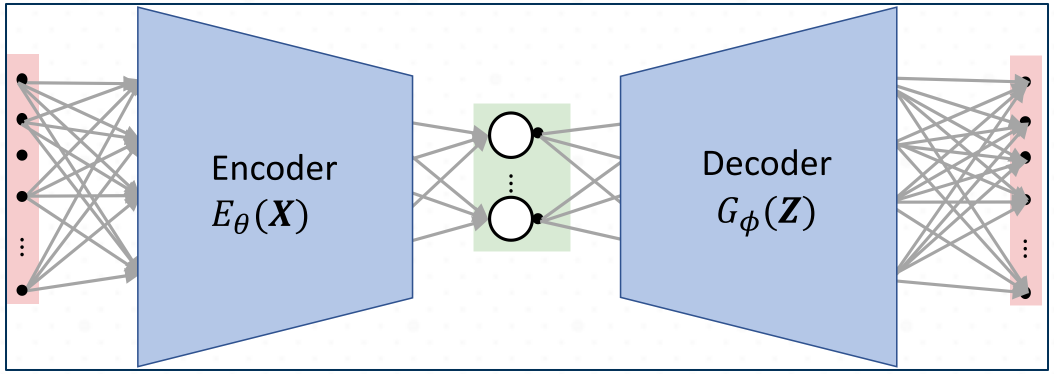
ECE 4252/8803: Fundamentals of Machine Learning (FunML)
Fall 2024



Lecture 18: Autoencoders

Overview

In this Lecture..

AEs: Introduction and Motivation

- Unsupervised Learning
- Autoencoders

Fully-connected Autoencoders

Convolutional Autoencoders

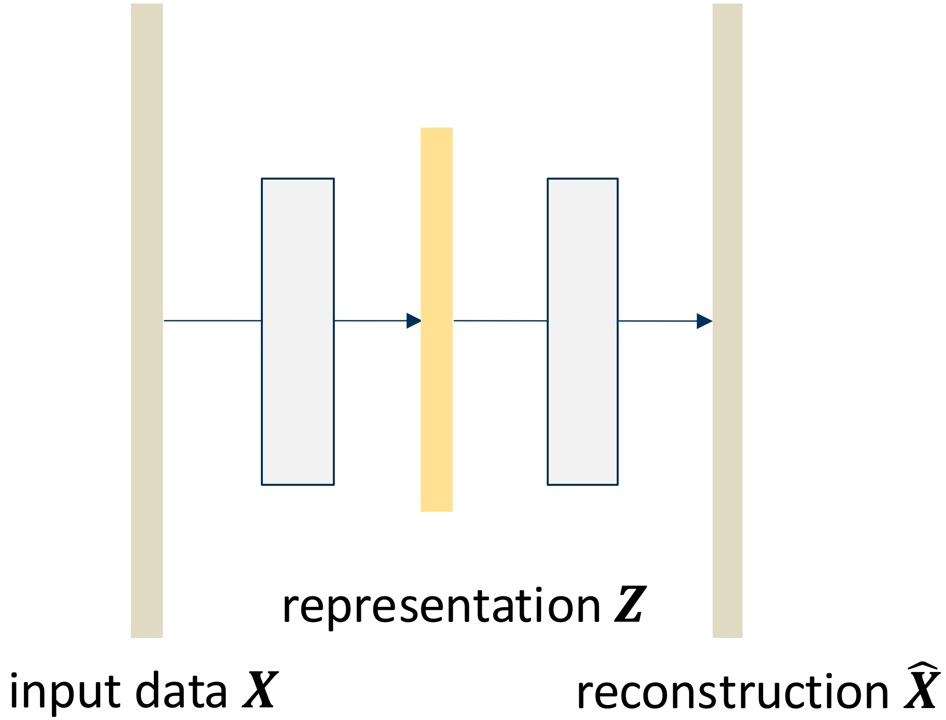
Regularized Autoencoders

Variational Autoencoders

Types of Learning

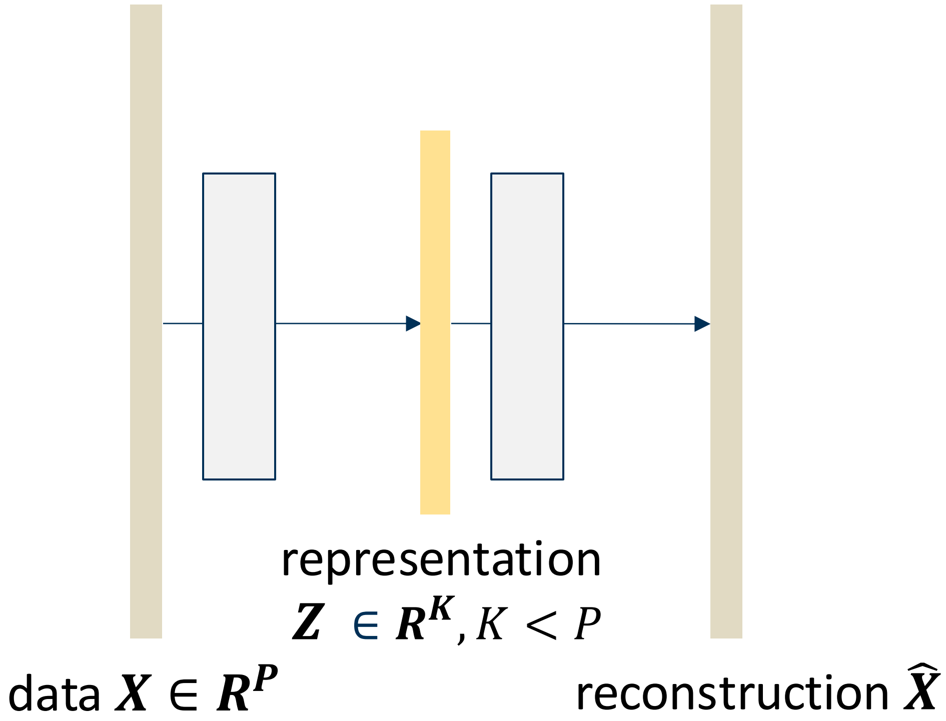
- Supervised Learning
  - Requires labeled training data $(\boldsymbol{x}_1, y_1), \dots, (\boldsymbol{x}_N, y_N)$
  - Goal: Learn a *mapping function* $f_{\boldsymbol{\theta}}: \mathcal{X} \to \mathcal{Y}$
  - Examples:
    - Classification, regression, etc.
- Unsupervised Learning
  - No labels required, only data $\boldsymbol{x}_1, \dots, \boldsymbol{x}_N$
  - Goal: Learn some underlying hidden ***structure*** of the data
  - Examples:
    - Clustering, dimensionality reduction, density estimation, etc.

Unsupervised approach for learning a ***lower-dimensional latent representation*** from the **unlabeled data** and reconstructing the data from this representation



Introduction
Autoencoders: Motivation

- **Unsupervised** learning
  - Learn a **lower-dimensional** representation $\boldsymbol{Z}$
  - Reconstruct the data
- $\boldsymbol{Z}$ describes the underlying **structure** of $\boldsymbol{X}$ that can be used for:
  - Visualization
    - Map $\boldsymbol{X}$ to $\boldsymbol{Z} \in \boldsymbol{R}^2$
  - **Dimensionality reduction**
    - Compress images, documents, text … etc.
  - Subsequent supervised tasks
    - Use $\boldsymbol{Z}$ to train/fine-tune for supervised tasks



Introduction

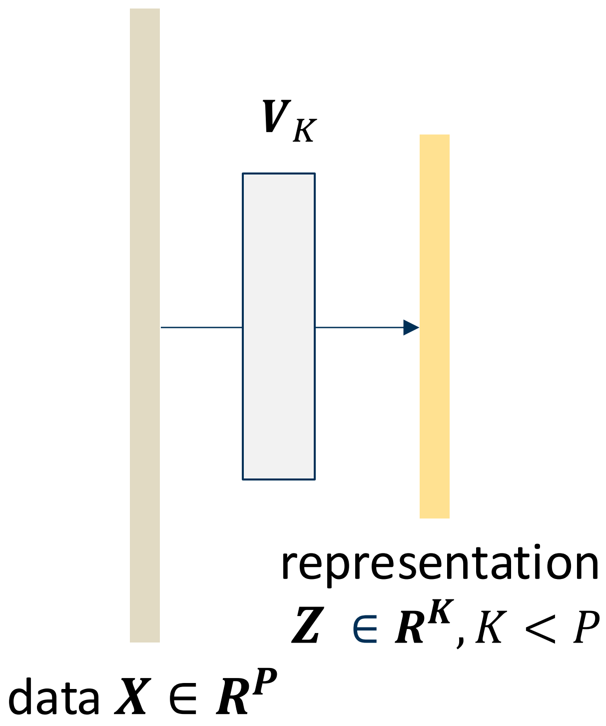
Autoencoders: Motivation

- Autoencoder: learning a lower-dimensional representation $\boldsymbol{Z}$ from unlabeled input data $\boldsymbol{X}$

- **Linear** dimensionality reduction by PCA :

$$\boldsymbol{Z} = \boldsymbol{X}\boldsymbol{V}_K$$

where $\boldsymbol{V}_K$ transforms $\boldsymbol{X}$ to a K-dimensional subspace (K<P)



Autoencoders: Motivation

- Autoencoder: learning a lower-dimensional representation $\boldsymbol{Z}$ from unlabeled input data $\boldsymbol{X}$

- **Linear** dimensionality reduction by PCA :

$$\boldsymbol{Z} = \boldsymbol{X}\boldsymbol{V}_K$$

where $\boldsymbol{V}_K$ transforms $\boldsymbol{X}$ to a K-dimensional subspace ($K<P$)

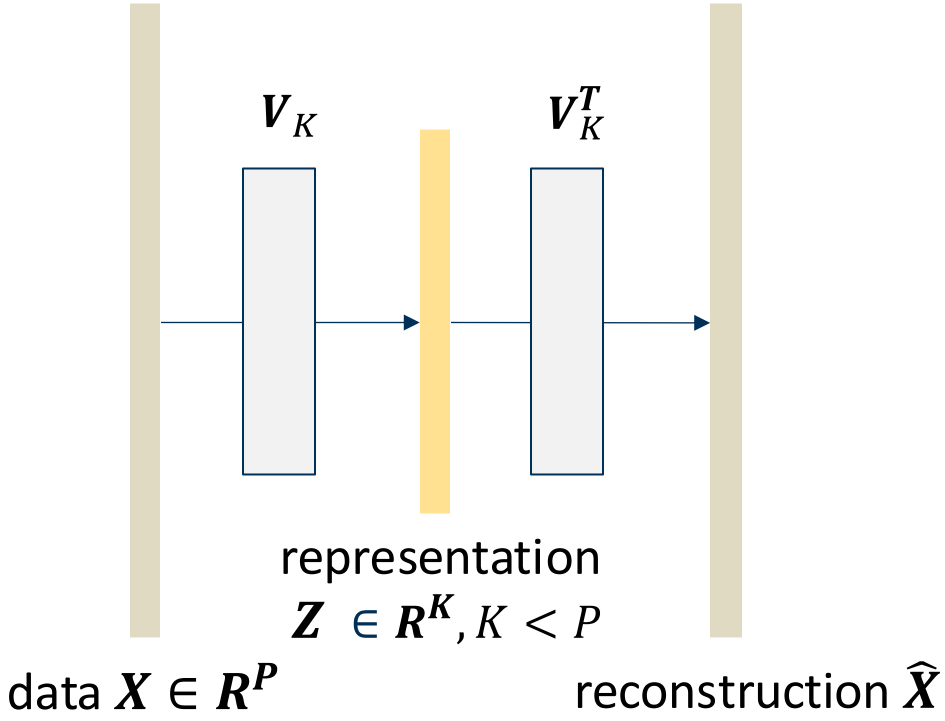
$\boldsymbol{X}$ can be reconstructed from $\boldsymbol{Z}$:

$$\widehat{\boldsymbol{X}} = \boldsymbol{X}\boldsymbol{V}_K\boldsymbol{V}_K^{\boldsymbol{T}}$$

Reconstruction error:

$$\left\|\boldsymbol{X} - \boldsymbol{X}\boldsymbol{V}_K\boldsymbol{V}_K^{\boldsymbol{T}}\right\|_{\boldsymbol{F}}$$

