

# Day 4: Unsupervised Learning

Naeemullah Khan  
[naeemullah.khan@kaust.edu.sa](mailto:naeemullah.khan@kaust.edu.sa)



جامعة الملك عبدالله  
للعلوم والتكنولوجيا  
King Abdullah University of  
Science and Technology

KAUST Academy  
King Abdullah University of Science and Technology

December 28, 2025

**By the end of this session, you will be able to:**

- ▶ Differentiate between Supervised and Unsupervised Learning paradigms.
- ▶ Understand K-Means clustering, and how it could be applied to discover hidden groups in data.
- ▶ Analyze high-dimensional data using PCA and t-SNE for visualization and compression.
- ▶ Understand how Autoencoders utilize neural networks for tasks like denoising, generation, and anomaly detection.

# Table of Contents

1. Unsupervised Learning
2. Clustering: K-Means
3. Dimensionality Reduction: PCA
4. Dimensionality Reduction: t-SNE
5. Autoencoders
6. References

# What is Unsupervised Learning?

**So far:** We've been doing **Supervised Learning**, learning from labeled data  $(x, y)$ .

## Supervised Learning

**Data:**  $(x, y)$  pairs

- ▶  $x$ : input (features)
- ▶  $y$ : label (target)

**Goal:** Learn  $f(x) \approx y$

**Examples:**

- ▶ Image classification
- ▶ House price prediction

## Unsupervised Learning

**Data:** Only  $x$  (no labels!)

- ▶ Just raw data
- ▶ No target to predict

**Goal:** Find hidden patterns or structure

**Examples:**

- ▶ Customer segmentation
- ▶ Data compression

**Key difference:** No labels  $\rightarrow$  algorithm must discover structure on its own!

# Why Unsupervised Learning?

## Labels are expensive!

- ▶ Labeling data requires human effort, time, and expertise
- ▶ Most real-world data is unlabeled
- ▶ Sometimes we don't even know what labels to look for

### The Big Idea

Can we learn useful things from data **without** labels?

Yes! Unsupervised learning finds hidden patterns, structure, and representations.

## What can we do without labels?

### 1. Clustering

Group similar data points together

**Example:** Customer segmentation, document group

### 3. Anomaly Detection

Identify unusual or outlier data points

**Example:** Fraud detection, quality control

### 2. Dimensionality Reduction

Compress high-dimensional data to lower dimensions

**Example:** Visualization, feature extraction

### 4. Generation

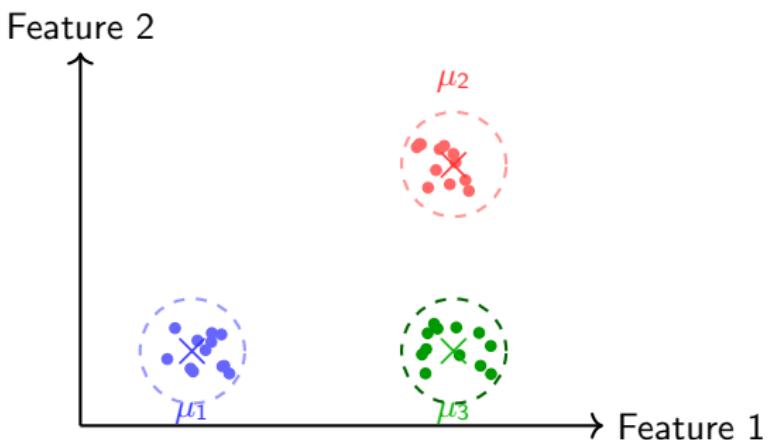
Learn to create new, realistic data samples

**Example:** Image synthesis, data augmentation

*We'll explore four powerful algorithms that tackle these tasks.*

# Clustering: K-Means

**Main Idea:** Represent each cluster by its center (mean)



- ▶ Each cluster has a **center** (centroid)  $\mu_k$  (marked with  $\times$ )
- ▶ Points belong to their **nearest** center
- ▶ Goal: Find centers that minimize total distance to assigned points

