

Lecture 18:

# Advanced Unsupervised Learning

**Ho-min Park**

[homin.park@ghent.ac.kr](mailto:homin.park@ghent.ac.kr)

[powersimmani@gmail.com](mailto:powersimmani@gmail.com)

# Lecture Contents

**Part 1:** Self-Supervised Learning

**Part 2:** Time Series Clustering

**Part 3:** Graph Clustering

**Part 4:** Advanced Topics and Applications

**Part 1/4:**

# **Self-Supervised Learning**

- 1.** Lecture Overview and Recap
- 2.** What is Self-Supervised Learning?
- 3.** Contrastive Learning Principles
- 4.** SimCLR Algorithm
- 5.** MoCo and BYOL
- 6.** Self-Supervised Pretraining
- 7.** Hands-on: Image Representation Learning

## Supervised Learning



**Data:** Labeled datasets

**Requirement:** Manual annotation

**Cost:** High (time + labor)

## Unsupervised Learning



**Data:** Unlabeled data

**Requirement:** No labels needed

**Goal:** Find patterns

## Self-Supervised Learning

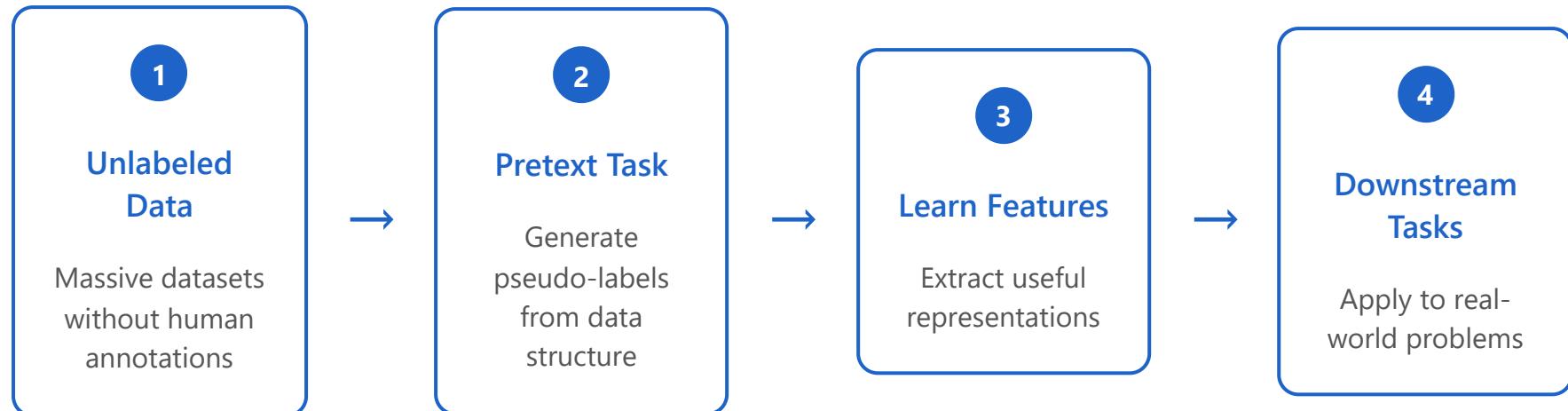


**Data:** Unlabeled → Auto-labeled

**Requirement:** Labels from data itself

**Advantage:** Leverage abundant data

*Traditional → Pattern Discovery → Modern Bridge Approach*



### Pretext Task Examples



#### Image Rotation

Predict rotation angle ( $0^\circ, 90^\circ, 180^\circ, 270^\circ$ )



#### Colorization

Predict colors from grayscale images



#### BERT (NLP)

Masked language modeling



#### GPT (NLP)

Next token prediction

✓ No human annotation required • Scales to massive datasets



**Core Idea:** Similar items close, dissimilar items far apart



**Positive Pairs:** Augmented versions of same data



**Negative Pairs:** Different data points



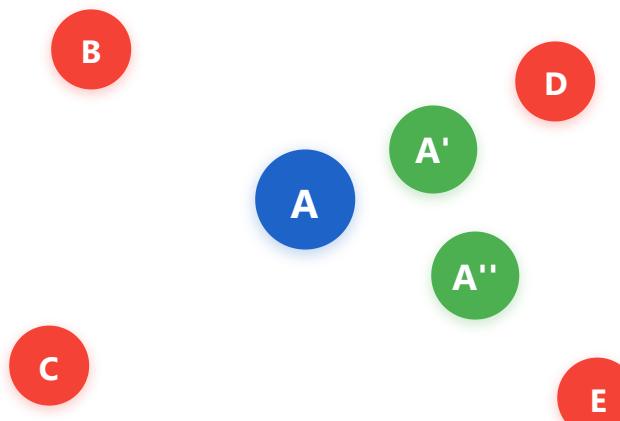
**InfoNCE Loss:** Maximize agreement of positive pairs



**Temperature ( $\tau$ ):** Controls distribution sharpness

Pull positives close ← → Push negatives away

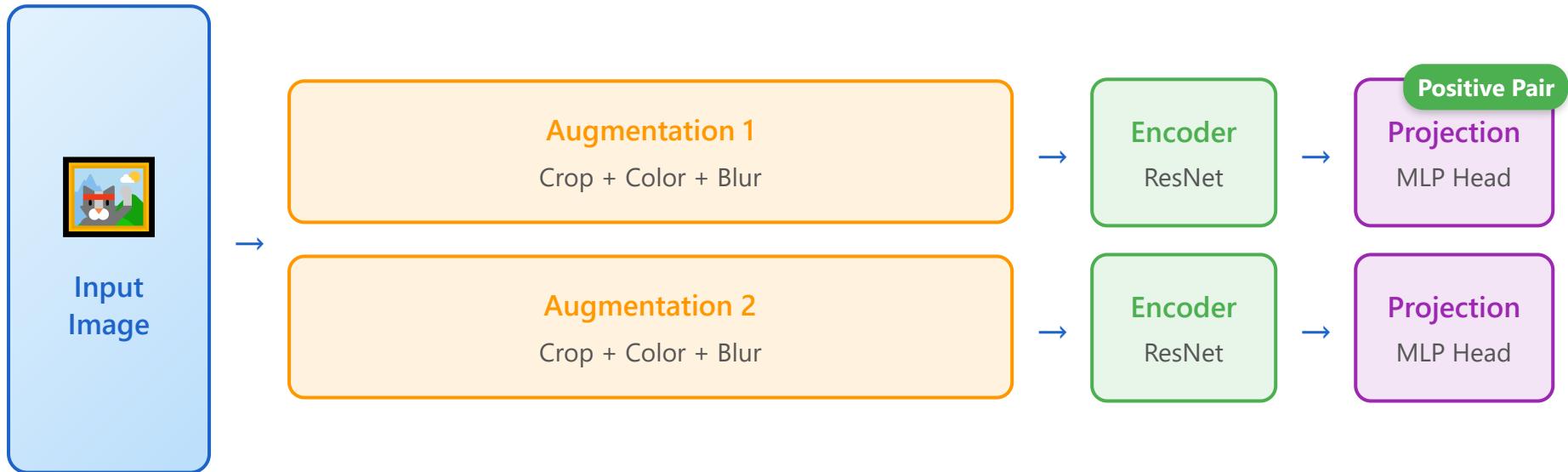
### Embedding Space Visualization



● Anchor ● Positive ● Negative

## SimCLR: Simple Framework for Contrastive Learning

Self-Supervised Visual Representation Learning



**NT-Xent Loss**

Maximize agreement between positive pairs with temperature scaling ( $\tau$ )

🎯 Large batch size: 4096-8192

🔁 Strong data augmentation

🏆 SOTA on ImageNet (no labels)



# Advanced Self-Supervised Methods

## Architecture Comparison: MoCo vs BYOL



### MoCo

Momentum Contrast

Query Encoder



Query

Momentum  
Encoder



Key

Queue of Negative Samples



**Queue-based:** Maintains large dictionary of negatives



**Momentum update:** Consistent representations over time



**Contrastive:** Uses negative pairs

✓ Excellent ImageNet Performance



### BYOL

Bootstrap Your Own Latent

Online Network



Predictor

Target Network



Projection

Stop-Gradient



**No negatives:** Only positive pairs needed



**Predictor network:** Asymmetric architecture



**Stop-gradient:** Prevents collapse

✓ Excellent ImageNet Performance



Vector Calculation Examples



## MoCo: InfoNCE Loss Calculation

### 1 Define Input Vectors

Query encoder produces query vector  $\mathbf{q}$ , momentum encoder produces positive key  $\mathbf{k}^+$  and negative keys from queue.