$\boldsymbol{V}_K$ maximizes the variance in $\boldsymbol{X}$ that has been preserved, while minimizing the reconstruction error



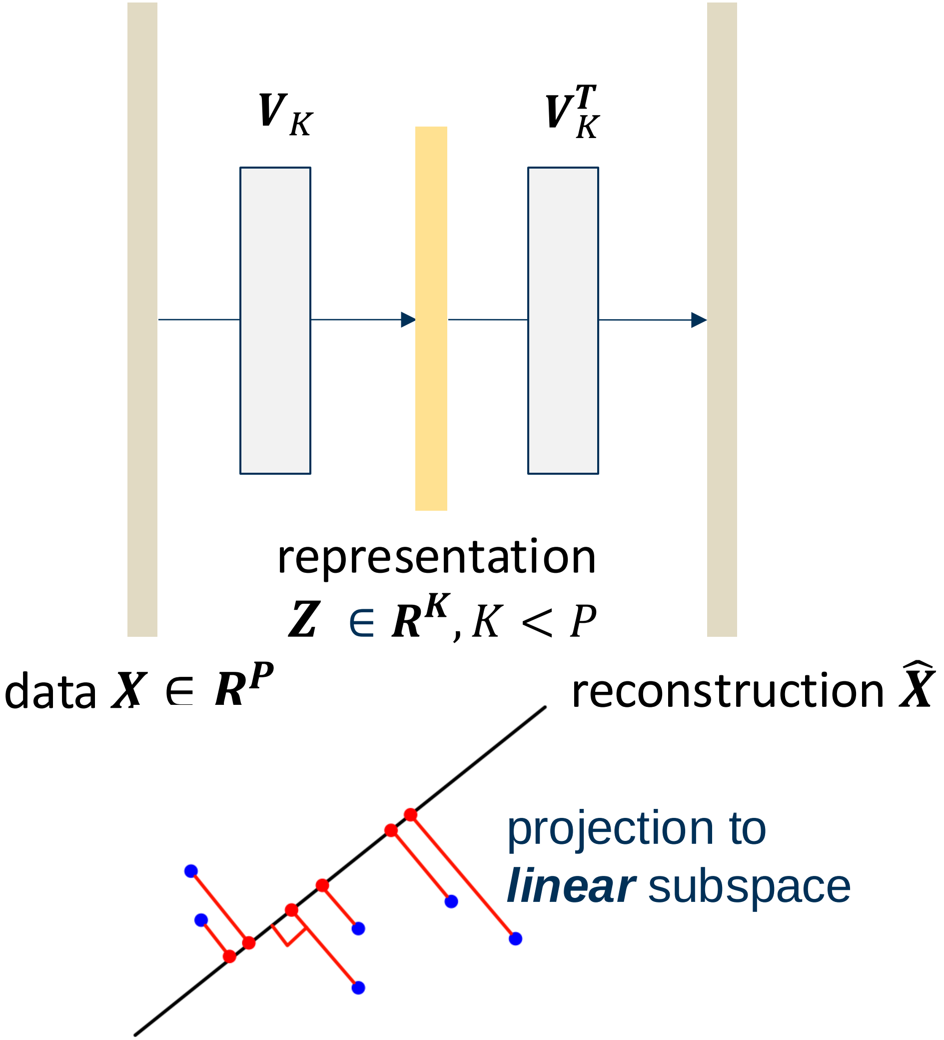
Introduction

Autoencoders: Motivation

- Autoencoder: learning a lower-dimensional representation $\boldsymbol{Z}$ from unlabeled input data $\boldsymbol{X}$
- **Linear** dimensionality reduction by PCA :

$$\boldsymbol{Z} = \boldsymbol{X}\boldsymbol{V}_K$$
$$\widehat{\boldsymbol{X}} = \boldsymbol{X}\boldsymbol{V}_K\boldsymbol{V}_K^T$$

- Linear autoencoder learns PCA when:
  - one fc layer with weights $\boldsymbol{V}_K$ and *linear* activation
  - one fc layer with weights $\boldsymbol{V}_K^T$ and *linear* activation
  - loss function $\boldsymbol{L}(\boldsymbol{X}, \widehat{\boldsymbol{X}}) = \|\boldsymbol{X} - \widehat{\boldsymbol{X}}\|_F$
  - **s.t.** $\boldsymbol{V}_K$ is the matrix of K eigenvectors of $\boldsymbol{X}^T\boldsymbol{X}$

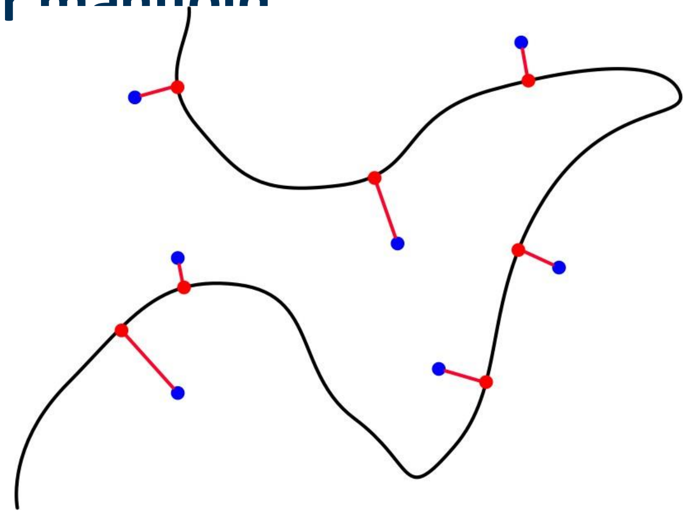


Introduction

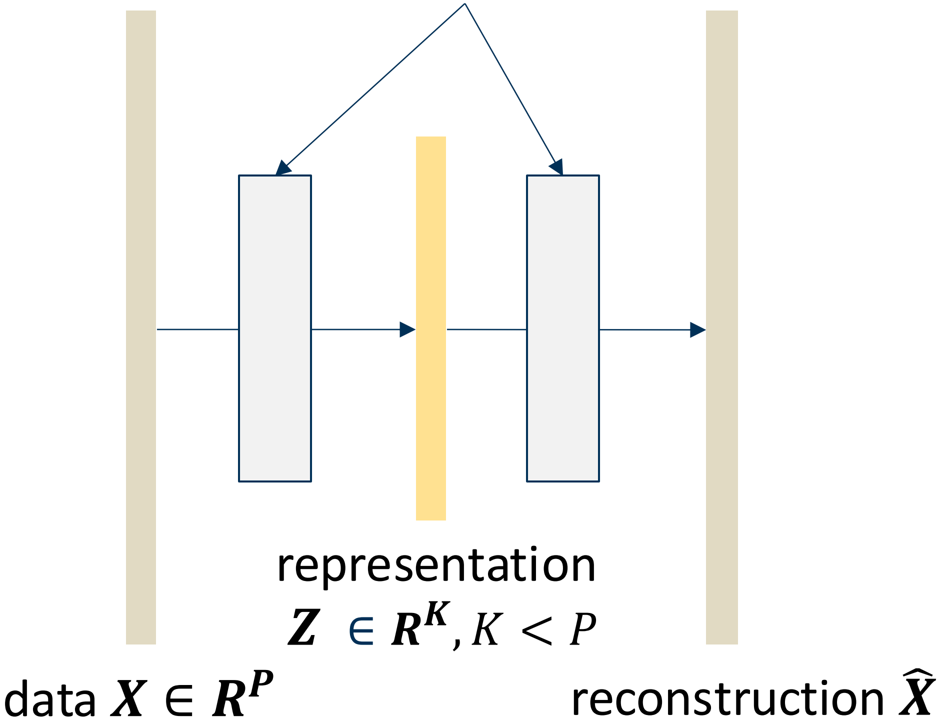
Autoencoders: Beyond PCA

- Autoencoder: learning a lower-dimensional representation $\boldsymbol{Z}$ from unlabeled input data $\boldsymbol{X}$

- Nonlinear autoencoders learn to project the data $\boldsymbol{X}$, not onto a linear subspace, but onto a **nonlinear manifold**



Neural networks with non-linear activations



Overview

In this Lecture..

AEs: Introduction and Motivation

Fully-connected Autoencoders

- Architecture
- Visualization

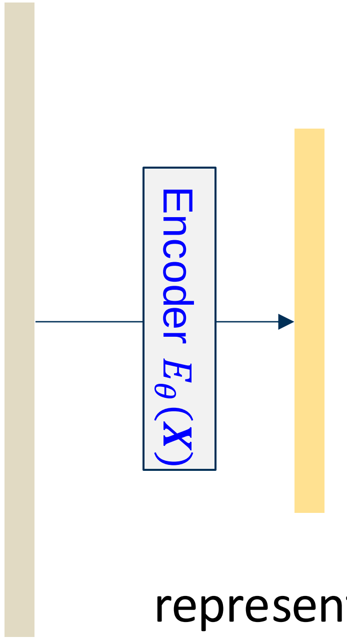
Convolutional Autoencoders

Regularized Autoencoders

Variational Autoencoders

Learning a lower-dimensional representation $\boldsymbol{Z}$ from unlabeled data $\boldsymbol{X}$

- an encoding function $E_\theta: \boldsymbol{X} \to \boldsymbol{Z}$

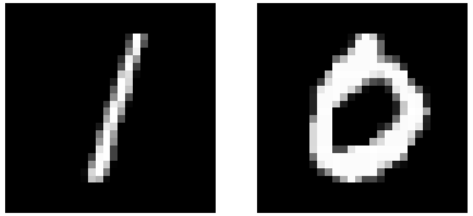


representation $\boldsymbol{Z}$

input data $\boldsymbol{X}$

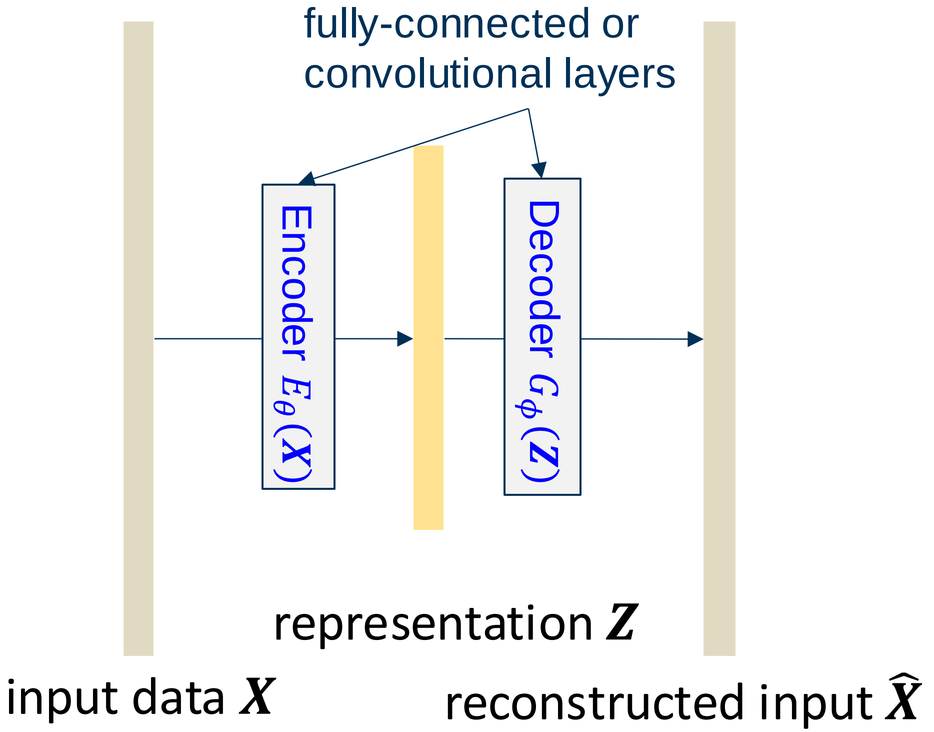
Want $\boldsymbol{Z}$ to capture meaningful patterns in data $\boldsymbol{X}$.

e.g., the digit “1” usually has a straight line, the digit “0” is circular



Learning a lower-dimensional representation $\boldsymbol{Z}$ from unlabeled data $\boldsymbol{X}$

