# CS 486/686
# Unsupervised Learning

Yuntian Deng

Lecture 17

RN 21.7.1 · PM 10.2 · GBC 14.1, 20.10.4

# Learning goals

- Understand what **unsupervised learning** is.
- Trace the **k-means** clustering algorithm.
- Perform **PCA**: mean-center, covariance, eigendecomposition.
- Understand the basic idea of **autoencoders** and **GANs**.

# Two big unsupervised tasks

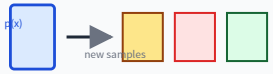
No labels — only the raw data $\{x_i\}$. What can we learn?

## Representation learning


Find a low-dimensional summary of each example.

Clustering, PCA, autoencoders.

## Generative modelling




Learn the data distribution so we can *sample new examples*.

GANs, VAEs, diffusion.

Today: one task from each, plus two neural variants.

# Clustering: hard vs soft

Group training examples into **clusters**, treating them like discovered categories.

### Hard clustering

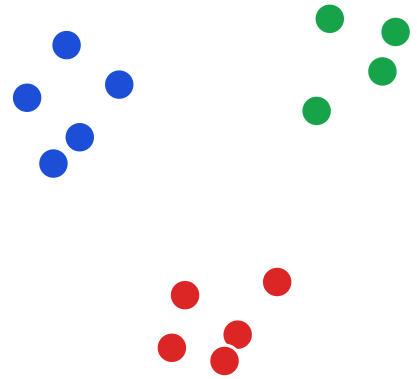
Each $x$ gets *exactly one* cluster.

$$\text{class}(x) = c$$

### Soft clustering

Each $x$ gets a *distribution* over clusters.

$$\text{class}(x) \sim P(C \mid x)$$



Each color is a discovered cluster — no labels were given.

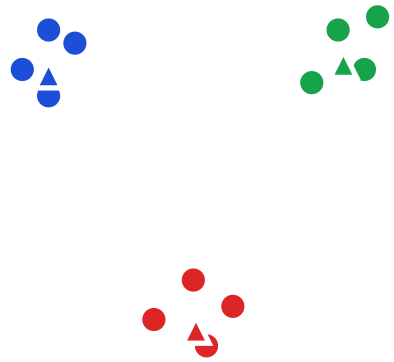
# k-Means: centroids and distance

**Inputs:** number of clusters $k$ and $m$ training examples $x_i \in \mathbb{R}^n$. Each cluster has a **centroid** $c \in \mathbb{R}^n$; each $x_i$ is assigned to its closest centroid.

### Euclidean distance (L2)

$$d(c, x) = \sqrt{\sum_{j=1}^{n} (c_j - x_j)^2}$$

- **Hard clustering**: each example goes to one centroid.
- Centroids live in the same feature space as the data.



Triangles = current centroids; circles = data points.

# The k-means algorithm

Alternate two steps until convergence:

**k-means($X$, $k$)**

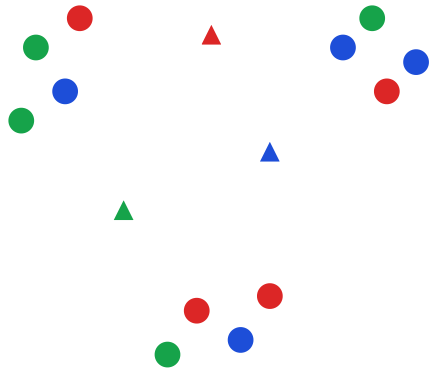
1. Initialize $k$ centroids $C[0 \ldots k-1]$ randomly.
2. Repeat until assignments stop changing:
   - **Assign step:** for each example $i$, $y[i] \leftarrow \arg\min_c d\big(C[c], X[i]\big)$.
   - **Update step:** for each cluster $c$, $C[c] \leftarrow$ mean of all $X[i]$ with $y[i] = c$.

- Guaranteed to converge with L2 distance.
- Not guaranteed to find the global optimum.
- **Mitigate:** random restarts, feature scaling.

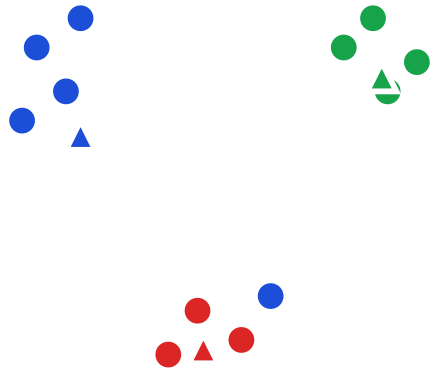
# k-Means in action

Same 12 points, three snapshots: random init → mid-iteration → converged.