$\mathbf{q}$  (query) = [0.8, 0.4, 0.2, 0.1] → normalized: [0.857, 0.429, 0.214, 0.107]

$\mathbf{k}^+$  (positive key) = [0.7, 0.5, 0.3, 0.1] → normalized: [0.761, 0.543, 0.326, 0.109]

$\mathbf{k}_1^-$  (negative key 1) = [-0.3, 0.6, -0.4, 0.5] → normalized: [-0.324, 0.649, -0.432, 0.541]

$\mathbf{k}_2^-$  (negative key 2) = [0.1, -0.7, 0.5, 0.3] → normalized: [0.110, -0.769, 0.549, 0.330]

### 2 Calculate Similarity Scores (Dot Products)

$$\text{similarity}(\mathbf{a}, \mathbf{b}) = \mathbf{a} \cdot \mathbf{b} = \sum(a_i \times b_i)$$

$$\begin{aligned}\mathbf{q} \cdot \mathbf{k}^+ &= (0.857 \times 0.761) + (0.429 \times 0.543) + (0.214 \times 0.326) + (0.107 \times 0.109) \\ &= 0.652 + 0.233 + 0.070 + 0.012 = \mathbf{0.967}\end{aligned}$$

$$\begin{aligned}\mathbf{q} \cdot \mathbf{k}_1^- &= (0.857 \times -0.324) + (0.429 \times 0.649) + (0.214 \times -0.432) + (0.107 \times 0.541) \\ &= -0.278 + 0.278 + (-0.092) + 0.058 = \mathbf{-0.034}\end{aligned}$$

$$\begin{aligned}\mathbf{q} \cdot \mathbf{k}_2^- &= (0.857 \times 0.110) + (0.429 \times -0.769) + (0.214 \times 0.549) + (0.107 \times 0.330) \\ &= 0.094 + (-0.330) + 0.117 + 0.035 = \mathbf{-0.084}\end{aligned}$$

**Positive Pair: 0.967**

High similarity ✓

**Negative Pairs: -0.034, -0.084**

Low similarity ✓

### 3 Apply Temperature Scaling ( $\tau = 0.07$ )

$$\text{scaled\_sim} = \text{similarity} / \tau$$

$$q \cdot k^+ / \tau = 0.967 / 0.07 = 13.81$$

$$q \cdot k_1^- / \tau = -0.034 / 0.07 = -0.49$$

$$q \cdot k_2^- / \tau = -0.084 / 0.07 = -1.20$$

Temperature  $\tau$  sharpens the distribution - lower  $\tau$  makes the model more confident.

### 4 Compute InfoNCE Loss

$$\mathcal{L} = -\log( \exp(q \cdot k^+ / \tau) / [ \exp(q \cdot k^+ / \tau) + \sum \exp(q \cdot k_i^- / \tau) ] )$$

$$\exp(13.81) = 999,148.5$$

$$\exp(-0.49) = 0.613$$

$$\exp(-1.20) = 0.301$$

$$\text{Denominator} = 999,148.5 + 0.613 + 0.301 = 999,149.4$$

$$\text{Softmax probability} = 999,148.5 / 999,149.4 = 0.999999$$

InfoNCE Loss

$$\mathcal{L} = -\log(0.999999) \approx 0.000001$$

Low loss indicates the model successfully distinguishes the positive pair from negatives. The goal is to minimize this loss during training.



## BYOL: Mean Squared Error Loss Calculation

### 1 Define Input Vectors

Online network produces prediction  $\mathbf{p}$ , target network (with stop-gradient) produces projection  $\mathbf{z}$ .

$$\mathbf{p} \text{ (online prediction)} = [0.6, 0.3, 0.5, 0.2]$$

$$\mathbf{z} \text{ (target projection)} = [0.7, 0.4, 0.4, 0.3]$$

### 2 L2 Normalize Both Vectors

$$\bar{\mathbf{x}} = \mathbf{x} / \|\mathbf{x}\|_2 \quad \text{where} \quad \|\mathbf{x}\|_2 = \sqrt{(\sum x_i^2)}$$

$$\|\mathbf{p}\|_2 = \sqrt{(0.6^2 + 0.3^2 + 0.5^2 + 0.2^2)} = \sqrt{(0.36 + 0.09 + 0.25 + 0.04)} = \sqrt{0.74} = 0.860$$

$$\begin{aligned}\bar{\mathbf{p}} &= [0.6/0.860, 0.3/0.860, 0.5/0.860, 0.2/0.860] \\ &= [0.698, 0.349, 0.581, 0.233]\end{aligned}$$

$$\|\mathbf{z}\|_2 = \sqrt{(0.7^2 + 0.4^2 + 0.4^2 + 0.3^2)} = \sqrt{(0.49 + 0.16 + 0.16 + 0.09)} = \sqrt{0.90} = 0.949$$

$$\begin{aligned}\bar{\mathbf{z}} &= [0.7/0.949, 0.4/0.949, 0.4/0.949, 0.3/0.949] \\ &= [0.738, 0.422, 0.422, 0.316]\end{aligned}$$

### 3 Calculate MSE Between Normalized Vectors

$$\mathcal{L} = \|\bar{\mathbf{p}} - \bar{\mathbf{z}}\|^2 = \sum (\bar{p}_i - \bar{z}_i)^2$$

Differences:

$$\begin{aligned}\bar{p}_1 - \bar{z}_1 &= 0.698 - 0.738 = -0.040 \\ \bar{p}_2 - \bar{z}_2 &= 0.349 - 0.422 = -0.073 \\ \bar{p}_3 - \bar{z}_3 &= 0.581 - 0.422 = +0.159 \\ \bar{p}_4 - \bar{z}_4 &= 0.233 - 0.316 = -0.083\end{aligned}$$

Squared differences:

$$\begin{aligned}(-0.040)^2 &= 0.0016 \\ (-0.073)^2 &= 0.0053 \\ (+0.159)^2 &= 0.0253 \\ (-0.083)^2 &= 0.0069\end{aligned}$$

BYOL Loss (MSE)

$$\mathcal{L} = 0.0016 + 0.0053 + 0.0253 + 0.0069 = 0.0391$$

#### 4 Alternative Formula (Cosine Similarity)

$$\mathcal{L} = 2 - 2 \times (\bar{p} \cdot \bar{z}) = 2 - 2 \times \cos(\theta)$$

$$\begin{aligned}\bar{p} \cdot \bar{z} &= (0.698 \times 0.738) + (0.349 \times 0.422) + (0.581 \times 0.422) + (0.233 \times 0.316) \\ &= 0.515 + 0.147 + 0.245 + 0.074 = 0.981\end{aligned}$$

BYOL Loss (Cosine Form)

$$\mathcal{L} = 2 - 2 \times 0.981 = 2 - 1.962 = 0.038$$

Both formulas are equivalent for L2-normalized vectors. The loss approaches 0 when vectors are identical (cosine = 1) and 4 when completely opposite (cosine = -1).

#### 5 Symmetrized Loss (Full BYOL)

$$\mathcal{L}_{\text{total}} = \mathcal{L}(p, z') + \mathcal{L}(p', z)$$

In practice, BYOL uses two augmented views and computes loss in both directions, then averages them. This symmetrization helps learn more robust representations.