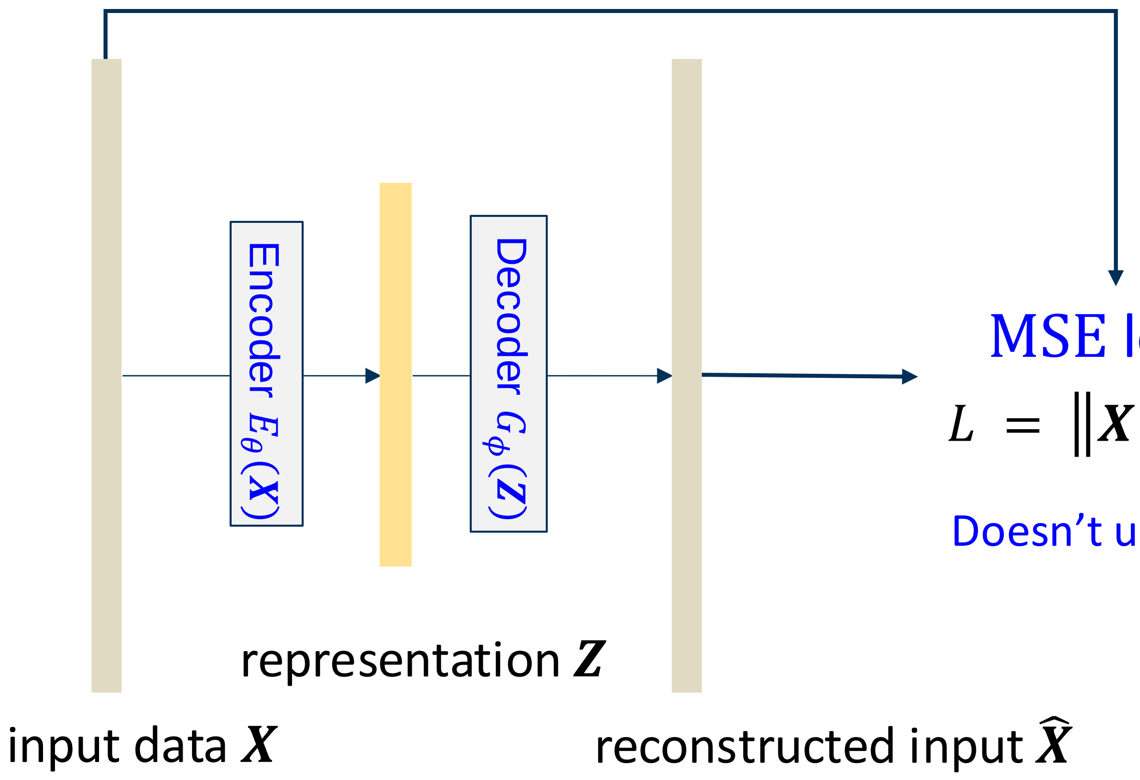
- an encoding function $E_\theta\colon \boldsymbol{X} \to \boldsymbol{Z}$
- a decoding function $G_\phi\colon \boldsymbol{Z} \to \widehat{\boldsymbol{X}}$



"Autoencoding" - encoding itself

How to learn this representation?

- Train such that representations can be used to reconstruct input data



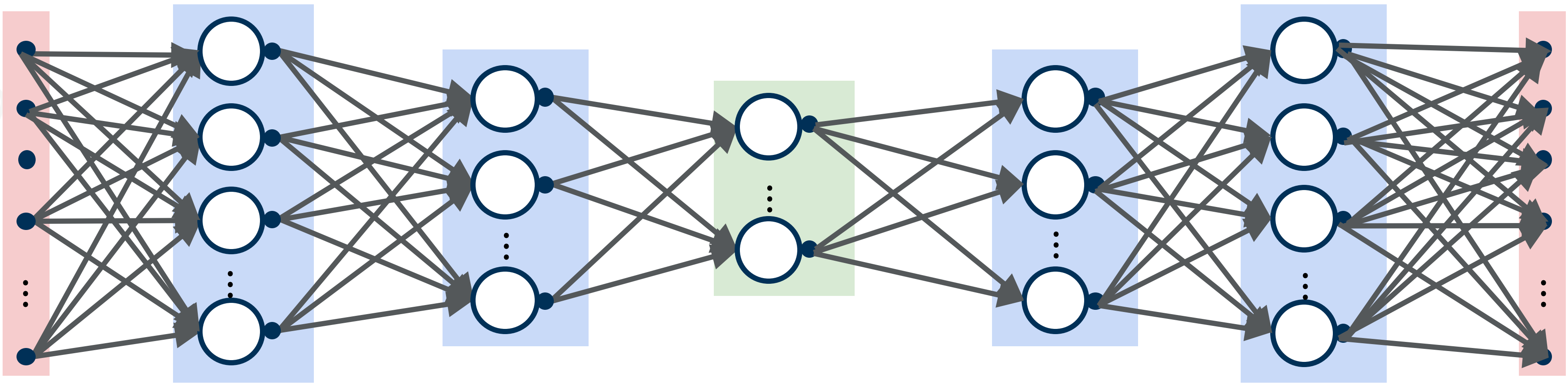
MSE loss :

$$L = \left\|\boldsymbol{X} - \widehat{\boldsymbol{X}}\right\|^2$$

Doesn't use labels!

Fully-connected Autoencoders
Architecture

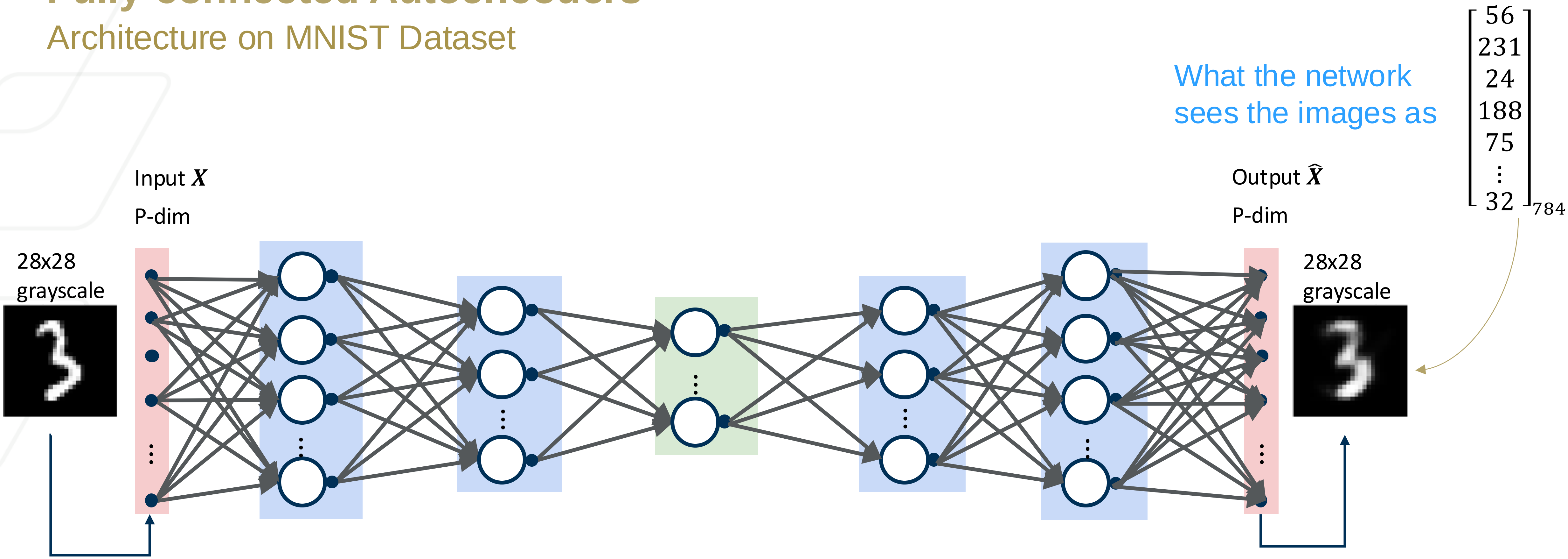
- Encoder and decoder are fully-connected layers



Original input $\boldsymbol{X}$ | Encoder $E_\theta(\boldsymbol{X})$ | Latent representation $\boldsymbol{Z}$ | Decoder $G_\phi(\boldsymbol{Z})$ | Reconstructed output $\widehat{\boldsymbol{X}}$

Fully-connected Autoencoders

Architecture on MNIST Dataset



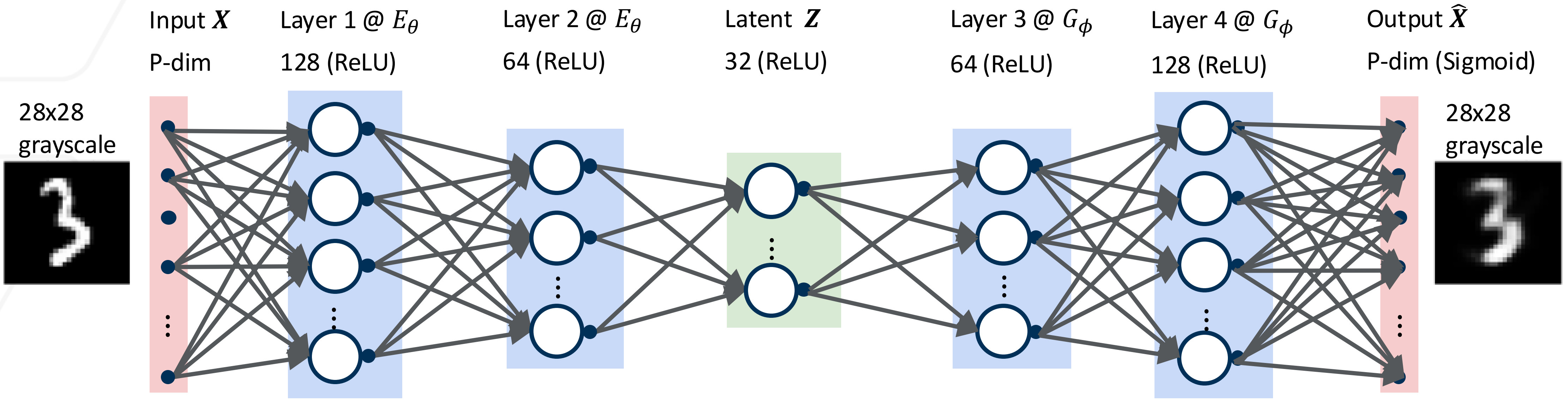
stretch pixels into P-dim vectors

reshape into 28x28 grayscale images

- P = 784 (28x28x1) for MNIST

Fully-connected Autoencoders

Architecture on MNIST Dataset



- P = 784 (28x28x1) for MNIST
- ReLU activation in the intermediate layers
- Sigmoid activation in the output layer

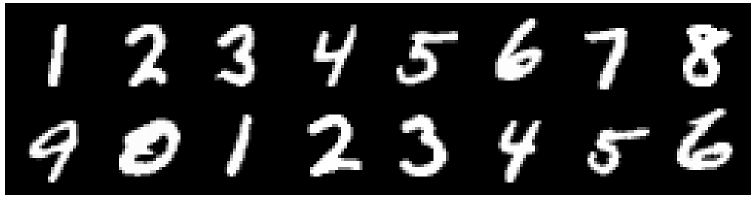
Fully-connected Autoencoders

Autoencoder Reconstructions on MNIST

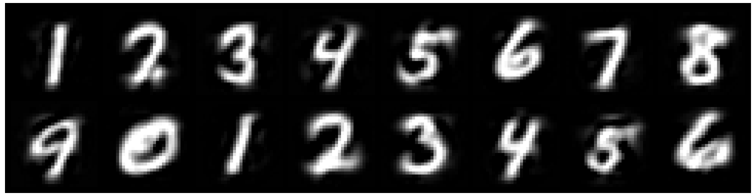
32-dimensional latent representation

Fully-connected AE (**linear**)

input

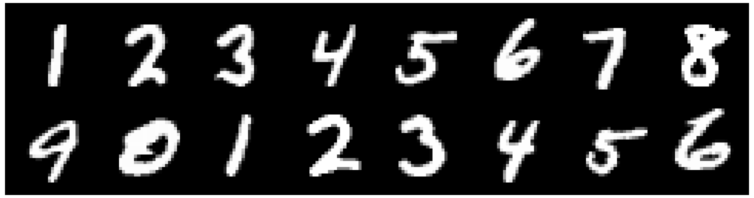


Reconstruction (MSE = 0.0677)