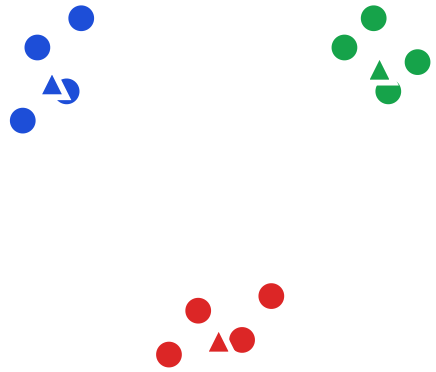
**Iteration 0 (random init)**



**Iteration 2 (mid)**



**Converged**



Centroids drift; assignments stabilize; cost decreases monotonically.

# One iteration: a numeric walk-through

$k = 2$, Euclidean (squared) distance. Data + initial centroids:

| example | $x_1$ | $x_2$ | $x_3$ |
|---|---|---|---|
| 1 | 0.2 | 0.5 | 0 |
| 2 | −0.6 | 2.1 | 1.2 |
| 3 | −0.5 | 1.9 | 1.3 |
| 4 | 0.1 | 0.5 | −0.3 |

$c_1 = (0.3, 0.8, -0.5), \quad c_2 = (-0.1, -0.5, 1.0).$

**Assign step** (squared distance to each centroid):

```
ex 1: d²(c₁) = 0.35, d²(c₂) = 2.09 → c₁
ex 2: d²(c₁) = 5.39, d²(c₂) = 7.05 → c₁
ex 3: d²(c₁) = 5.09, d²(c₂) = 6.01 → c₁
ex 4: d²(c₁) = 0.17, d²(c₂) = 2.73 → c₁
```

**Update step**:

$$c_1' = (-0.2, 1.25, 0.55), \quad c_2' = \varnothing$$

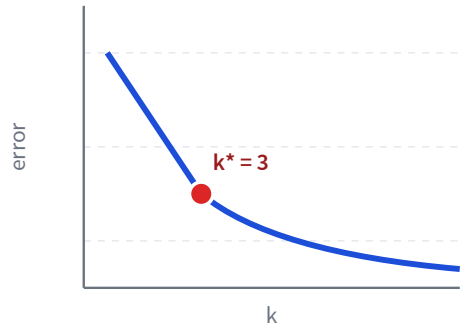
**Watch out:** $c_2$ got no examples. Re-initialize it (random restart).

# How do we choose $k$?

Larger $k$ always gives lower training error — but defeats the point of compression.

## Elbow method

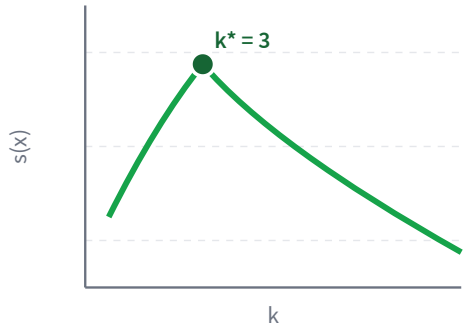
1. Run k-means for $k \in \{1, \ldots, k_{\max}\}$.
2. Plot average within-cluster distance.
3. Pick the "elbow" where error stops dropping fast.




## Silhouette score

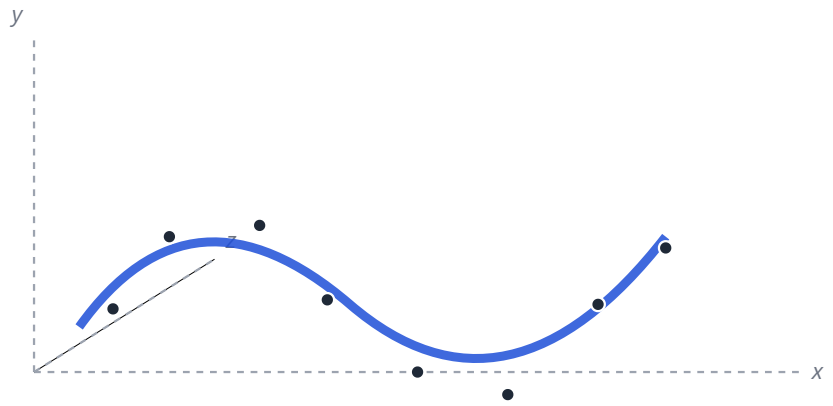
$$s(x) = \frac{b(x) - a(x)}{\max(a(x),\, b(x))}$$

- $a(x)$ = mean distance to own cluster.
- $b(x)$ = mean distance to *nearest other* cluster.
- Pick $k$ maximizing average $s$.



# Dimension reduction: the manifold idea

High-dimensional data often lives on a **low-dimensional manifold**. Extra dimensions are mostly noise.

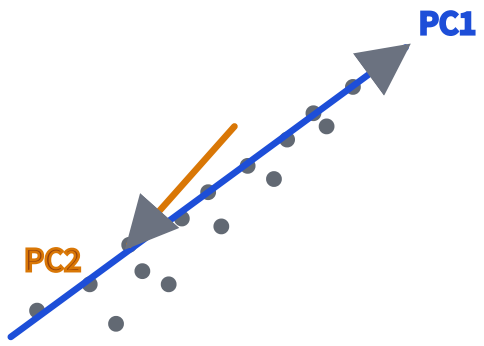



Goal: recover the intrinsic 1D coordinate along the blue ribbon.