## K-Means Algorithm

**Input:** Data  $\{x_1, x_2, \dots, x_n\}$  and number of clusters  $K$

**Initialize:** Randomly place  $K$  cluster centers  $\mu_1, \mu_2, \dots, \mu_K$

**Repeat until convergence:**

1. **Assignment Step:** For each point  $x_i$ , assign to nearest center

$$c_i = \arg \min_{k=1, \dots, K} \|x_i - \mu_k\|^2$$

(find which center is closest)

2. **Update Step:** Move each center to mean of assigned points

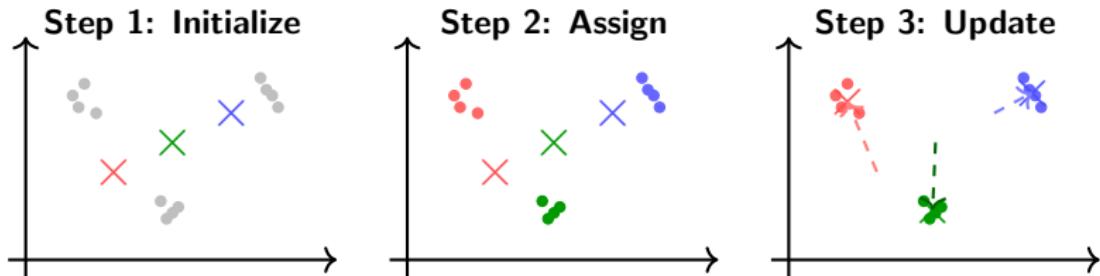
$$\mu_k = \frac{1}{|C_k|} \sum_{i \in C_k} x_i$$

(average all points in cluster  $k$ )

**Stop when:** Assignments don't change OR max iterations reached



# K-Means: Step-by-Step Example



**Repeat steps 2-3** until centers stop moving (typically 10-100 iterations)

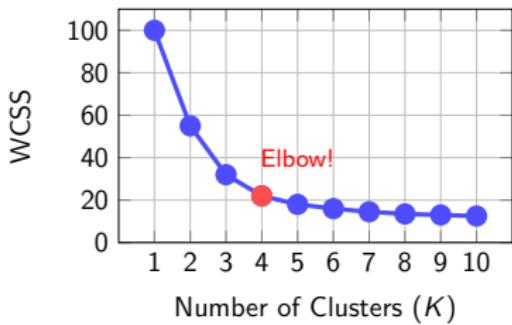
# How to Choose K?

**Problem:** How many clusters should we use?

## Elbow Method

Plot "within-cluster sum of squares" (WCSS) vs  $K$

$$\text{WCSS} = \sum_{k=1}^K \sum_{i \in C_k} \|x_i - \mu_k\|^2$$



Look for the "elbow" where adding more clusters doesn't help much (in the above case it is 4)

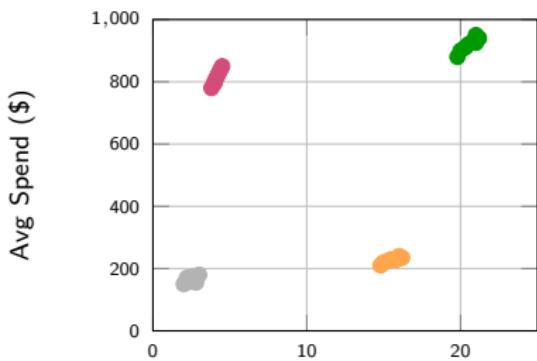
# Application 1: Customer Segmentation

**Use case:** Group customers for targeted marketing

## Discovered Segments:

### Example: E-commerce Data

- ▶ Features: frequency, spend, recency
- ▶ Run K-Means with  $K = 4$
- ▶ Interpret clusters