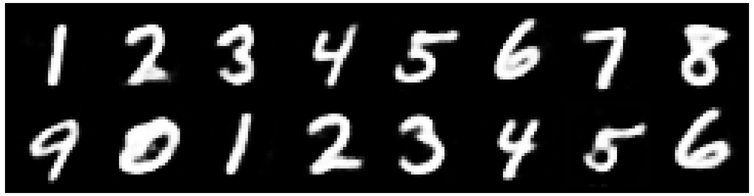


Fully-connected AE (**nonlinear**)

input



Reconstruction (MSE = 0.0306)

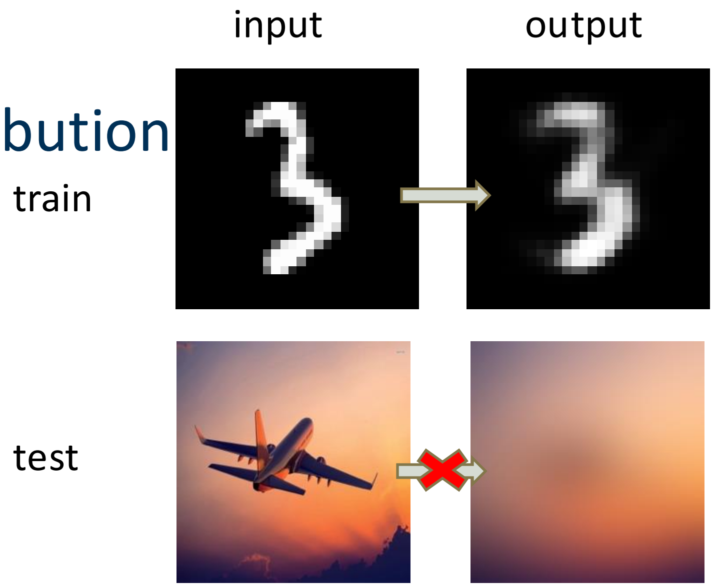


Linear AE: NOT using nonlinear activations; Nonlinear AE: using ReLU and Sigmoid

AE with nonlinearity learns more complex nonlinear latent representation to generate sharper reconstructions

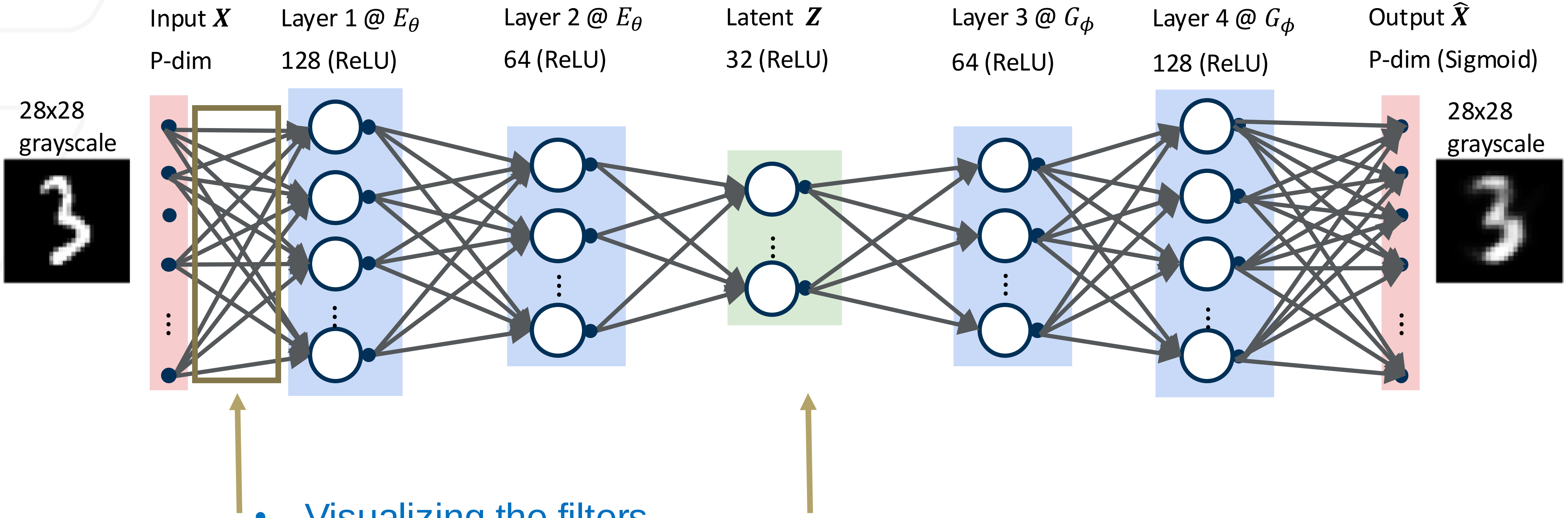
Fully-connected Autoencoders
Dimensionality Reduction

- *Unsupervised*: training without image labels
- *Lossy*: the reconstruction output is more blurry
- *Data-specific*: test and training data follow same distribution
  - Can be used for anomaly detection



Fully-connected Autoencoders

Visualization



- Visualizing the filters
- Visualizing the latent representations

Mathematical Notation:

- $\boldsymbol{X} \in \mathbb{R}^{N \times P}$: P-dimensional input vectors $\boldsymbol{x}_i \in \mathbb{R}^{\boldsymbol{P}}$, $i = 1 \dots N$
- $\boldsymbol{W}^{(k)} \in \mathbb{R}^{P^{(k)} \times P^{(k-1)}}$: weights matrix of neurons in layer k
- $P^{(k)}$: the number of neurons in layer k
- $P^{(k-1)}$: the number of neurons in layer $k-1$

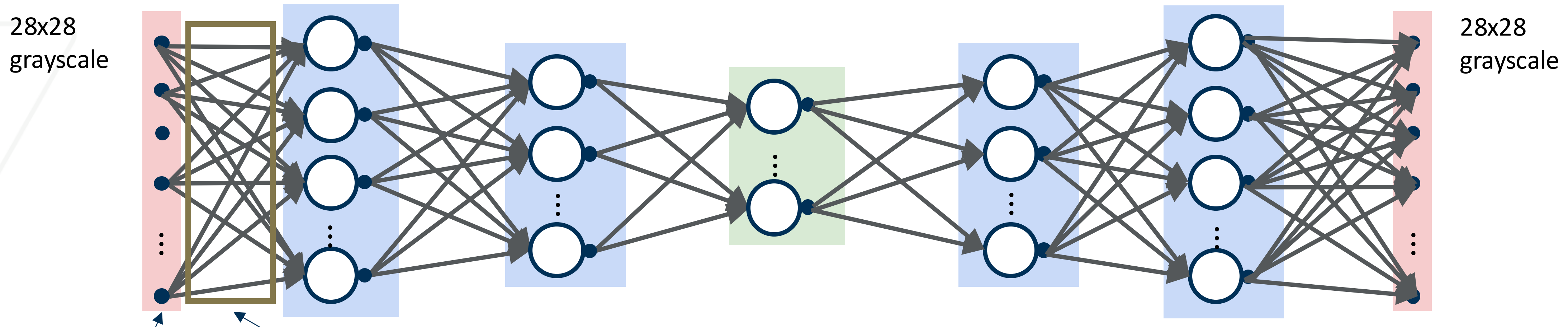
Visualization:

Extract the weights $\boldsymbol{W}^{(\boldsymbol{k})}$ (w/o bias) in layer k and reshape into an array of shape $(P^{(k)}, h, w, c)$, where $h \times w \times c = P^{(k-1)}$

Fully-connected Autoencoders

Visualization of Learned Filters

Extract $\boldsymbol{W}^{(1)}$ in the first layer of the fully-connected autoencoder



$\boldsymbol{x_i} \in \mathbb{R}^{(\mathbf{1}\times\mathbf{28}\times\mathbf{28})}$

$\boldsymbol{W}^{(\mathbf{1})} \in \mathbb{R}^{\mathbf{128}\times(\mathbf{1}\times\mathbf{28}\times\mathbf{28})}$ $\xrightarrow{\text{reshape}}$

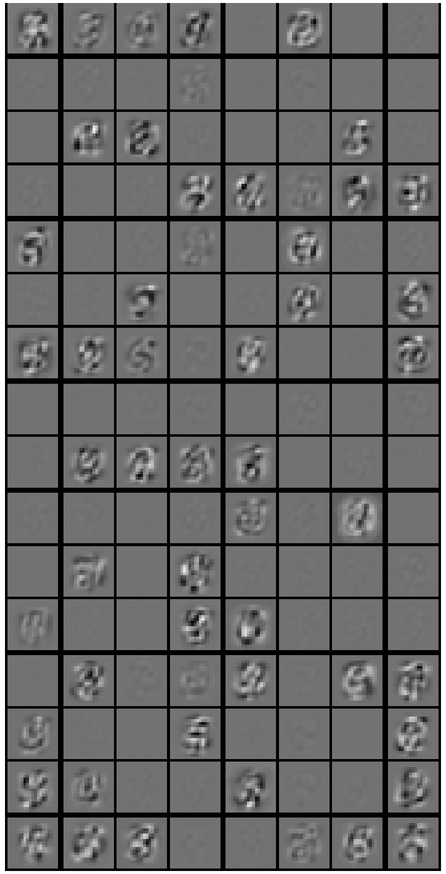
weights are visualized as
128 greyscale images of size 28x28

$\boldsymbol{W}^{(\mathbf{1})} \in \mathbb{R}^{(\mathbf{128})\times(\mathbf{28})\times(\mathbf{28})\times(\mathbf{1})}$

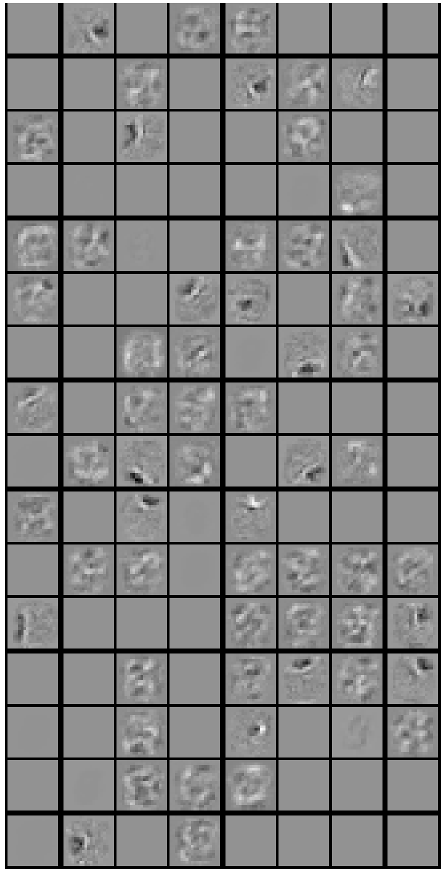
Fully-connected Autoencoders

Visualization of Learned Filters: First Layer

Fully-connected AE (**linear**)



Fully-connected AE (**nonlinear**)



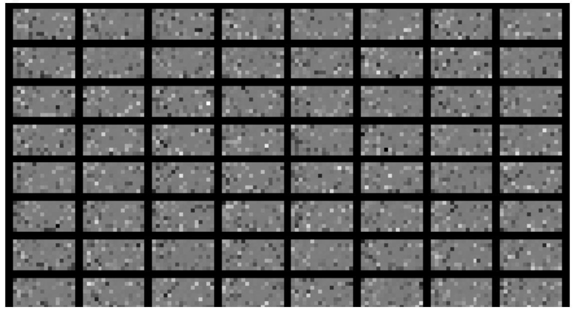
Extract the weights $\boldsymbol{W}^{(1)} \in \mathbb{R}^{128\times(1\times28\times28)}$ in **layer 1** @ E_θ