If  $\mathcal{L}(p, z') = 0.038$  and  $\mathcal{L}(p', z) = 0.042$

$\mathcal{L}_{\text{total}} = 0.038 + 0.042 = 0.080$



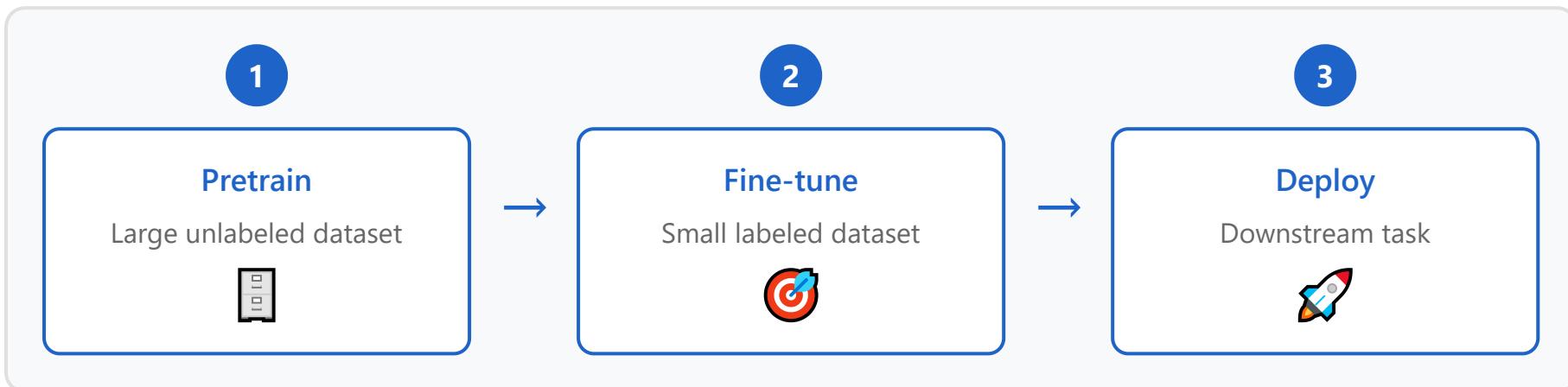
## Key Differences in Loss Computation

### MoCo

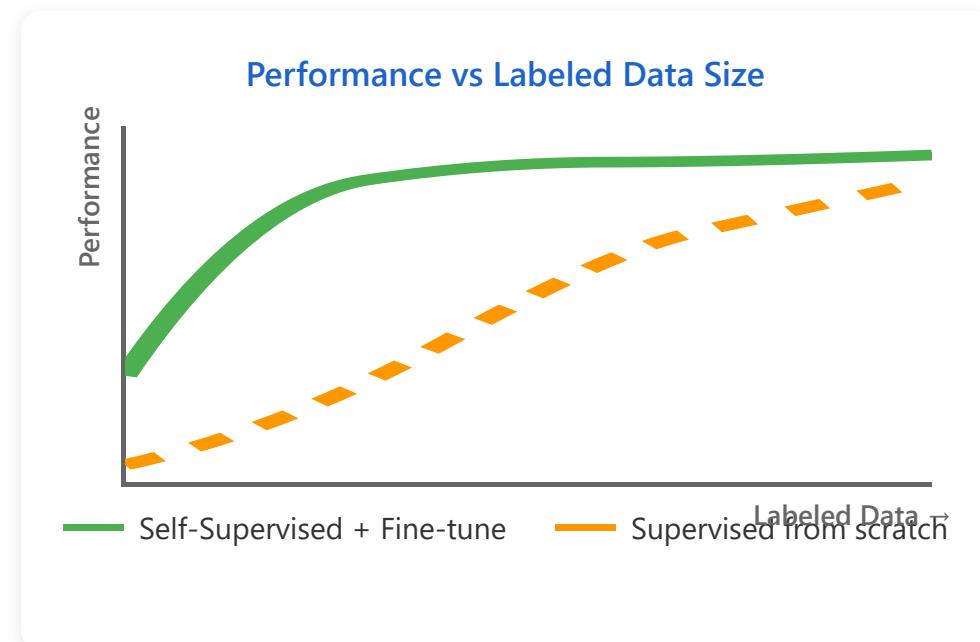
- **Contrastive:** Requires negative samples
- **InfoNCE:** Softmax over all keys
- **Queue:** 65,536 negatives typically
- **Goal:** Maximize pos/neg ratio

### BYOL

- **Non-contrastive:** No negatives needed
- **MSE:** Direct regression loss
- **Predictor:** Prevents collapse
- **Goal:** Match target representation



- 🌐 Cross-domain transfer learning
- 📈 Reduces labeled data needs
- 🏥 Medical imaging applications
- 🛰️ Satellite imagery analysis
- ✓ Match pretraining & target domains



Text

Images

Audio

Video

# Hands-on: Image Representation Learning with SimCLR

## PyTorch Implementation Workflow

### 1 Dataset Setup

Load unlabeled dataset for pretraining



CIFAR-10 / STL-10

### 2 Implement SimCLR

Build encoder & projection head

PyTorch



### 3 Data Augmentation

Setup augmentation pipeline

Crop + Color + Blur



### 4 Train Encoder

Optimize with contrastive loss

NT-Xent Loss



### 5 Linear Probe

Evaluate on labeled subset

Frozen Features



### 6 Visualize Embeddings

Plot learned representations

t-SNE



**Final Step:** Compare self-supervised results with supervised baseline performance



Example Data and Calculation Process



## 1. Input Images & Augmentation

Image	Size	Augmentation
Cat (Original)	32×32×3	-
Cat View 1	32×32×3	Random Crop + Flip
Cat View 2	32×32×3	Color Jitter + Blur

**Key Point:** Generate two different views from the same image to form a positive pair



## 2. Encoder & Projection Head

### Architecture:

Input (3×32×32) → ResNet-18 → h (512-dim) → MLP → z (128-dim)

# Example embedding vectors

$z_1 = [0.23, -0.45, 0.67, \dots, 0.12]$  # 128-dim

$z_2 = [0.21, -0.43, 0.65, \dots, 0.15]$  # 128-dim (positive)

$z_3 = [-0.89, 0.34, -0.12, \dots, 0.56]$  # 128-dim (negative)

**Key Point:** Calculate cosine similarity after L2 normalization



## 3. NT-Xent Loss Calculation (Batch Size N=4, Total 8 Samples)

### NT-Xent Loss Formula:

$$\ell(i, j) = -\log[\exp(\text{sim}(z_i, z_j)/\tau) / \sum \exp(\text{sim}(z_i, z_k)/\tau)]$$

$\tau$  (temperature) = 0.5

### Step 1: Calculate Similarity Matrix (8×8)

```
sim(z1, z2) = 0.92 // positive pair  
sim(z1, z3) = 0.15 // negative  
sim(z1, z4) = -0.23 // negative  
sim(z1, z5) = 0.08 // negative  
...  
...
```

## Step 2: Temperature Scaling ( $\tau=0.5$ )

```
sim(z1, z2) / τ = 0.92/0.5 = 1.84  
sim(z1, z3) / τ = 0.15/0.5 = 0.30  
sim(z1, z4) / τ = -0.23/0.5 = -0.46  
...
```

## Step 3: Exponential Calculation

```
exp(1.84) = 6.297 // positive (numerator)  
exp(0.30) = 1.350 // negative  
exp(-0.46) = 0.631 // negative  
exp(0.16) = 1.174 // negative  
...  
Denominator Sum = 6.297 + 1.350 + 0.631 + 1.174 + ... = 12.845
```

## Step 4: Loss Calculation

```
l(z1, z2) = -log(6.297 / 12.845)  
l(z1, z2) = -log(0.490)  
l(z1, z2) = 0.713
```

Final Batch Loss (Average)

**Loss = 0.713**

 **Interpretation:** Lower loss indicates that the model has learned good representations where positive pairs are close and negative pairs are far apart.

## Training Results Example

## ⚡ Self-Supervised Pretraining

Epoch	Loss	Time
1	2.456	8 min
50	1.234	-
100	0.876	-
200	0.623	26 hours

**Settings:** CIFAR-10 unlabeled, batch=256, ResNet-18

## 📊 Linear Probe Evaluation (CIFAR-10 Test)

Method	Accuracy	Training Data
Random Init	23.4%	5,000 labels
SimCLR (ours)	78.6%	5,000 labels
Supervised	91.2%	50,000 labels

Self-Supervised Improvement

+55.2%

## 💻 PyTorch Implementation (NT-Xent Loss)

```
# SimCLR NT-Xent Loss Implementation
def nt_xent_loss(z_i, z_j, temperature=0.5):
    """
    Args:
        z_i, z_j: [batch_size, embedding_dim] - positive pair
    Returns:
        loss: scalar tensor
    """
    batch_size = z_i.shape[0]

    # L2 normalization
    z_i = F.normalize(z_i, dim=1)
    z_j = F.normalize(z_j, dim=1)
```

```
# Concatenate: [2*batch_size, embedding_dim]
z = torch.cat([z_i, z_j], dim=0)

# Similarity matrix: [2N, 2N]
similarity_matrix = torch.mm(z, z.T) / temperature

# Mask to exclude self-similarity
mask = torch.eye(2 * batch_size, dtype=torch.bool)
similarity_matrix = similarity_matrix.masked_fill(mask, -9e15)

# Positive pairs: (i, N+i) and (N+i, i)
positives = torch.cat([
    torch.diag(similarity_matrix, batch_size),
    torch.diag(similarity_matrix, -batch_size)
])

# NT-Xent loss calculation
nominator = torch.exp(positives)
denominator = torch.sum(torch.exp(similarity_matrix), dim=1)
loss = -torch.log(nominator / denominator)

return loss.mean()
```



**Key Insight:** Self-supervised learning can learn powerful feature representations without labels, achieving high performance with only a small amount of labeled data.