# Principal Component Analysis (PCA)

### The PCA recipe

Find **orthogonal axes** that successively **maximize the variance** of the projected data.




- First PC = direction of largest variance.
- Second PC = direction orthogonal to first that maximizes remaining variance.
- Project onto the top $k$ PCs for a compact representation.

# How to compute PCA

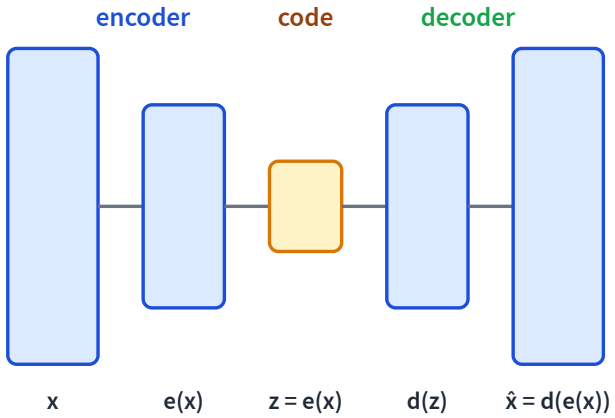
**PCA($X$, $k$)**

1. Mean-center the data: $x \leftarrow x - \mathrm{mean}(x)$.
2. Compute the covariance matrix $\Sigma = \frac{1}{m} X^\top X$.
3. Eigendecompose $\Sigma$: $\Sigma\, v_k = \lambda_k\, v_k$. Eigenvectors $v_k$ are the principal components, sorted by $\lambda_1 \geq \lambda_2 \geq \ldots$
4. Project onto the top $k$ PCs: $z = V_{\text{top-}k}^\top\, x$.

Fraction of variance kept by the $k$-th PC: $\dfrac{\lambda_k}{\sum_j \lambda_j}$. Cumulative across top $k$ PCs guides how many to keep.

# Autoencoders: encoder + decoder

A neural net learns to **squeeze and reconstruct** its input.




**Loss:**

$$E = \sum_i (x_i - d(e(x_i)))^2$$

Encoder + decoder are trained jointly so that the bottleneck z keeps enough info to rebuild x.

# Linear vs deep autoencoders

## Linear autoencoder

Single linear layer; weights shared between encoder and decoder.

$$\hat{z} = Wx, \quad \hat{x} = W^\top \hat{z}$$

With squared loss, the solution recovers the same subspace as PCA.

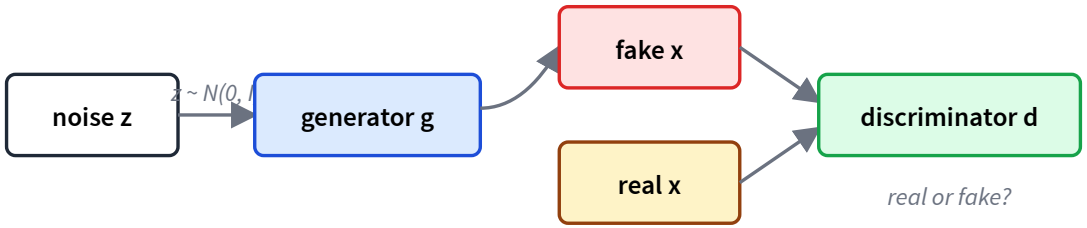
## Deep neural autoencoder

Stack non-linear layers (ReLU, etc.) in both encoder and decoder; train end-to-end with backprop.

Captures non-linear manifolds — the modern workhorse for representation learning.

# Generative Adversarial Networks (GANs)

Two networks play a game: a **generator** tries to fool a **discriminator**.




- **Generator** $g(z)$: turns Gaussian noise $z$ into a synthetic example that should look real.
- **Discriminator** $d(x)$: binary classifier — was $x$ drawn from training data, or from $g$?

# GAN training: a minimax game

Both networks optimize the same value, in opposite directions:

$$E = \mathbb{E}_{x \sim p_{\text{data}}}\big[\log d(x)\big] + \mathbb{E}_{z \sim p_z}\big[\log(1 - d(g(z)))\big]$$

### Discriminator

Maximize $E$: push $d(\text{real}) \to 1$ and $d(\text{fake}) \to 0$.

### Generator

Minimize $E$: generate fakes that *fool* the discriminator.

At equilibrium: $g$ produces realistic samples and $d(x) \approx \frac{1}{2}$ — maximally uncertain.

# Learning goals (recap) — Next: decision trees

✓ Understand what **unsupervised learning** is.

✓ Trace the **k-means** clustering algorithm.

✓ Perform **PCA** by mean-centering, covariance, and eigendecomposition.

✓ Understand the basic idea of **autoencoders** and **GANs**.

L18: back to supervised — the **decision tree** learner.