Reshape $\boldsymbol{W}^{(1)}$ into 128 filters of size 28x28

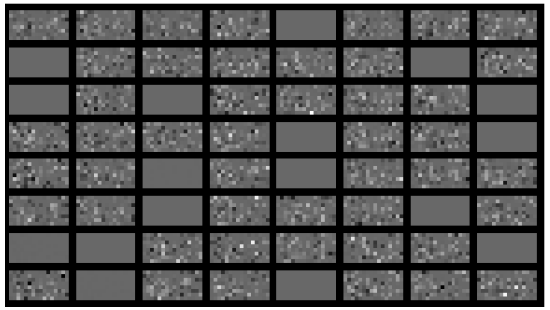
Fully-connected Autoencoders

Visualization of Learned Filters: Second and Third Layers

Fully-connected AE (**linear**)



Fully-connected AE (**nonlinear**)

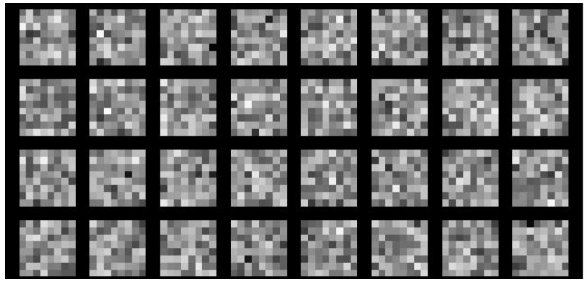


Extract the weights $\boldsymbol{W}^{(2)} \in \mathbb{R}^{\mathbf{64\times(1\times8\times16)}}$ in **layer 2** @ E_θ

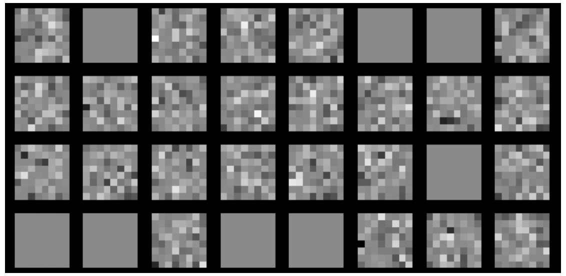
Reshape $\boldsymbol{W}^{(2)}$ into 64 filters of size 8x16

Takeaway: More neurons are utilized when we go deeper – not that interesting!

Fully-connected AE (**linear**)



Fully-connected AE (**nonlinear**)



Extract the weights $\boldsymbol{W}^{(3)} \in \mathbb{R}^{\mathbf{32\times(1\times8\times8)}}$ in **layer 3** @ E_θ

Reshape $\boldsymbol{W}^{(3)}$ into 32 filters of size 8x8

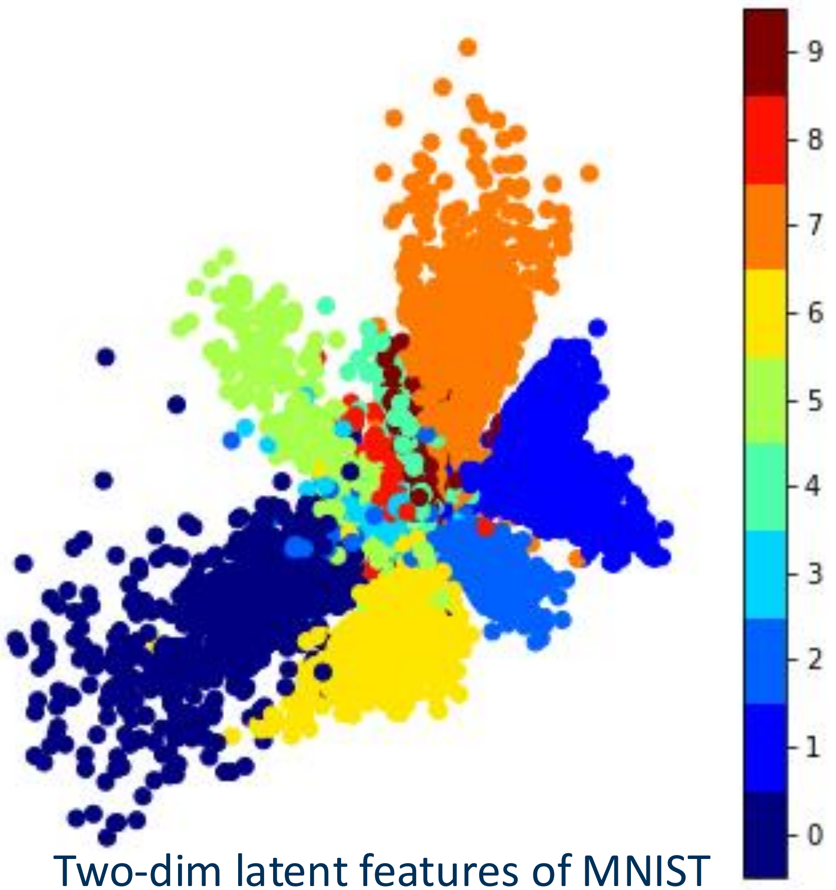
Fully-connected Autoencoders
Visualization of 2D Latent Space

We want to visualize the latent representations

Train a fc-autoencoder on MNIST:

- 2-dimensional latent representation
- visualize color-coded representations

(according to their classes)



Two-dim latent features of MNIST

Visualization of 2D Latent Space

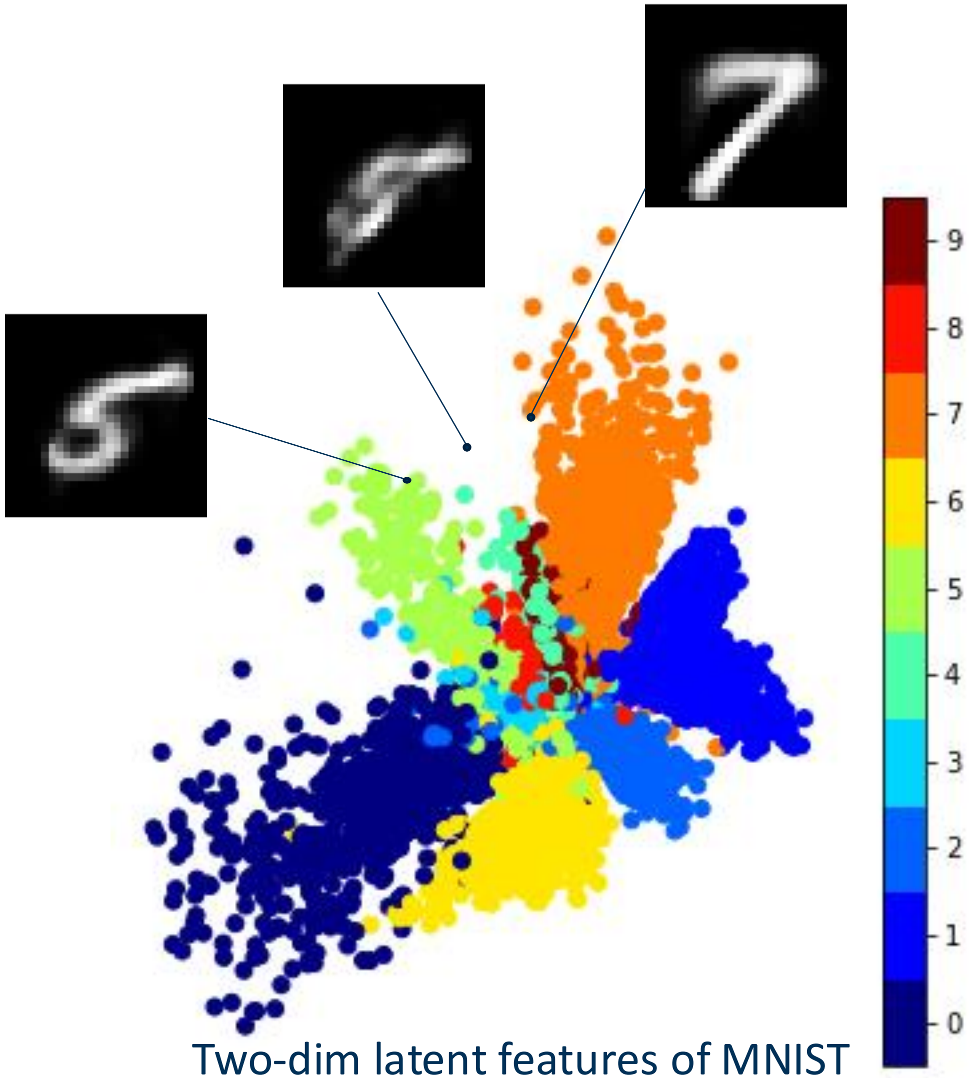
Train a fc-autoencoder on MNIST:

- 2-dimensional latent representation
- visualize color-coded representations

(according to their classes)

The observations on latent features:

- Weak discriminability
    - Unsupervised learning discovers data structures but not aims to separate different classes
- Discontinuity:
    - The “digit-like” image decoded from gap regions look unrealistic

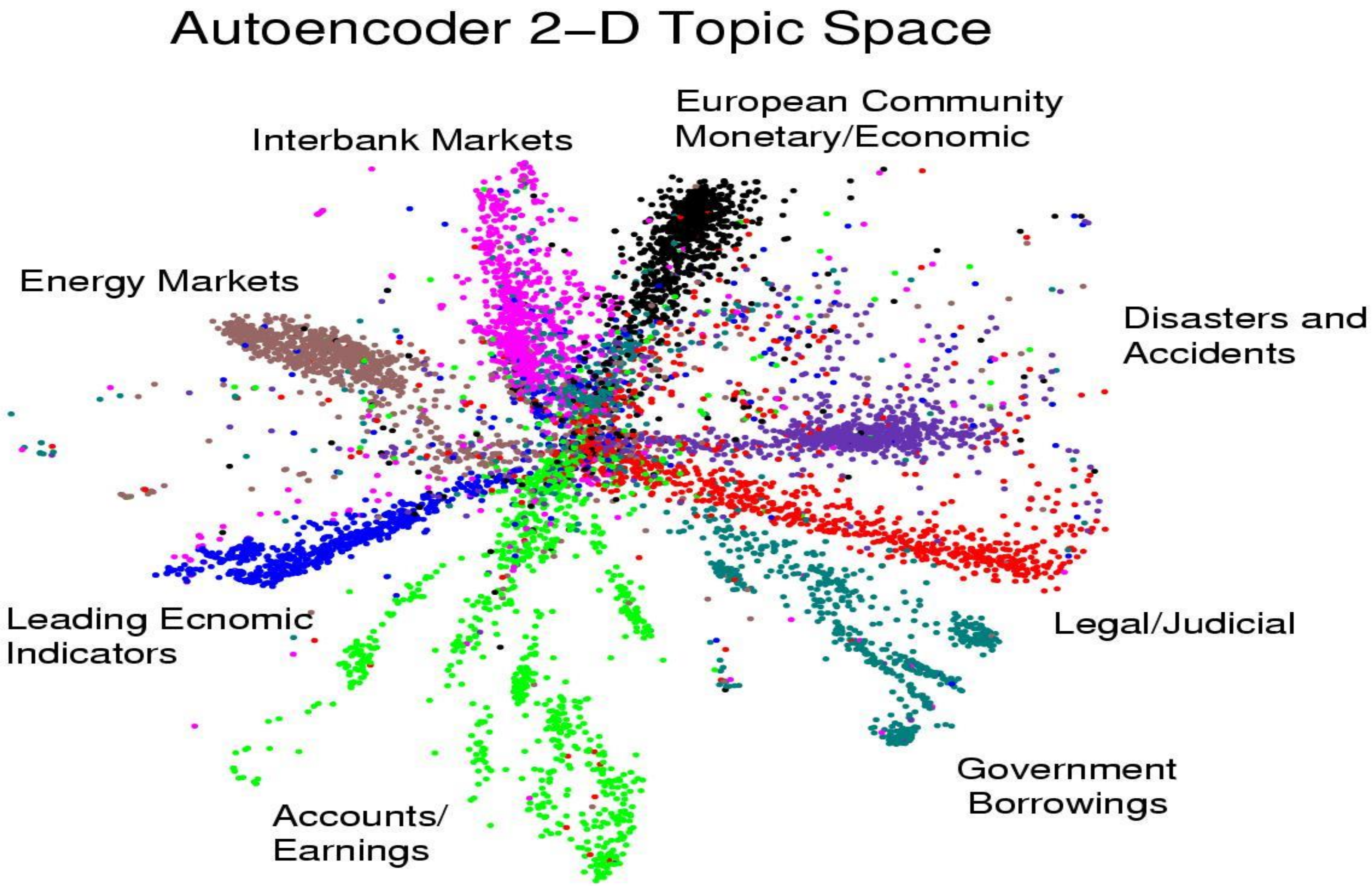


Two-dim latent features of MNIST

Fully-connected Autoencoders

Visualization of 2D Latent Space

- document categories



Overview

In this Lecture..