**Part 2/4:**

# **Time Series Clustering**

- 8.** Characteristics of Time Series Data
- 9.** DTW (Dynamic Time Warping) Principles
- 10.** DTW-based Clustering
- 11.** K-Shape Algorithm
- 12.** Subsequence Clustering
- 13.** Hands-on: Stock/Sensor Data Clustering



## Time Series Data

Sequential data with temporal dependencies and ordering

## Key Challenges

**Variable lengths:** Different time durations

**Sampling rates:** Inconsistent frequencies

**Phase shifts:** Time warping between patterns

**Noise:** Random fluctuations & outliers

**Euclidean distance:** Inadequate for alignment

## Time Series Characteristics

### Trend & Seasonality



### Phase Shift (Time Warping)



### Noisy Signal



### Different Lengths



## Key Considerations

 Finance

 IoT Sensors

Seasonality

Trends

Outliers

 Healthcare

Alignment-invariant metrics

## Dynamic Time Warping

DTW Algorithm



Warps time axis to align sequences



Finds optimal alignment path



Allows one-to-many matching



DTW distance = sum of aligned distances



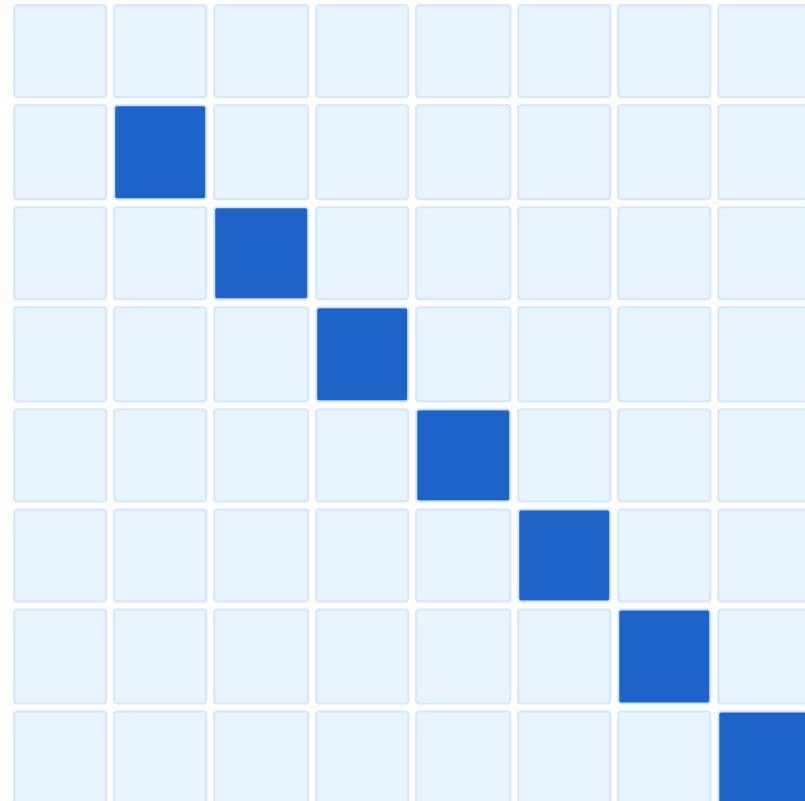
Handles speed variations & phase shifts

Time Complexity

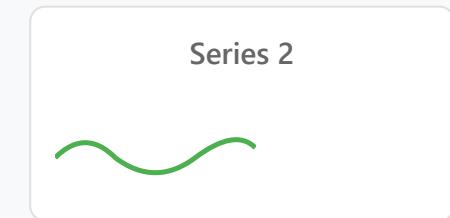
$O(n^2)$

## DTW Alignment Matrix & Warping Path

Series 1



Series 2



Alignment Constraints

Sakoe-Chiba Band

Itakura Parallelogram

# DTW Calculation Example

Step-by-Step Computation

Sequence X

1    3    4    5

Sequence Y

2    3    5

📊 DTW Matrix (Cumulative Cost)

	0	2	3	5
0	$ 1-2 $ 1	$\infty$	$\infty$	$\infty$
1	$ 3-2 $ 2	$ 3-3 $ 2	$\infty$	$\infty$
3	$ 4-2 $ 4	$ 4-3 $ 3	$ 4-5 $ 4	$\infty$
4	$ 5-2 $ 7	$ 5-3 $ 5	$ 5-5 $ 4	4

12  
34

Calculation Steps

- 1 Initialize first cell with cost  $|x_1-y_1|$

$$D(1,1) = |1-2| = 1$$

- 2 Fill remaining cells using:

$$D(i,j) = \text{cost}(i,j) + \min(D(i-1,j), D(i,j-1), D(i-1,j-1))$$

- 3 Example:  $D(2,2)$

$$|3-3| + \min(1, 2, 1) = 0 + 2 = 2$$

- 4 Continue until reaching  $D(n,m)$

5

Backtrack from end to find optimal path

Final DTW Distance

4

# DTW-based Clustering

Time Series Clustering with DTW Distance Metric

## Clustering Methods



### K-medoids

Better than K-means (pairwise distances)



### Hierarchical

DTW linkage for dendrogram



### DBA

DTW Barycenter Averaging for centroids



### Computational Cost

$$O(n^2m^2)$$

$n$  series  $\times$  length  $m$



### Speed Approximations

FastDTW

PrunedDTW

## DTW Clustering Workflow

1

Compute DTW distance matrix for all pairs

2

Apply clustering algorithm (K-medoids/Hierarchical)

3

Compute cluster centroids using DBA

4

Iterate until convergence or max iterations

5

Evaluate clustering quality with silhouette score



### Key Applications



Gesture  
Recognition



ECG Analysis



Stock Patterns



Speech Recognition

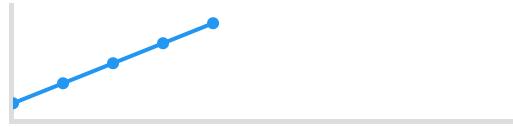
# DTW Clustering: Calculation Example

Step-by-step computation with 3 time series

## Input Time Series

Series A

[1, 2, 3, 4, 5]



Series B

[1, 3, 2, 4, 5]



Series C

[5, 4, 3, 2, 1]



## Step 1: Compute DTW Distance Matrix

	A	B	C
A	0.0	2.4	8.9
B	2.4	0.0	9.2
C	8.9	9.2	0.0

$$\text{DTW}(A,B) = 2.4 < \text{DTW}(A,C) = 8.9$$



## Step 2: Apply K-medoids (k=2)

Cluster 1



Medoid: Series A

Cluster 2



Medoid: Series C



## Step 3: DBA Centroid for Cluster 1

Average Alignment



Cluster 1 Centroid



[1.0, 2.5, 2.5, 4.0, 5.0]

# K-Shape Algorithm

Shape-based Time Series Clustering

## Key Features



Cross-correlation for similarity



Scale & shift invariant distance



Efficient shape-based centroid



Iterative refinement like K-means



Faster than DTW-based methods

✓ Best For

Normalized time series data

## Method Comparison

Feature	K-Shape	DTW
Speed	Fast	Slow
Scale Invariant	✓	✗
Shift Invariant	✓	✗
Time Warp	✗	✓
Complexity	$O(nm)$	$O(n^2m^2)$

## K-Shape Algorithm Flow

- 1 Initialize k cluster centroids
- 2 Assign series to nearest centroid (cross-correlation)
- 3 Update centroids (shape extraction)
- 4 Repeat until convergence

# K-Shape Algorithm Example

Step-by-Step Calculation with 2D Vectors

## 📌 Input Data: 4 Time Series (k=2 clusters)

$$\mathbf{x}_1 = [1, 2]$$

$$\mathbf{x}_2 = [2, 3]$$

$$\mathbf{x}_3 = [8, 9]$$

$$\mathbf{x}_4 = [9, 10]$$

## 🎯 Step 1: Normalization (z-score)

$$z = (x - \mu) / \sigma$$

$$\mathbf{z}_1 = [-1.0, 1.0]$$

$$\mathbf{z}_2 = [-1.0, 1.0]$$

$$\mathbf{z}_3 = [-1.0, 1.0]$$

$$\mathbf{z}_4 = [-1.0, 1.0]$$

### 💡 Example: Normalize $\mathbf{x}_1 = [1, 2]$

$$\mu = (1 + 2) / 2 = 1.5$$

$$\sigma = \sqrt{((1-1.5)^2 + (2-1.5)^2) / 2} = 0.5$$

$$z_1[0] = (1 - 1.5) / 0.5 = -1.0$$

$$z_1[1] = (2 - 1.5) / 0.5 = 1.0$$

Result

$$\mathbf{z}_1 = [-1.0, 1.0]$$

### 📝 Key Insight

All normalized vectors have the same shape pattern  $[-1.0, 1.0]$ , showing they follow the same increasing trend!

This normalization makes K-Shape **scale-invariant** - it focuses on shape, not magnitude.