### Inactive

Low freq,  
low spend

**Action:**  
Re-engage

### Bargain

High freq,  
low spend

**Action:**  
Discounts

### Big Spenders

Low freq,  
high spend

**Action:**  
Premium  
Service

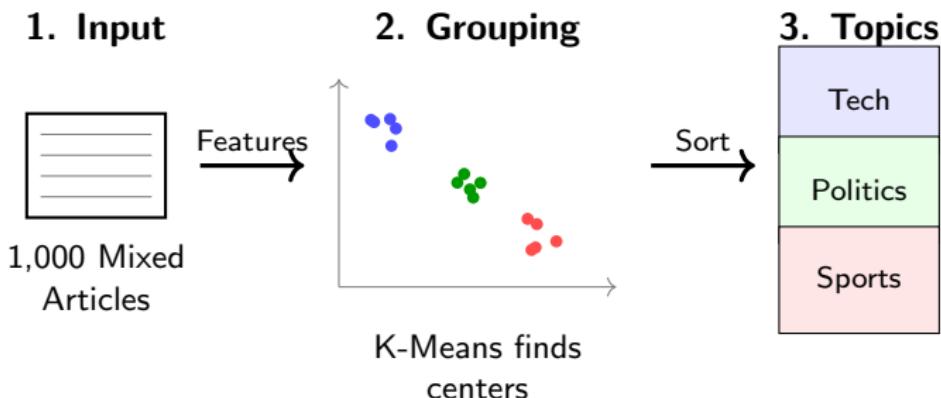
### VIP

High freq,  
high spend

**Action:**  
Loyalty  
Program

# Application 2: Document Clustering

**Use case:** Automatically organize news articles by topic.



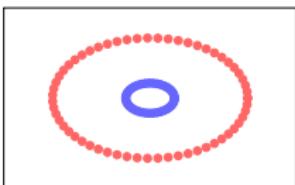
## Process

- ▶ Convert text into numbers (Text Features).
- ▶ Similar stories appear close together in space.
- ▶ K-Means automatically groups them into topics.

*Example: Google News grouping stories from different sources.*

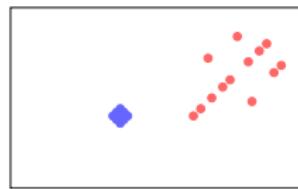
K-Means assumes spherical, equal-sized clusters

## Problem 1: Non-spherical



Fails on concentric circles!

## Problem 2: Different sizes



Struggles with size differences!

## Other Limitations

- ▶ Sensitive to outliers
- ▶ Must specify  $K$  in advance
- ▶ Random initialization matters
- ▶ Bad results for high dimensional data

**Alternatives:** DBSCAN (density-based), Gaussian Mixture Models, Hierarchical clustering

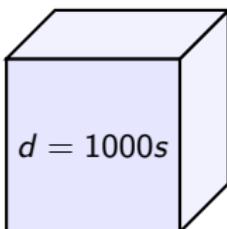
# The Curse of Dimensionality

**Real-world data is high-dimensional:**

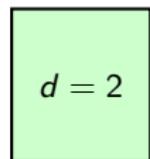
## Examples:

- ▶ Images:  $1000 \times 1000 = 1M$  pixels
- ▶ Text: 10,000+ words
- ▶ Genomics: 20,000+ genes
- ▶ Sensors: 100+ measurements

Many dimensions



PCA  
→



Few dimensions

## Problems:

- ▶ Hard to visualize
- ▶ Slow to compute
- ▶ Lots of redundancy
- ▶ Noisy measurements

**Solution:** Find a lower-dimensional representation while keeping important information  
(Dimensionality Reduction)

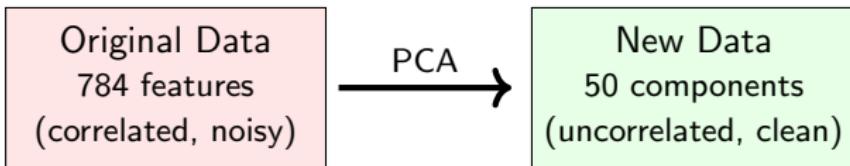
**Principal Component Analysis (PCA)** is a dimensionality reduction algorithm that finds the most important directions in your data.

## Core Idea

Instead of using original features, create **new features** (principal components) that:

- ▶ Capture maximum variance
- ▶ Are uncorrelated with each other
- ▶ Are ranked by importance

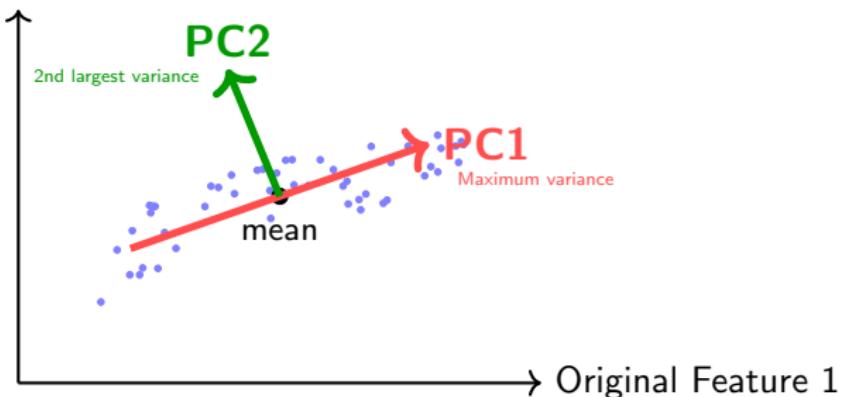
Then keep only the top few components!