Introduction and Motivation

Fully-connected Autoencoders

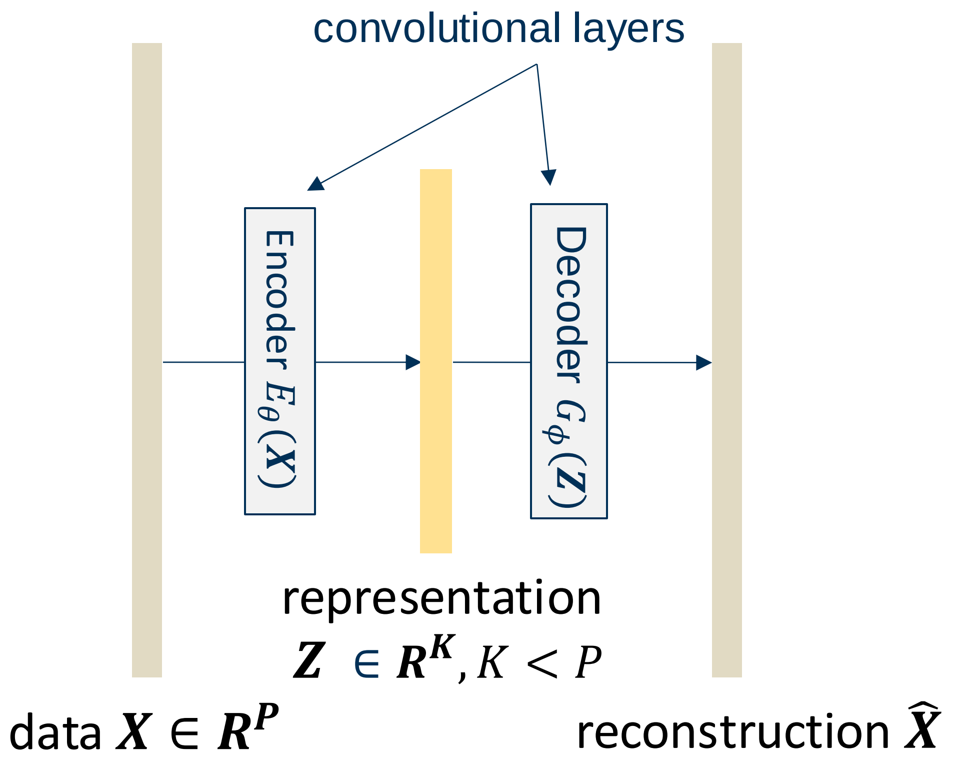
Convolutional Autoencoders

- Convolutional Filters
- Unpooling

Regularized Autoencoders

Variational Autoencoders

- Fully-connected autoencoders:
  - Does not leverage stationary
  - Redundancy in the parameters
  - Learn from every sample as an independent feature
- Convolutional autoencoders:
  - Preserving spatial locality
  - Sharing weights across different locations.
  - Reconstruction based on the local latent features



Autoencoders

Convolutional Autoencoders

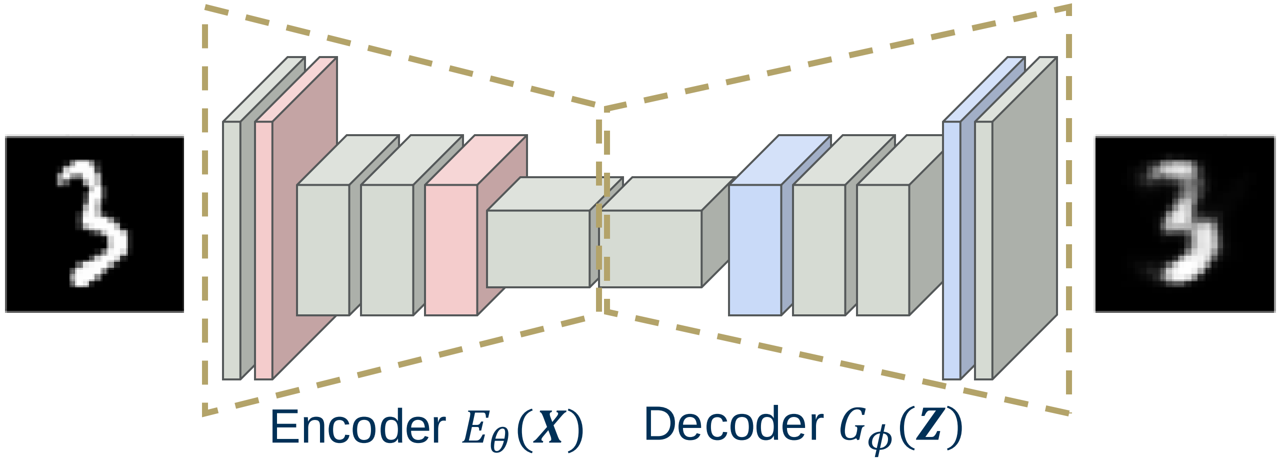
Encoder: (dimension decreases)

- Convolution (with stride)
- or Convolution + Pooling

Decoder: (dimension increases)

- Unpooling + Convolution
- or Transposed Convolution with stride

  (learning the “Unpooling + Convolution”)



Convolutional Autoencoders

Unpooling

Nearest Neighbor

The output feature map becomes blocky

Input 2x2

1	2
3	4

Output 4x4

1	1	2	2
1	1	2	2
3	3	4	4
3	3	4	4

Bed of “Nails”

The upsampled elements always have a fixed location

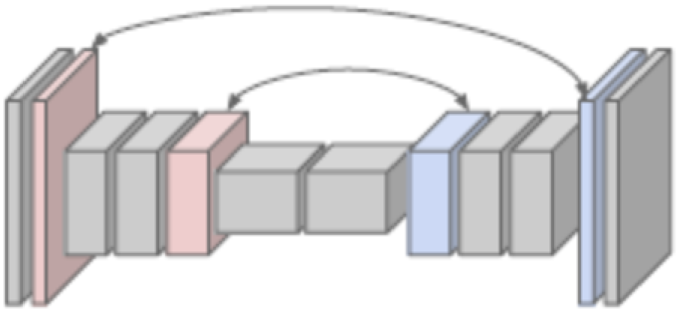
Input 2x2

1	2
3	4

Output 4x4

1	0	2	0
0	0	0	0
3	0	4	0
0	0	0	0

Convolutional Autoencoders
Max-unpooling with indices



Max Pooling in $E_\theta(\boldsymbol{X})$

Remember which element was max!

1	2	6	3
3	5	2	1
1	2	2	1
7	3	4	8

Input 4x4

5	6
7	8

Output 2x2

Indices: $\begin{bmatrix} 5 & 2 \\ 12 & 15 \end{bmatrix}$

Max Unpooling in $G_\phi(\boldsymbol{Z})$

Use positions from pooling layer

5	6
7	8

Input 2x2

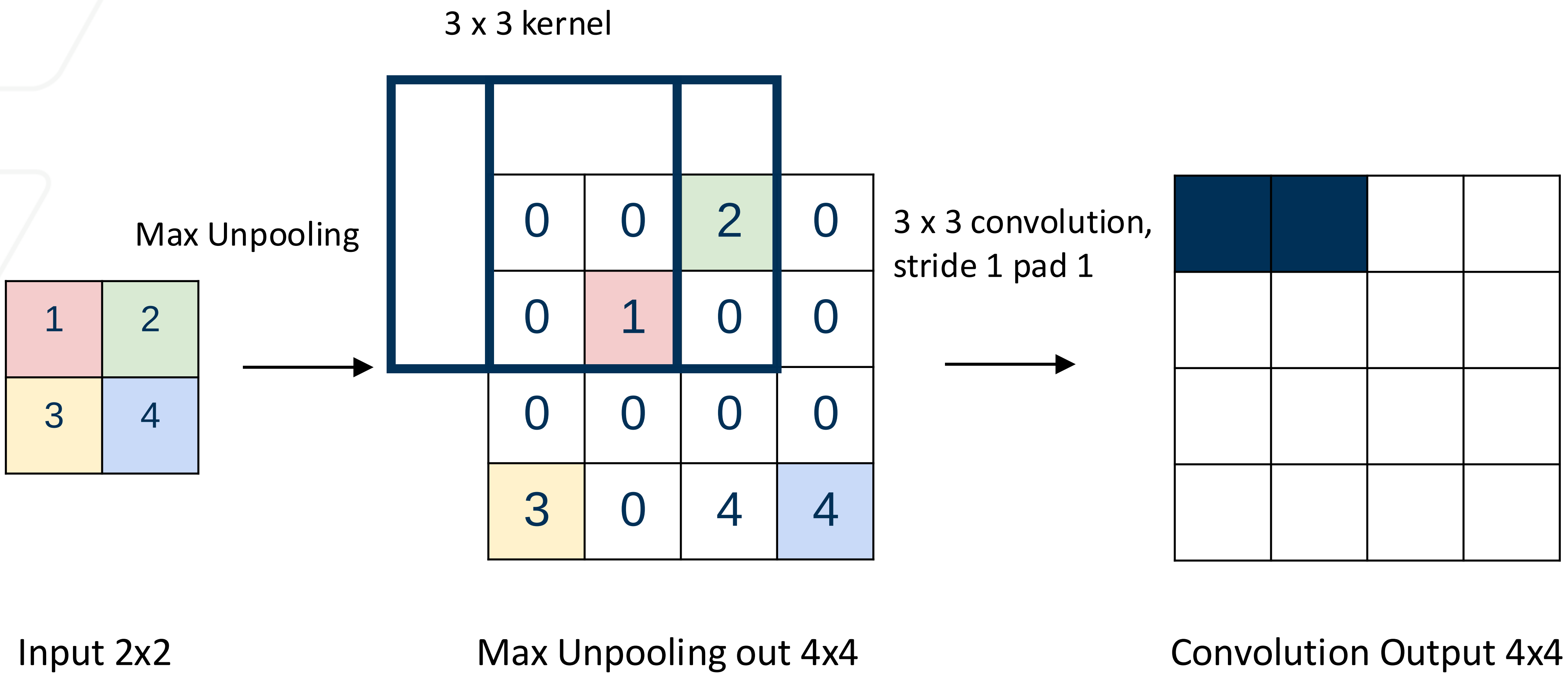
Indices: $\begin{bmatrix} 5 & 2 \\ 12 & 15 \end{bmatrix}$