# K-Shape Algorithm Example

Iteration 0: Initial Cluster Assignment

Iteration 0

## Step 2: Initialize Centroids (Random)

$$\mu_1 = [-1.0, 1.0]$$

$$\mu_2 = [-1.0, 1.0]$$

### Cross-Correlation Distance (SBD)

$$SBD(x, y) = 1 - \max(CC(x, y)) / (||x|| \times ||y||)$$
$$CC(x, y) = \sum x[i] \times y[i] \quad (\text{Cross-Correlation})$$

Calculate distance for  $z_1$  to  $\mu_1$ :

$$CC(z_1, \mu_1) = (-1.0) \times (-1.0) + (1.0) \times (1.0) = 2.0$$

$$||z_1|| = \sqrt{(1.0^2 + 1.0^2)} = 1.414$$

$$||\mu_1|| = 1.414$$

$$SBD(z_1, \mu_1) = 1 - 2.0 / (1.414 \times 1.414) = 0.0$$

Distance Result

$z_1 \rightarrow \text{Cluster 1 (distance} = 0.0)$

## Initial Cluster Assignment

Cluster 1

$$z_1 = [-1.0, 1.0]$$

Cluster 2

$$z_3 = [-1.0, 1.0]$$

$z_2 = [-1.0, 1.0]$

$z_4 = [-1.0, 1.0]$

# K-Shape Algorithm Example

Iteration 1: Update Centroids

Iteration 1

## Step 3: Update Cluster 1 Centroid

$\mu = \operatorname{argmax} \sum CC(x_i, y) / \|y\|^2$   
Simplified: Average of cluster members

Members:  $z_1, z_2 = [-1.0, 1.0]$

$$\mu_1[0] = (-1.0 + -1.0) / 2 = -1.0$$

$$\mu_1[1] = (1.0 + 1.0) / 2 = 1.0$$

Updated Centroid

$$\mu_1 = [-1.0, 1.0]$$

## Step 3: Update Cluster 2 Centroid

Members:  $z_3, z_4 = [-1.0, 1.0]$

$$\mu_2[0] = (-1.0 + -1.0) / 2 = -1.0$$

$$\mu_2[1] = (1.0 + 1.0) / 2 = 1.0$$

Updated Centroid

$$\mu_2 = [-1.0, 1.0]$$

## ⚠ Convergence Check

**Centroids unchanged!**

$$\mu_1 = \mu_2 = [-1.0, 1.0]$$

**✓ Algorithm converged!**

## 💡 Why did it converge so quickly?

All normalized vectors have identical shape  $[-1.0, 1.0]$ , so they all have perfect cross-correlation with any centroid of the same shape. The algorithm converged in just one iteration!

# K-Shape Algorithm Example

## Final Results & Analysis

### Final Cluster Assignment

#### Cluster 1

$x_1 = [1, 2]$

$x_2 = [2, 3]$

Centroid: [-1.0, 1.0]

#### Cluster 2

$x_3 = [8, 9]$

$x_4 = [9, 10]$

Centroid: [-1.0, 1.0]

### Key Observations

#### 1. Shape-based clustering:

Groups series by pattern, not absolute values

#### 2. Same normalized shape:

All series show increasing trend [-1, 1]

#### 3. Scale invariance:

[1,2] and [8,9] treated similarly

### Algorithm Insights

#### ✓ Fast convergence:

Converged in 1 iteration

#### ✓ Cross-correlation:

Measures shape similarity efficiently

#### ✓ Normalization effect:

All vectors → same pattern

### Complexity Analysis

For n=4 series, m=2 dimensions, k=2 clusters:

K-Shape

$O(4 \times 2) = O(8)$

DTW

$O(16 \times 4) = O(64)$

K-Shape is 8x faster! ⚡

# Subsequence Clustering

Finding Recurring Patterns in Long Time Series

## Subsequence Extraction & Clustering Pipeline



Sliding Window Example:



## Key Challenges

### ⚠️ Trivial Matches

Self-matches and near-duplicates

### ⚠️ Overlapping Patterns

Adjacent subsequences too similar

## Applications & Use Cases



### Key Applications

## Solutions



### Matrix Profile

Efficient motif discovery technique



### STOMP Algorithm

Scalable Time series Ordered-search Matrix Profile



Anomaly  
Detection



Pattern Mining



Motif Discovery



Sensor Analysis



### Example Use Case

Identify repeated behaviors in IoT sensor data: recurring activity patterns, operational cycles, or anomalous sequences in manufacturing or healthcare monitoring systems.

# Hands-on: Stock/Sensor Data Clustering

Time Series Clustering Implementation Workflow

## 1 Dataset Selection

Yahoo Finance or UCI sensor data



## 2 Preprocessing

Normalization & resampling

`z-score`

`interpolate`



## 3 DTW Distance

Implement distance function

`Python/NumPy`



## 4 Hierarchical Clustering

Apply DTW linkage

`scipy.cluster`



## 5 Compare K-Shape

Alternative algorithm comparison

`tslearn`



## 6 Visualization

Plot clusters & centroids

`matplotlib`



## 7 Interpret Results

Identify similar behaviors



### Data Sources



Yahoo Finance (stocks) • UCI ML Repository  
(sensors)



### Key Question

Which stocks/sensors behave similarly?

**Part 3/4:**

# **Graph Clustering**

- 14.** Introduction to Graph Data
- 15.** Spectral Clustering
- 16.** Community Detection (Louvain)
- 17.** GNN Fundamentals
- 18.** Clustering with GCN
- 19.** Deep Graph Infomax
- 20.** Hands-on: Social Network Analysis



## Graph

Nodes (vertices) connected by Edges

### Graph Types

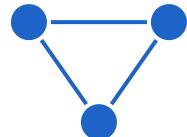
Directed

Undirected

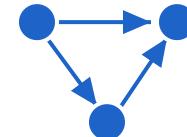
Weighted

Attributed

### Undirected



### Directed



### Graph Properties



**Degree:** # of connections

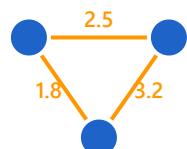


**Density:** edge ratio

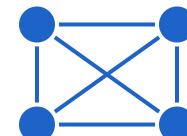


**Clustering coeff:** local structure

### Weighted



### Complete



## Numerical Vector Examples

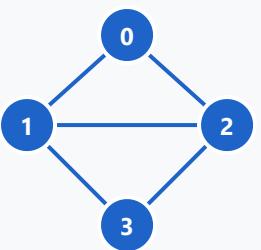
### Sample Graph G

### Adjacency Matrix A

	0	1	2	3
0	0	1	1	0

### Edge List E

```
# List of (source, target) tuples E = [(0, 1), # node 0 ↔ node 1 (0, 2), # node 0
```



$V = \{0, 1, 2, 3\}$   
 $|E| = 5$  edges

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

### Adjacency List

```
# Dict: node → neighbors adj = { 0: [1, 2], 1: [0, 2, 3], 2: [0, 1, 3], 3: [1, 2] }
```

```
↳ node 2 (1, 2), # node 1 ↳ node 2 (1, 3),
# node 1 ↳ node 3 (2, 3) # node 2 ↳ node 3
]
```

### Node Feature Matrix X

Node	$f_1$	$f_2$	$f_3$	$f_4$
0	0.5	1.2	-0.3	0.8
1	1.1	-0.5	0.7	0.2
2	-0.2	0.9	1.5	-0.1
3	0.8	0.3	-0.6	1.4

$X \in \mathbb{R}^{4 \times 4}$  (4 nodes  $\times$  4 features)