# PCA Intuition: Finding the Best Directions

**Goal:** Find directions where data varies the most.

Original Feature 2



- ▶ PC1: Direction with most spread
- ▶ PC2: Perpendicular to PC1, next most spread
- ▶ PC3, PC4, ... : Each perpendicular to all previous ones

# How Does PCA Work? (Part 1)

## Step 1: Prepare the Data

### 1. Center the Data

Shift the data so the center is at the origin (0,0).

$$X_{\text{centered}} = X - \bar{X}$$

*(Subtract the mean from every feature)*

### 2. Compute Covariance Matrix

Calculate how features vary with respect to each other.

$$C = \frac{1}{n} X_{\text{centered}}^T X_{\text{centered}}$$

*(Finds correlations between features)*

## Step 2: Find Directions & Project

### 3. Find Principal Components

Calculate **Eigenvectors** and **Eigenvalues** of the covariance matrix  $C$ .

- ▶ **Eigenvectors:** The directions of the new axes.
- ▶ **Eigenvalues:** The amount of variance (information) along those axes.

### 4. Select & Project

Pick the top  $k$  eigenvectors with the largest eigenvalues and transform the data.

$$Z = X_{\text{centered}} \cdot V_k$$

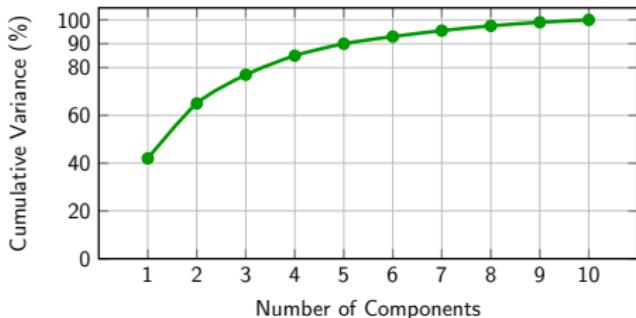
*(Reduces data from  $d$  dimensions  $\rightarrow k$  dimensions)*

# How Many Components to Keep?

**Key question:** How many components are enough?

## Cumulative Variance Explained

(The most common method for selecting  $K$ )