0	0	6	0
0	5	0	0
0	0	0	0
7	0	4	8

Output 4x4

Decoder with convolutions

Max-unpooling with indices



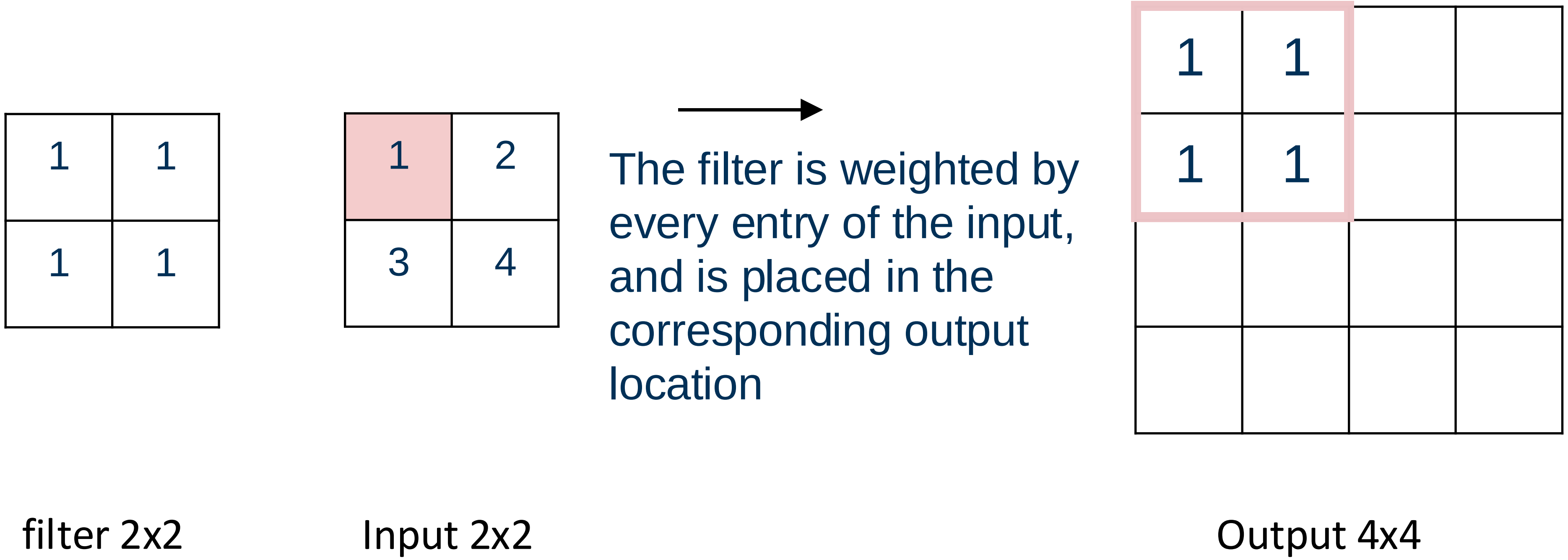
Learnable Upsampling

Transposed Convolution with Stride

2 x 2 **transposed** convolution, stride 2

learn “unpooling + convolution”

The filter is weighted by every entry of the input, and is placed in the corresponding output location



Learnable Upsampling

Transposed Convolution with Stride

2 x 2 **transposed** convolution, stride 2

1	1
1	1

filter 2x2

1	2
3	4

Input 2x2

Input gives weight for filter

1	1	2	2
1	1	2	2

Output 4x4

stride 2: moves 2 pixels in the output for every one pixel in the input

Learnable Upsampling

Transposed Convolution with Stride

2 x 2 **transposed** convolution, stride 2

1	1
1	1

filter 2x2

1	2
3	4

Input 2x2

Input gives weight for filter

1	1	2	2
1	1	2	2
3	3		
3	3		

Output 4x4

stride 2: moves 2 pixels in the output for every one pixel in the input

Learnable Upsampling

Transposed Convolution with Stride

2 x 2 **transposed** convolution, stride 2

1	1
1	1

filter 2x2

1	2
3	4

Input 2x2

Input gives weight for filter

1	1	2	2
1	1	2	2
3	3	4	4
3	3	4	4

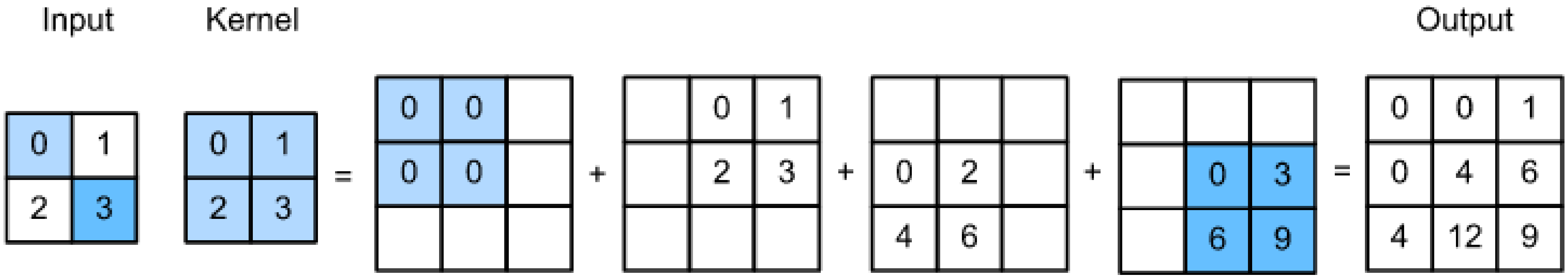
Output 4x4

stride 2: moves 2 pixels in the output for every one pixel in the input

Learnable Upsampling

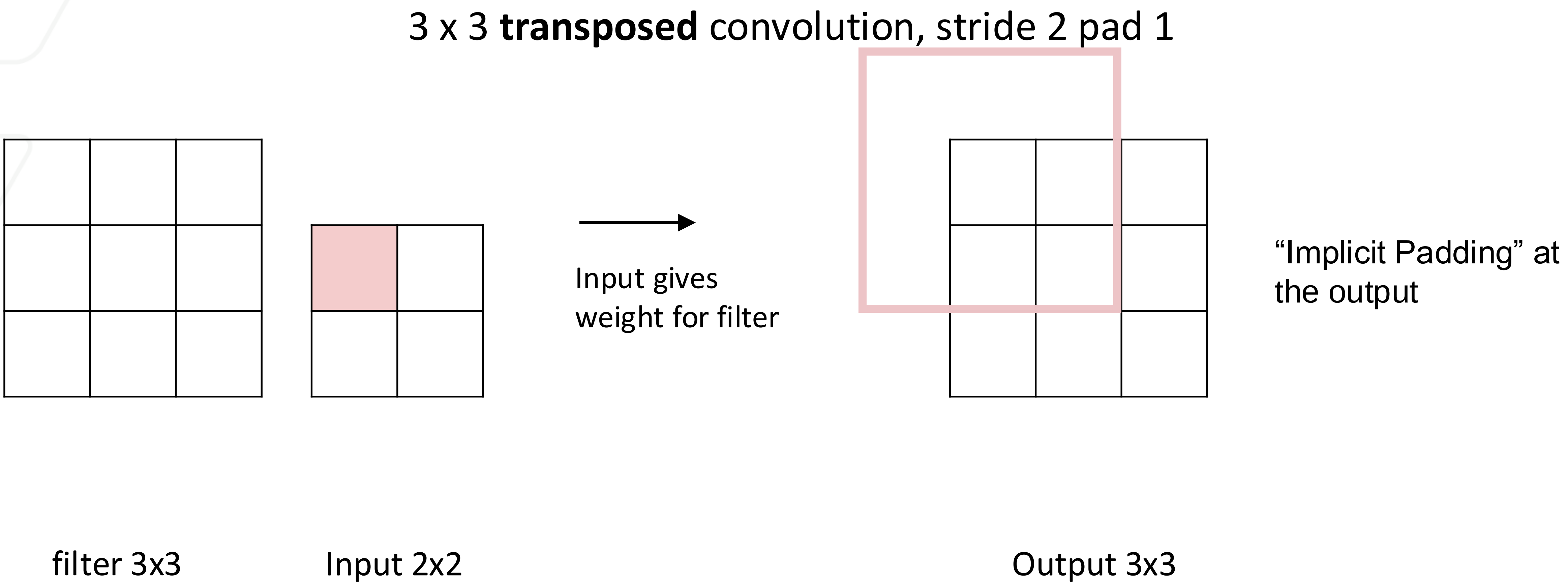
Transposed Convolution with Stride

2 x 2 **transposed** convolution, stride **1**



Learnable Upsampling

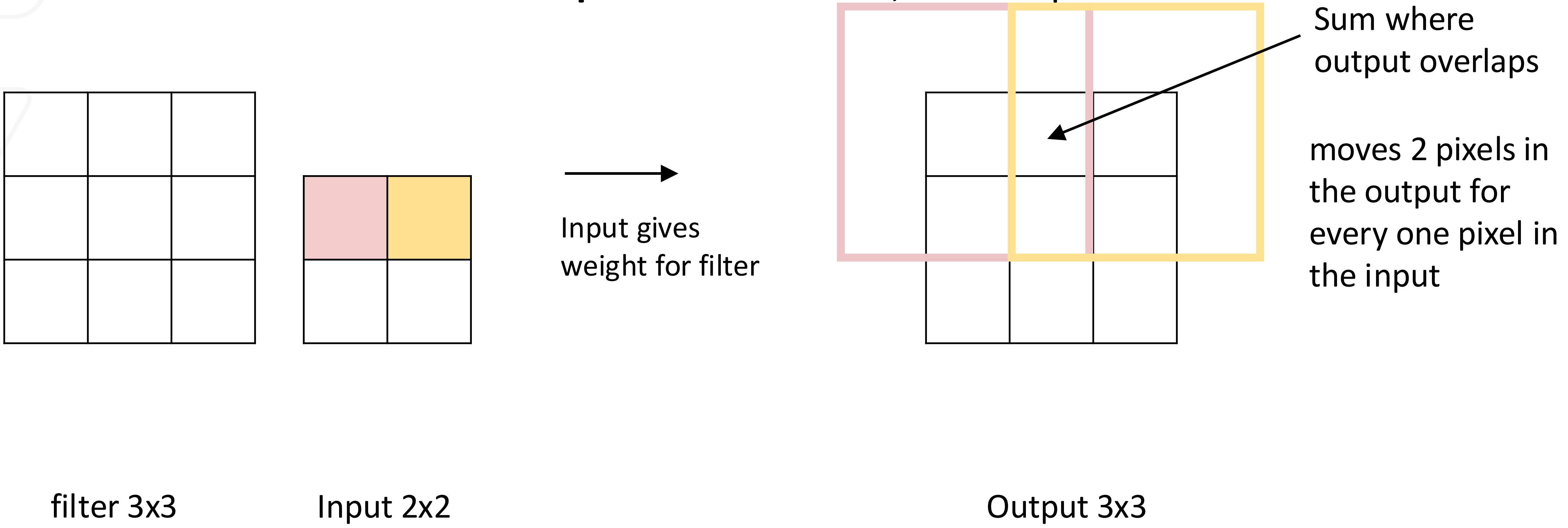
Transposed Convolution with Stride



Learnable Upsampling

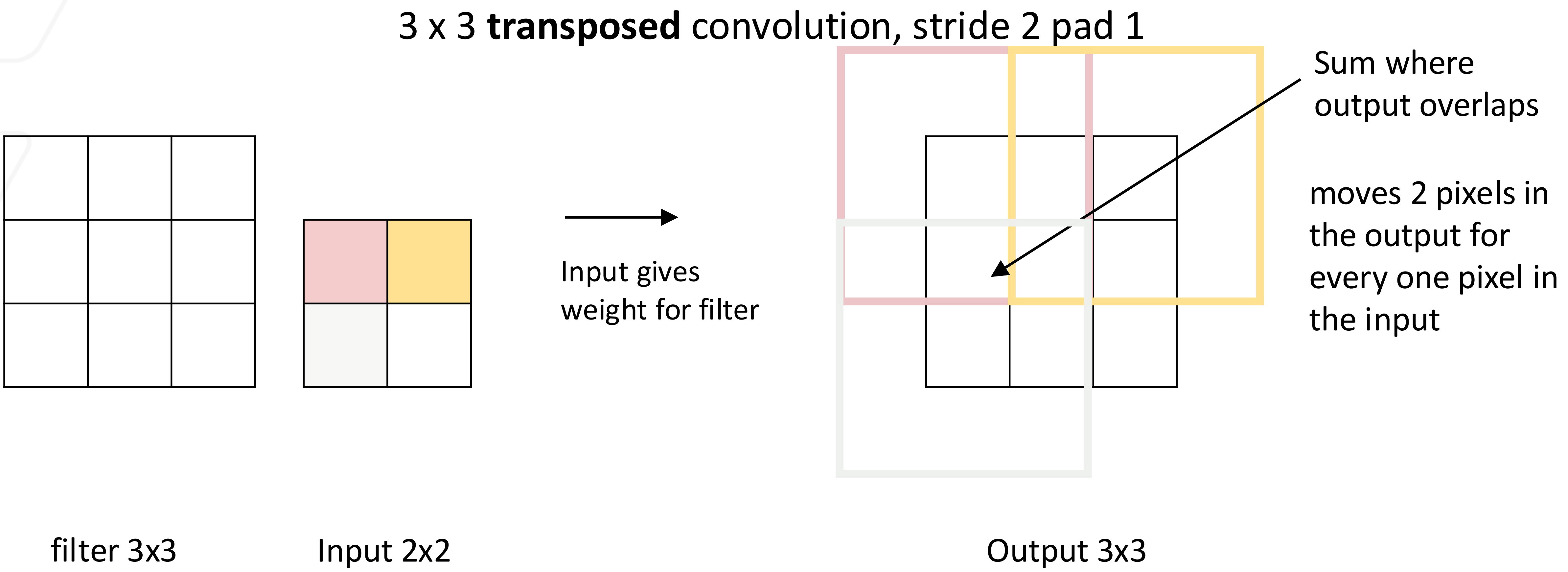
Transposed Convolution with Stride

3 x 3 **transposed** convolution, stride 2 pad 1



Learnable Upsampling

Transposed Convolution with Stride



Learnable Upsampling

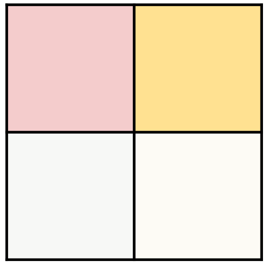
Transposed Convolution with Stride

learn “unpooling + convolution”