### Computed Graph Properties

Degree Vector

$$\mathbf{d} = [2, 3, 3, 2]$$

$\deg(0)=2, \deg(1)=3, \deg(2)=3, \deg(3)=2$

Graph Density

$$\rho = 0.833$$

$2|E| / (|V|(|V|-1)) = 10/12$

Avg Clustering Coeff

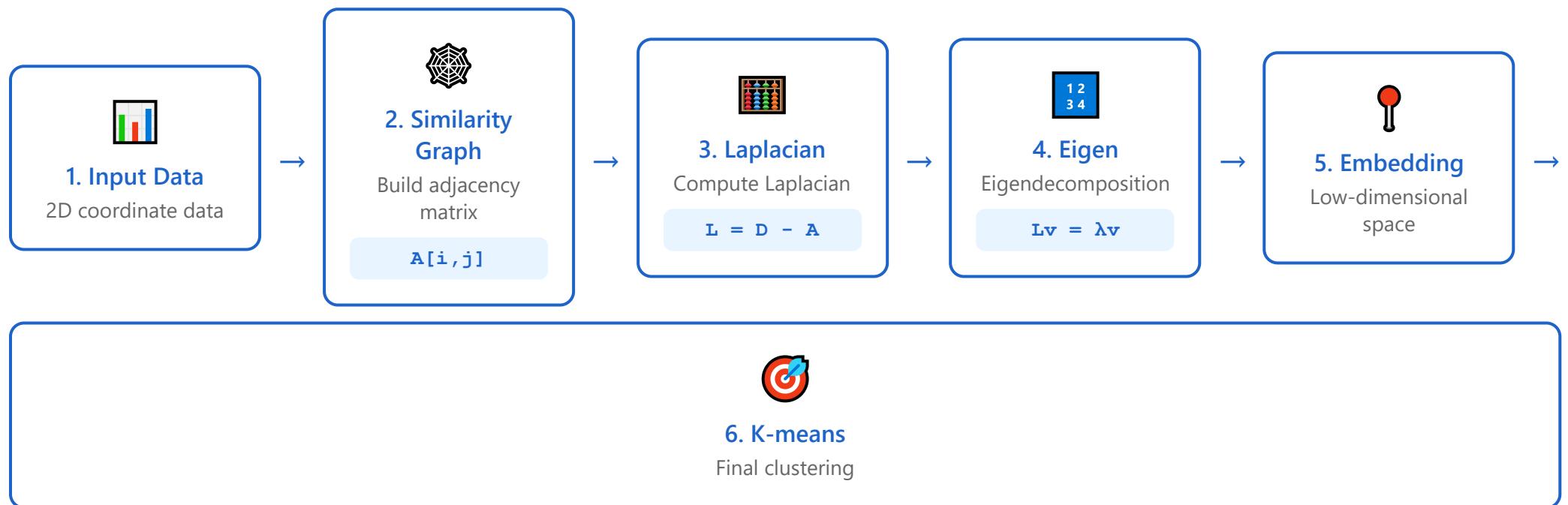
$$\bar{C} = 0.667$$

$(1 + 0.33 + 0.33 + 1) / 4$

# Spectral Clustering

Step-by-Step Calculation Example

## Spectral Clustering Pipeline



## Step-by-Step Calculation Example

1 Input Data

2 Adjacency Matrix

$p_1 = (1, 1)$

$p_2 = (1, 2)$

$p_3 = (2, 1)$

$p_4 = (5, 5)$

$p_5 = (5, 6)$

$p_6 = (6, 5)$

### Data Description

6 points in 2D space

- $p_1, p_2, p_3$ : Lower-left cluster
- $p_4, p_5, p_6$ : Upper-right cluster

$$A[i,j] = \exp(-\|p_i - p_j\|^2 / 2\sigma^2)$$

Using  $\sigma = 1.0$ , higher weight for closer points

Matrix A ( $6 \times 6$ )

1.00	0.61	0.61	0.00	0.00	0.00
0.61	1.00	0.37	0.00	0.00	0.00
0.61	0.37	1.00	0.00	0.00	0.00
0.00	0.00	0.00	1.00	0.61	0.61
0.00	0.00	0.00	0.61	1.00	0.37
0.00	0.00	0.00	0.61	0.37	1.00

Block diagonal structure represents two clusters

### 3 Degree Matrix

$$D[i,i] = \sum_j A[i,j]$$

Sum of edge weights for each node

Matrix D ( $6 \times 6$  diagonal)

### 4 Laplacian Matrix

$$L = D - A$$

Matrix representing graph structure

Matrix L ( $6 \times 6$ )

2.22	0	0	0	0	0
0	1.98	0	0	0	0
0	0	1.98	0	0	0
0	0	0	2.22	0	0
0	0	0	0	1.98	0
0	0	0	0	0	1.98

1.22	-0.61	-0.61	0	0	0
-0.61	0.98	-0.37	0	0	0
-0.61	-0.37	0.98	0	0	0
0	0	0	1.22	-0.61	-0.61
0	0	0	-0.61	0.98	-0.37
0	0	0	-0.61	-0.37	0.98

Block diagonal structure = two separated clusters

## 5 Eigendecomposition

$$L\mathbf{v} = \lambda\mathbf{v}$$

Select eigenvectors corresponding to k smallest eigenvalues

### Eigenvalues

$\lambda_1 = 0.00$  (first, connected graph)

$\lambda_2 = 0.00$  (second, 2 clusters!)

$\lambda_3 = 0.61$

$\lambda_4 = 1.59$

## 6 Spectral Embedding

$$\mathbf{X} = [\mathbf{v}_2, \mathbf{v}_3, \dots, \mathbf{v}_{k+1}]$$

For k=2 clusters, use  $v_2$  and  $v_3$

### Embedding Coordinates (6×2)

$\lambda_5 = 1.59$

$\lambda_6 = 1.83$

### ⚠ Key Observation

2 eigenvalues near zero  $\rightarrow$  2 clusters exist!

Number of eigenvalues = Number of connected components (clusters)

### Second Eigenvector ( $v_2$ )

$v_2 = [0.58, 0.58, 0.58, -0.58, -0.58, -0.58]^T$

Positive values  $\rightarrow$  Cluster 1 ( $p_1, p_2, p_3$ )

Negative values  $\rightarrow$  Cluster 2 ( $p_4, p_5, p_6$ )

0.58	-0.12
0.58	0.45
0.58	-0.33
-0.58	-0.12
-0.58	0.45
-0.58	-0.33

First column ( $v_2$ ) clearly separates clusters  
Positive (0.58) vs Negative (-0.58)

## 7 K-means Clustering

### K-means ( $X, k=2$ )

Perform K-means in embedding space

### ✓ Final Cluster Assignment

Cluster 1:  $\{p_1, p_2, p_3\}$

Cluster 2:  $\{p_4, p_5, p_6\}$

### Why Does This Work?

1. Original space requires non-linear separation
2. Laplacian eigenvectors reflect graph structure
3. Embedding space allows linear separation
4. K-means easily discovers clusters



### Key Insights

#### Advantages of Spectral Clustering

- Can find non-convex clusters
- Directly utilizes graph structure
- Can estimate number of clusters from eigenvalues
- Theoretical guarantees (minimizes normalized cut)

#### Complete Process Summary

Data → Similarity Graph → Laplacian → Eigenvectors → Low-dim Embedding → K-means

# Community Detection - Louvain Algorithm

Finding Densely Connected Groups in Networks

## Communities

Densely connected groups with sparse inter-group connections

## Modularity Optimization

Measure of community structure quality

$$Q = \sum [e_{in} - (d_{tot}/2m)^2]$$

## Community Structure Visualization



## Two-Phase Iterative Approach

### 1 Local Phase

Move nodes to neighboring communities to maximize modularity gain

## ✓ Key Advantages

⚡ Fast & Scalable

🌲 Hierarchical

☒ Large Networks

🎯 High Quality

## Applications

## 2 Global Phase

Aggregate communities into super-nodes, build new network

 Iterate until modularity cannot be improved

Social Networks

Protein Interactions

Web Analysis

Citation Networks

## Step 0: Initial State

2D Vector Input and Graph Construction

## Input: 2D Vectors

**v0:** (1.0, 1.5)  
**v1:** (1.2, 1.8)  
**v2:** (1.5, 1.3)  
**v3:** (5.0, 5.2)  
**v4:** (5.3, 5.5)  
**v5:** (5.1, 4.9)

**Connection Criteria:** Euclidean distance  $\leq 1.0$

### Generated Edges:

- (v0, v1): 0.36 • (v0, v2): 0.54
- (v1, v2): 0.58 • (v3, v4): 0.42
- (v3, v5): 0.36 • (v4, v5): 0.63

**Total Edges (m):** 6



### Initial State

Each node forms its own community.

**Q = 0.000**

## Step 1: Move Node v0

v0 → C1 ( $\Delta Q = +0.0833$ )

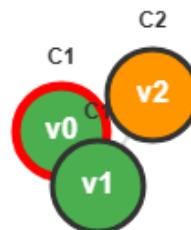
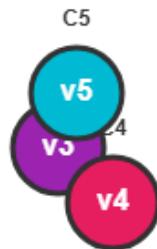
### Node v0 Analysis

**Neighbors:** v1(C1), v2(C2)

#### Move Simulation:

- To C1:  $\Delta Q = +0.0833 \checkmark$
- To C2:  $\Delta Q = +0.0694$

#### Move v0 to C1



v0 → C1

Q = 0.083

C1={v0, v1}, C2={v2}, C3={v3}, C4={v4}, C5={v5}

## Step 2: Check Node v1

v1 stays (already optimal)

### Node v1 Analysis

**Neighbors:** v0(C1), v2(C2)

#### Move Simulation:

- To C2:  $\Delta Q \approx 0$

**Stay in current position**



v1 stays

No change

Q = 0.083

## Step 3: Move Node v2

$v2 \rightarrow C1 (\Delta Q = +0.167)$

### Node v2 Analysis

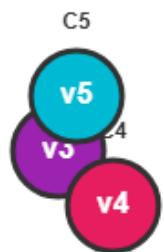
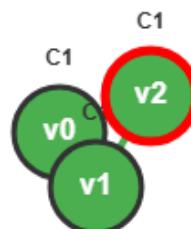
**Neighbors:**  $v0(C1), v1(C1)$

#### Move Simulation:

- To  $C1: \Delta Q = +0.167 \checkmark$

#### Move v2 to C1

Left cluster complete!