**Rule of Thumb:** Choose the smallest  $K$  that preserves **80–95%** of the total variance.

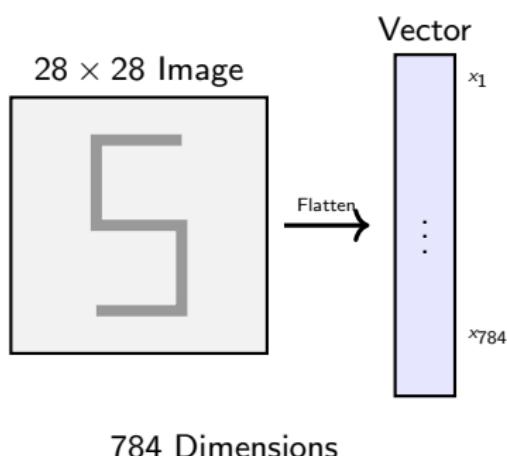
# Application 1: Data Visualization

**The Data:** MNIST Digits, 70,000 handwritten digits (0–9).

## The Challenge:

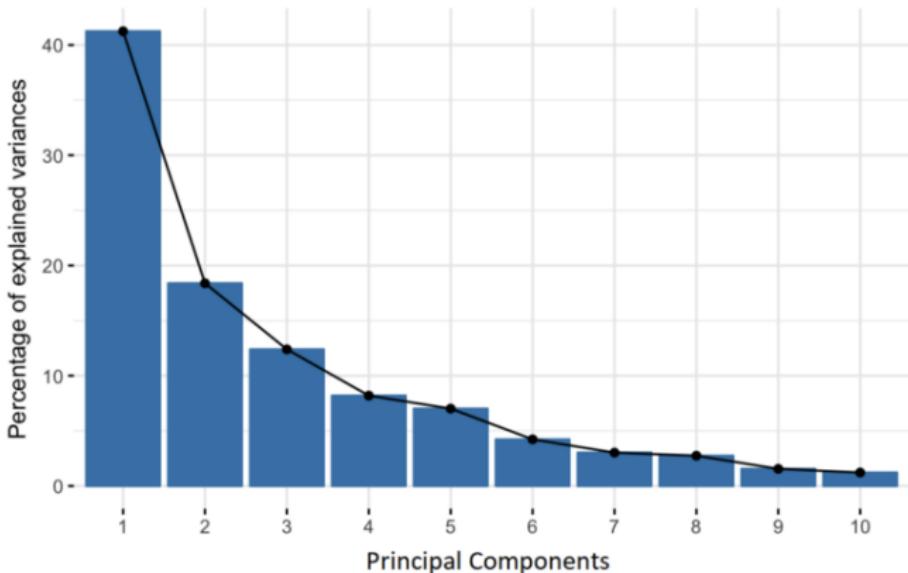
- ▶ Each image is  $28 \times 28$  pixels.
- ▶ Total features:  
 $28 \times 28 = 784$  dimensions!
- ▶ We cannot "see" in 784 dimensions.

**Question:** Can we compress this to just **2D** to see how the digits relate?



# Application 1: Data Visualization

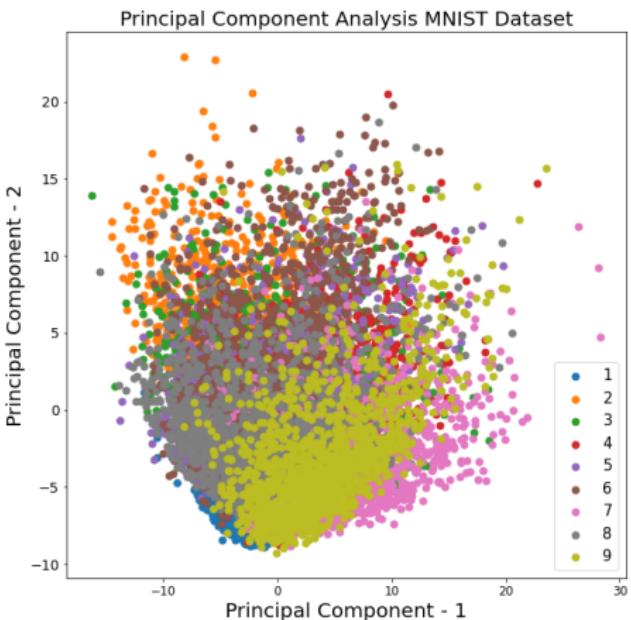
**Applying PCA:** We project the 784 dimensions down to the top 2 Principal Components.



# Application 1: Data Visualization

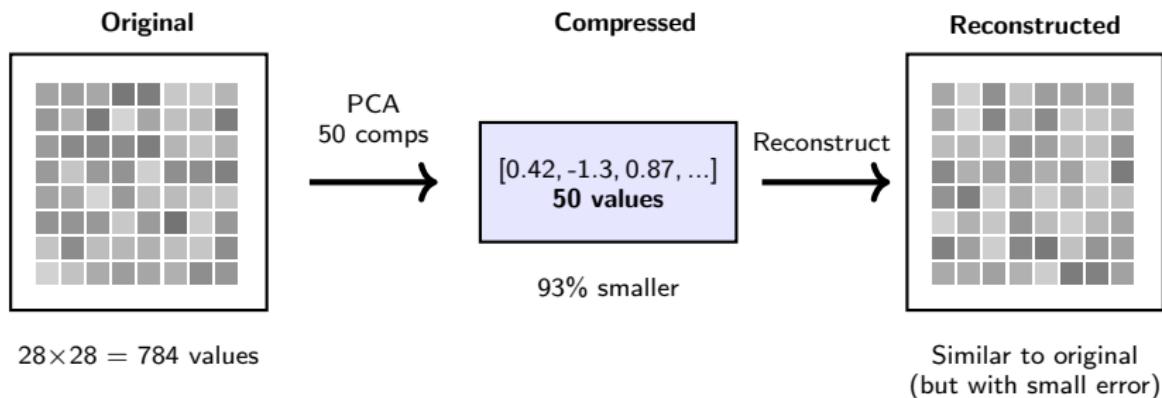
## Is this the best visualization?