Transposed convolution is NOT the inverse of a convolution

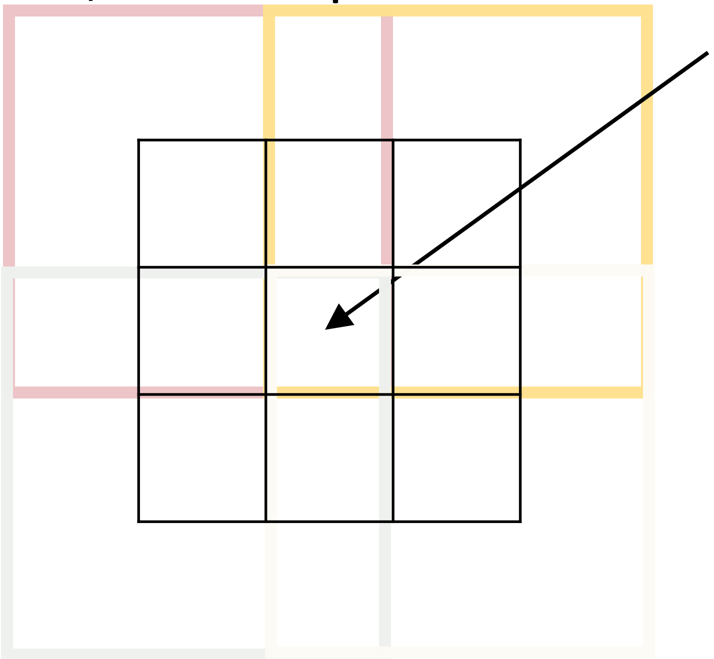
Other names:
-Upconvolution
-Fractionally strided convolution
-Backward strided convolution

3 x 3 **transposed** convolution, stride 2 pad 1



Input 2x2

Input gives weight for filter

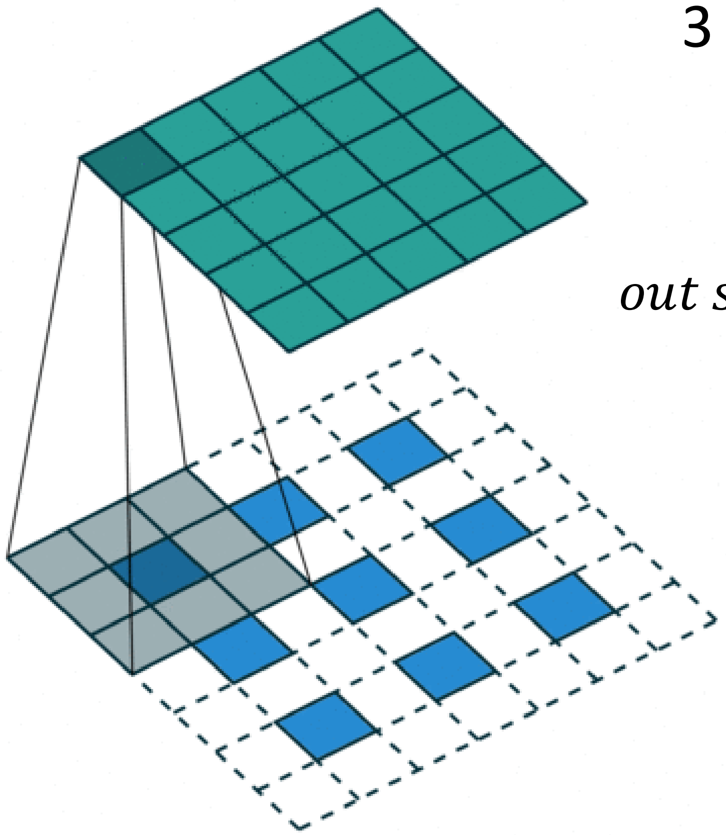


Sum where output overlaps

moves 2 pixels in the output for every one pixel in the input

Output 3x3 (“crop” 1 pixel at the border)

Transposed Convolution
Upsampling



3 x 3 **transposed** convolution, stride 2 pad 1

$$out\ size = stride \times (input\ size - 1) + filter\ size - 2 \times padding$$

$$5 = 2 \times (3 - 1) + 3 - 2 \times 1$$

Recall for convolutional operation:

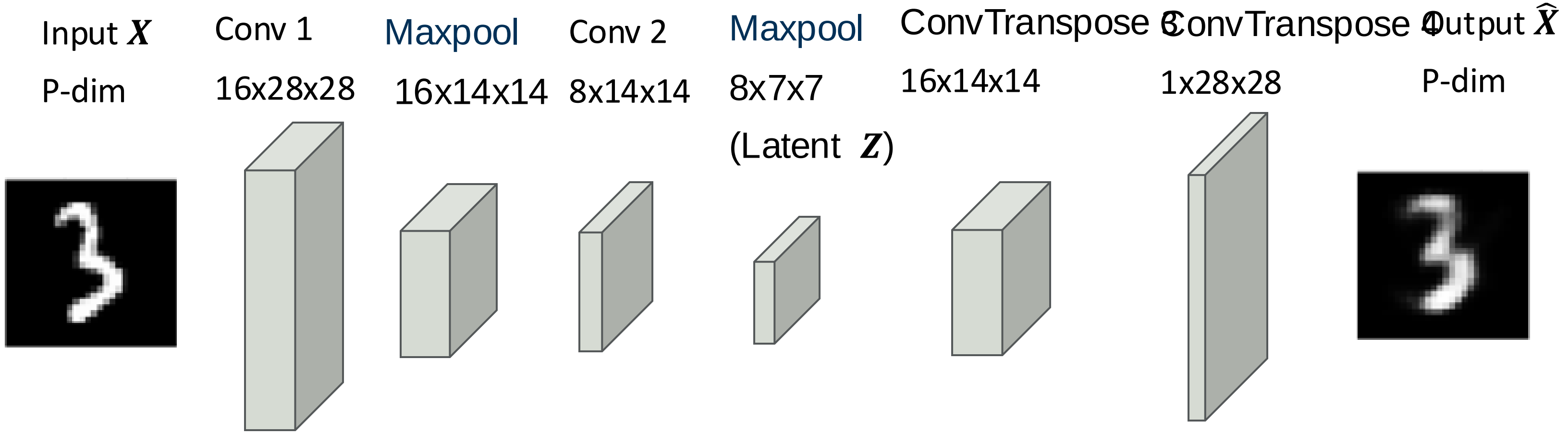
$$out\ size = \left\lfloor \frac{input\ size + 2 \times padding - filter\ size}{stride} + 1 \right\rfloor$$

Inputs: 3 x 3 blue maps, outputs: 5 x 5 green maps

Convolutional Autoencoder

Training on MNIST

Convolutional autoencoder:



Encoder:

- Conv 1: 16 kernels of 3x3, padding=1 (ReLU)
- Maxpooling: stride = 2
- Conv 2: 8 kernels of 3x3, padding=1 (ReLU)
- Maxpooling: with stride = 2

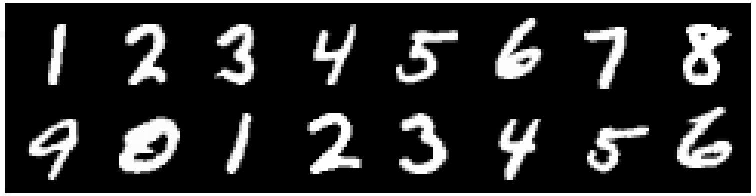
Decoder:

- Transposed conv 3: 16 kernels of 2x2, stride=2 (ReLU)
- Transposed conv 4: 1 kernels of 2x2, stride=2 (Sigmoid)

Experimental Results

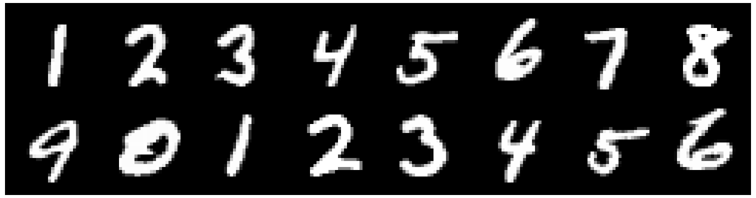
FC-linear AE vs FC-nonlinear AE vs Convolutional AE

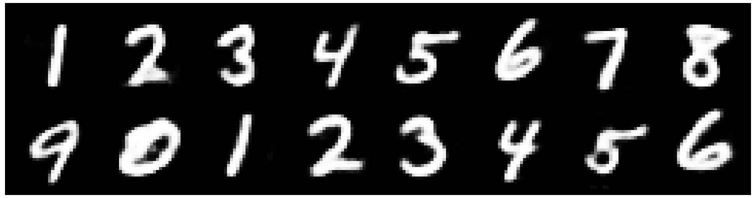
Fully-connected AE (linear)



MSE = 0.0677

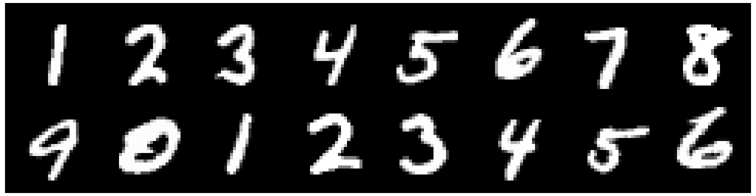
Fully-connected AE (nonlinear)

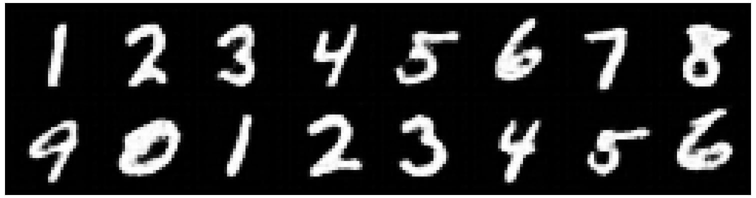




MSE = 0.0306

Convolutional AE





MSE = 0.0130

input (top row) and reconstruction (bottom row)

Terminology

- $Distribution$: (sample space) the set of all possible samples
- $Dataset$: a set of samples drawn from a distribution
- $Batch$: a subset of samples drawn from the dataset
- $Sample$: a single data object represented as a set of features
- $Feature$: value of a single attribute, property, in a sample. Could be numeric or categorical.

Appendix A: Notations

- x_i: a single feature
- $\boldsymbol{x}_i$: feature vector (a data sample)
- $\boldsymbol{x}_{:,i}$: feature vector of all data samples
- $\boldsymbol{X}$: matrix of feature vectors (dataset)
- $\boldsymbol{W}$: weight matrix
- $\boldsymbol{Z}$: latent representation
- E_θ: encoding function
- G_ϕ: decoding function
- $\widehat{\boldsymbol{X}}$: reconstruction of data
- $\boldsymbol{\Omega}(\boldsymbol{Z})$: sparsity constraint
- $\hat{\rho}_{\boldsymbol{j}}$: average activation of neuron z_{ij}
- $\widetilde{\boldsymbol{X}}$:corrupted input
- N: number of data samples
- P: number of features in a feature vector
- $P^{(k)}$: the number of neurons in layer k
- α: learning rate
- Bold letter/symbol: vector
- Bold capital letters/symbol: matrix