**Q = 0.250**

**v2 → C1**

C1={v0, v1, v2}, C3={v3}, C4={v4}, C5={v5}

## **Step 4: Move Node v3**

v3 → C4 ( $\Delta Q = +0.0833$ )

---

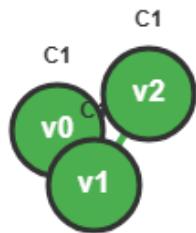
## Node v3 Analysis

**Neighbors:** v4(C4), v5(C5)

### Move Simulation:

- To C4:  $\Delta Q = +0.0833 \checkmark$
- To C5:  $\Delta Q = +0.0694$

### Move v3 to C4



v3 → C4

C1={v0, v1, v2}, C4={v3, v4}, C5={v5}

Q = 0.333

## Step 5: Check Node v4

v4 stays

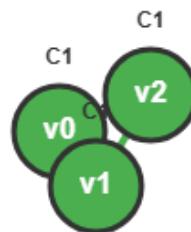
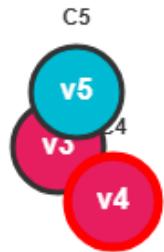
### Node v4 Analysis

**Neighbors:** v3(C4), v5(C5)

**Move Simulation:**

- To C5:  $\Delta Q \approx 0$

**Stay in current position**



v4 stays

No change

$Q = 0.333$

## Step 6: Phase 1 Complete

$v5 \rightarrow C4 (\Delta Q = +0.167)$

### Node v5 Analysis

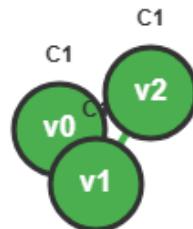
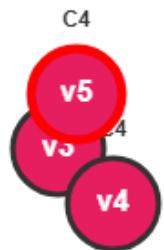
**Neighbors:**  $v3(C4), v4(C4)$

#### Move Simulation:

- To  $C4: \Delta Q = +0.167 \checkmark$

**Move  $v5$  to  $C4$**

**Phase 1 Complete!**



**Phase 1 Complete**

**$Q = 0.500$**

**Final:**  $C1 = \{v0, v1, v2\}, C4 = \{v3, v4, v5\}$

## Step 7: Phase 2 - Final Result

Super-node Creation and Algorithm Convergence

### ⌚ Super-nodes

**S1 = {v0, v1, v2}**

- Internal edges: 3
- Degree: 6

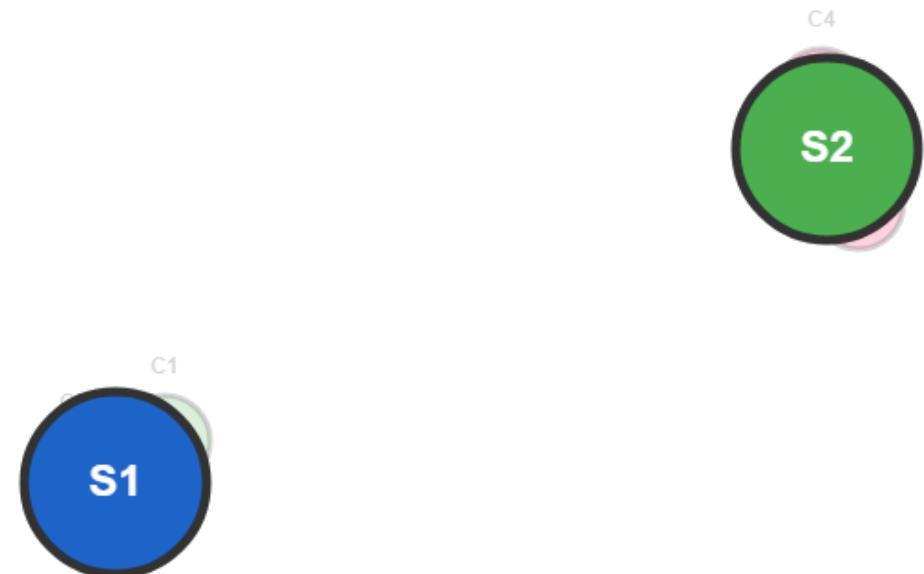
**S2 = {v3, v4, v5}**

- Internal edges: 3
- Degree: 6

### New Graph:

- Nodes: 2
- Edges: 0

✓ Algorithm Terminated



**Q = 0.500**

### Final Result

**2 communities found!**

No further improvement possible



## Algorithm Summary

### Phase 1 Results

- Node moves: **4**
- Initial communities: **6**
- Final communities: **2**
- Modularity: **0.000 → 0.500**

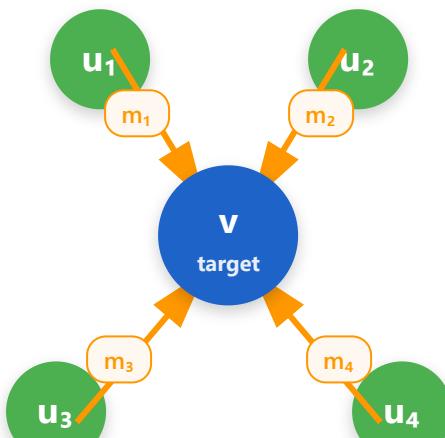
### Phase 2 Results

- Super-nodes: **2**
- Hierarchy levels: **2**
- Inter-community edges: **0**
- Status: **Converged ✓**

# Graph Neural Networks (GNN) Fundamentals

Deep Learning on Graph-Structured Data

## Message Passing Mechanism



### 1 Aggregation

Collect neighbor messages

### 2 Transformation

Apply neural network

### 3 Update

Compute new representation

## GNN Pipeline

Node Features



Hidden Representations



Predictions

## ✓ Key Properties

⟳ Permutation invariant

📊 Learn node embeddings

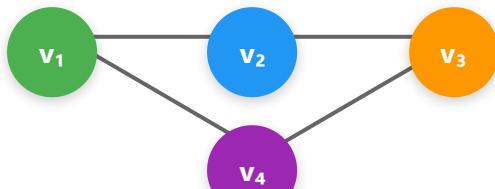
📍 Preserve graph structure

## 🚀 Foundation

Basis for modern graph machine learning methods

# How to Convert Graph to Convolution Input

## Example: Simple Graph



Adjacency Matrix (A)

$$\begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix}$$

Feature Matrix (X)

$$\begin{bmatrix} x_1 & 1 & x_1 & 2 & \dots & x_1 & a \\ x_2 & 1 & x_2 & 2 & \dots & x_2 & a \\ x_3 & 1 & x_3 & 2 & \dots & x_3 & a \\ x_4 & 1 & x_4 & 2 & \dots & x_4 & a \end{bmatrix}$$

## Graph Convolution Operation

### Step 1: Normalize Adjacency

$$\tilde{A} = D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$$

(Add self-loops & normalize)



### Step 2: Aggregate Features

$$\tilde{H} = \tilde{A} X$$

(Weighted sum of neighbors)



### Step 3: Apply Weights

$$H' = \tilde{H} W$$

(Linear transformation)



### Step 4: Non-linearity

$$H^{(l+1)} = \sigma(H')$$

(Apply activation function)

### Complete Formula

$$H^{(l+1)} = \sigma(\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} H^{(l)} W^{(l)})$$

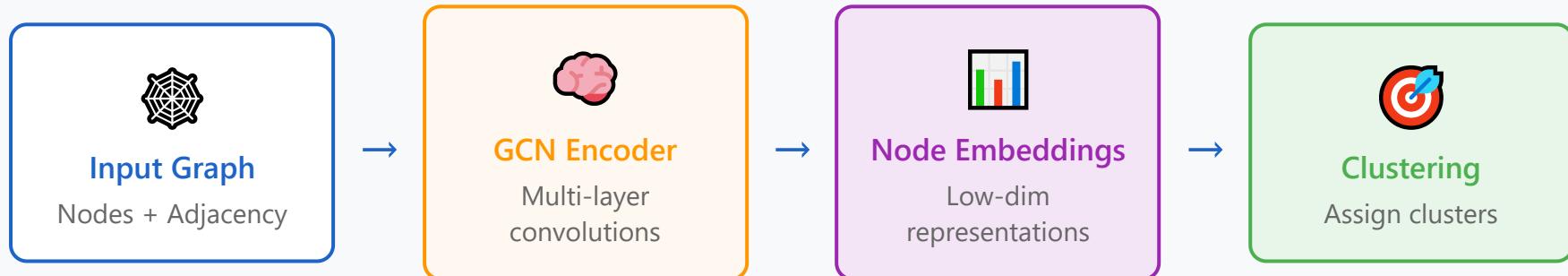
### Key Points

- **Adjacency Matrix (A):** Encodes graph structure (who connects to whom)
- **Feature Matrix (X):** Node features (d-dimensional vectors for each node)
- **Degree Matrix (D):** Diagonal matrix with node degrees for normalization
- **Weight Matrix (W):** Learnable parameters (like CNN filters)
- **Output (H):** New node representations that incorporate neighborhood information