- ▶ It captures some structure, but points overlap heavily.
- ▶ **Reason:** PCA is **linear**, it cannot unfold complex, curved structures.
- ▶ PCA is mostly used for compression. We will cover a better visualization algorithm shortly.



# Application 2: Data Compression

**Use case:** Store data more efficiently



**More components = better reconstruction, less compression**

**10 PCs**  
99% compression  
(blurry)

**50 PCs**  
93% compression  
(good)

**200 PCs**  
75% compression  
(near perfect)

## PCA Limitations:

- ▶ **Linear only:** Finds linear combinations of features
- ▶ **Sensitive to scale:** Must standardize data first
- ▶ **Not Interpretable:** PCs are combinations of all features
- ▶ **Outliers:** Can distort principal components

*Let's have a look at a better algorithm for high dimensional data visualization...*

**PCA is great... but it's linear.** Many datasets have **non-linear** structure.

## Problem

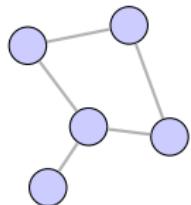
- ▶ High-dimensional data can be hard to visualize
- ▶ PCA may **mix** classes if the separation is non-linear
- ▶ We mainly want a **good 2D plot** for exploration

# t-Distributed Stochastic Neighbor Embedding (t-SNE)

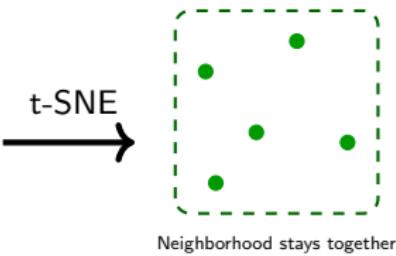
## Main Idea

t-SNE tries to build a 2D map where: neighbors in high-D remain neighbors in 2D.

High-D

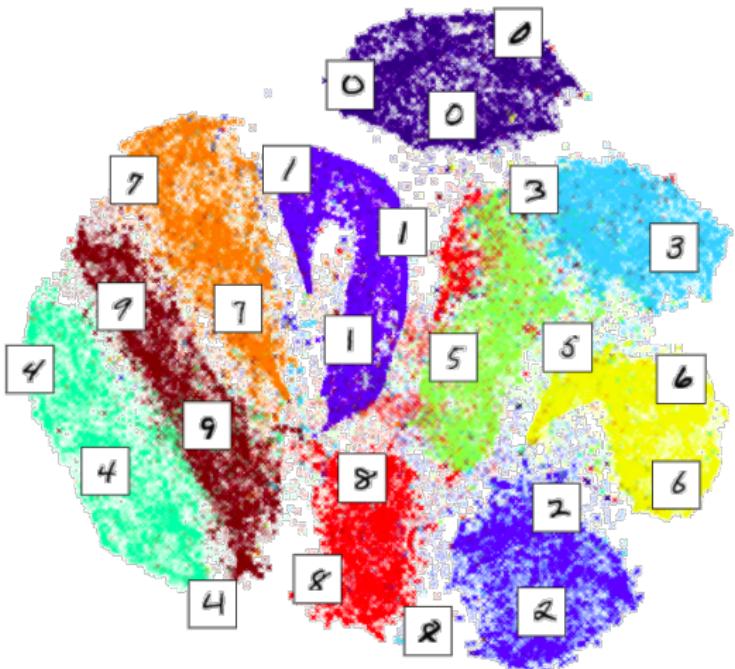


2D map



t-SNE

# Example: Visualizing MNIST via t-SNE



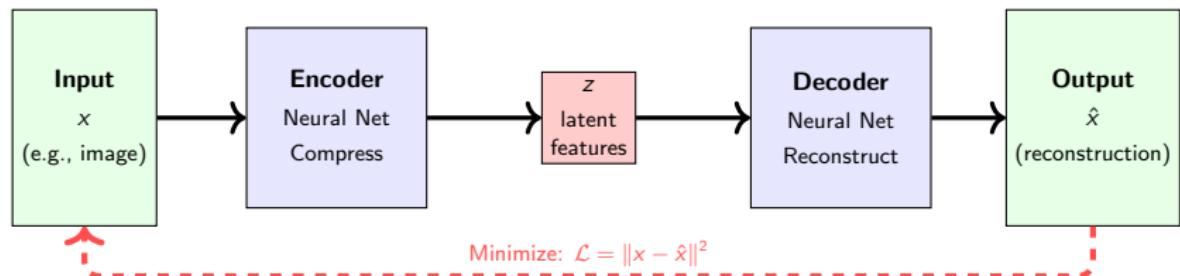
*This is much better than PCA! We can see how similar digits are grouped together, while different ones are far from each other.*

We have covered classical algorithms like K-Means and PCA.

Now, let's explore a powerful Neural Network architecture designed for unsupervised learning...

# What is an Autoencoder?

**Autoencoder:** a type of neural network that learn to compress data into a compact form and then reconstruct it to match the original input.



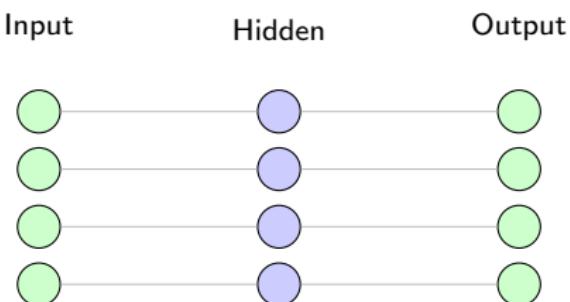
**The trick:** Force information through a narrow **bottleneck**  $z$ .

This forces the network to learn compressed, meaningful representations!

# Why Does the Bottleneck Work?

What if there's no bottleneck?

## Without Bottleneck

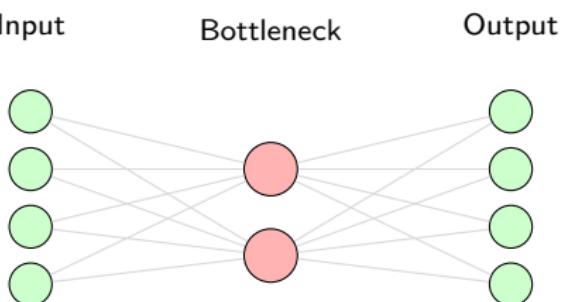


Network can just copy:

$$h_i = x_i, \hat{x}_i = h_i$$

Learns nothing useful!

## With Bottleneck



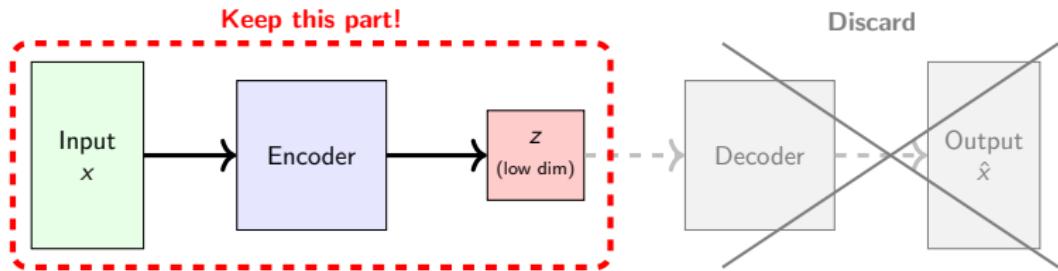
Must compress  $4D \rightarrow 2D$

Forces learning of structure!

Learns meaningful features!

# Application 1: Dimensionality Reduction

How to use an Autoencoder for compression:

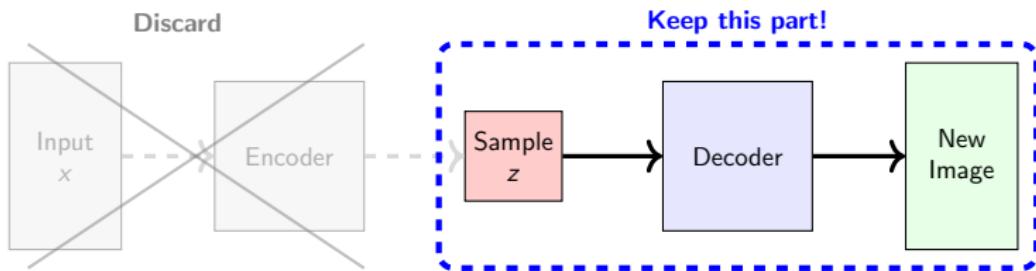


## The Process

1. **Train** the full network to reconstruct the input ( $x \approx \hat{x}$ ).
2. **Discard** the Decoder (it's only needed for training).
3. **Use the Encoder** to map high-dimensional data  $x$  to low-dimensional code  $z$ .

# Application 2: Image Generation

## How to use an Autoencoder for generation:

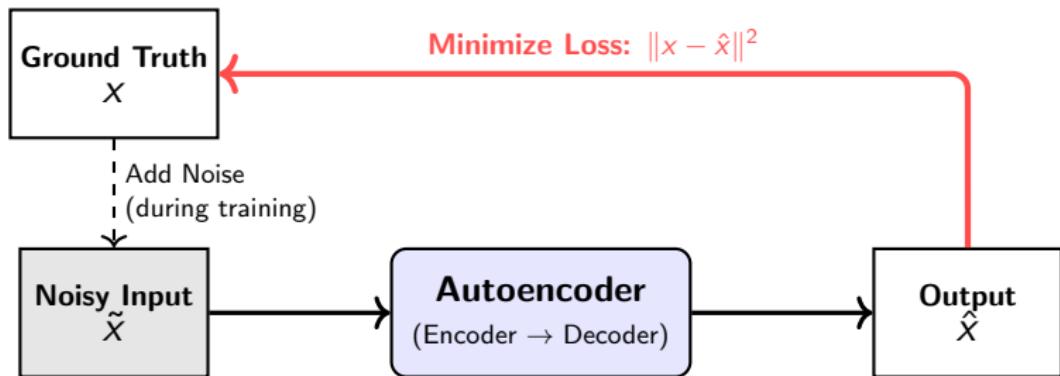


### The Process

1. **Train** the full network on images to learn features.
2. **Discard** the Encoder.
3. **Sample** a random vector  $z$  (pick random numbers).
4. **Generate** a brand new image by running  $z$  through the Decoder.

# Application 3: Denoising Autoencoders

**Idea:** Train the network to recover the original signal from corrupted input.



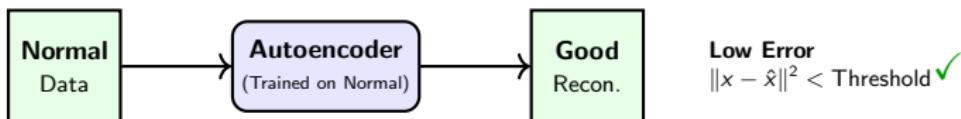
## How it works

- ▶ Take clean image  $x$ .
- ▶ Corrupt it to get  $\tilde{x}$ .
- ▶ Force AE to map  $\tilde{x} \rightarrow x$ .

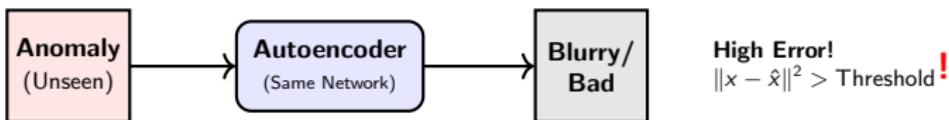
# Application 4: Anomaly Detection

**Concept:** The network learns to reconstruct "Normal" data perfectly.  
When it sees an anomaly, it fails to reconstruct it.

## 1. Training / Normal Operation



## 2. Detection Phase



## Real-World Use Cases

- ▶ **Credit Card Fraud:** Normal transactions reconstruct well, theft looks weird.
- ▶ **Manufacturing:** Detect defective parts on an assembly line.

- ▶ Aurélien Géron, *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*
- ▶ Andrew Ng, *Machine Learning Specialization* (Coursera/DeepLearning.AI)

*Slides contributed by Mohamed Eltayeb*