# Clustering with Graph Convolutional Networks (GCN)

End-to-End Trainable Graph Clustering

## GCN-based Clustering Architecture



### Reconstruction Loss



Preserve graph structure

$$L_{\text{recon}} = ||A - \hat{A}||^2$$

### Clustering Loss



Optimize cluster assignment

$$L_{\text{cluster}} = KL(P || Q)$$

End-to-End Trainable:  $L_{\text{total}} = L_{\text{recon}} + \lambda \cdot L_{\text{cluster}}$



### Key Features

- ✓ Supervised or unsupervised learning
- ✓ Captures local & global structure



### Advantages

- ✓ Outperforms spectral methods
- ✓ Better on benchmarks

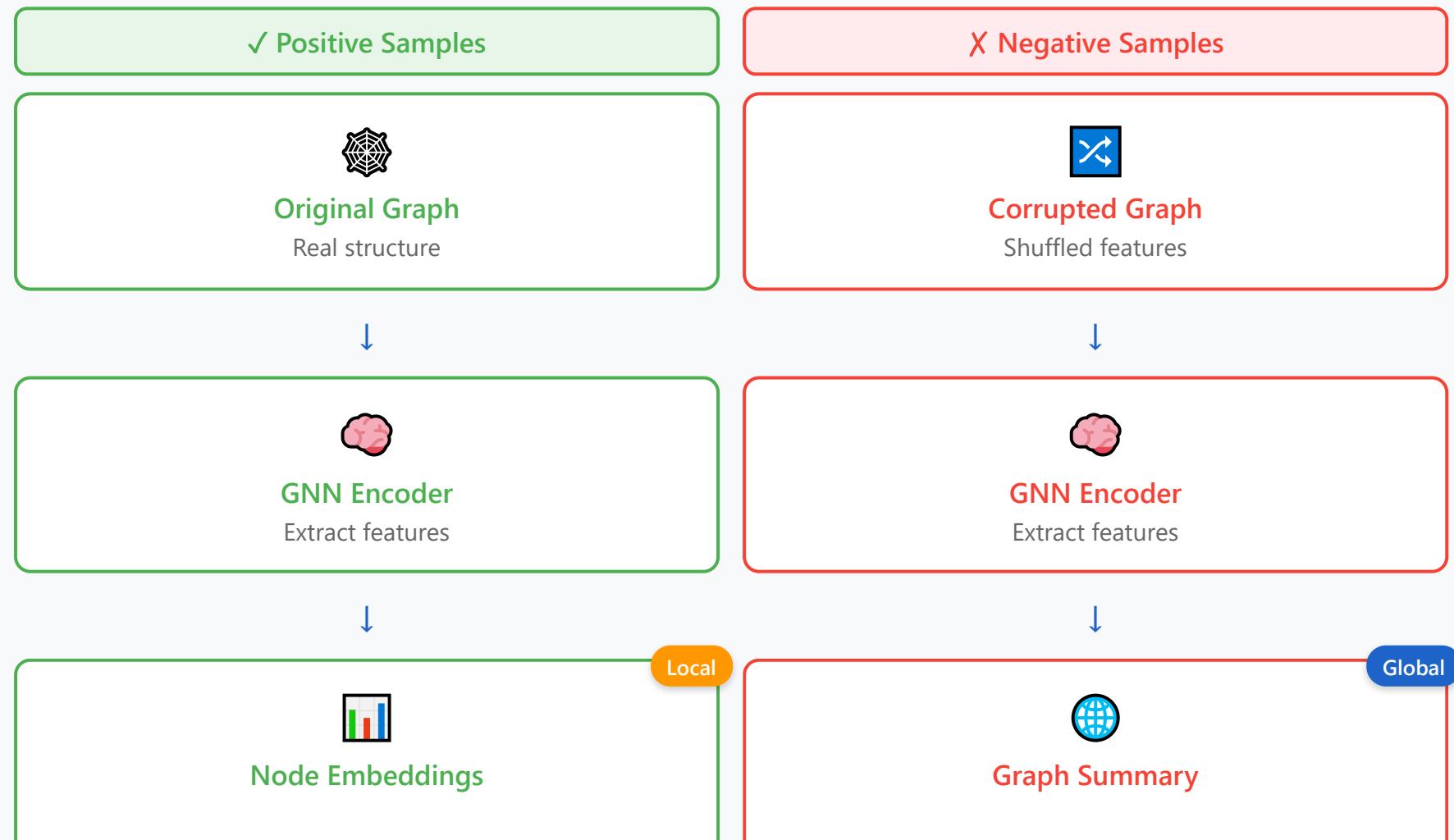
✓ Joint optimization of encoder & clustering

✓ Learns rich node representations

# Deep Graph Infomax (DGI)

Self-Supervised Graph Representation Learning

## Contrastive Learning Architecture



 **Objective: Maximize Mutual Information**

Maximize MI between local node embeddings and global graph summary

**Self-Supervised**

Learn from graph structure alone

**No Labels Needed**

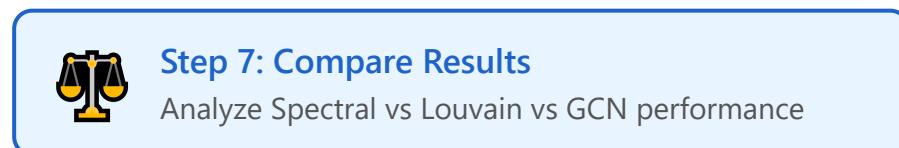
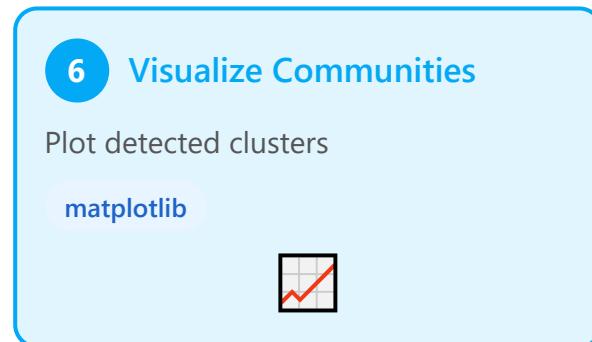
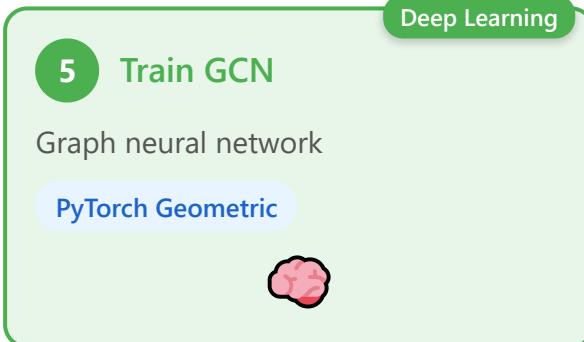
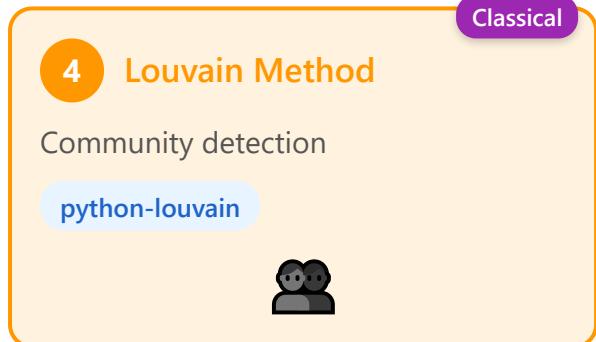
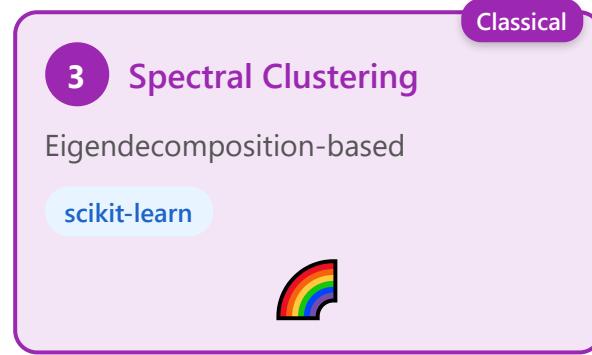
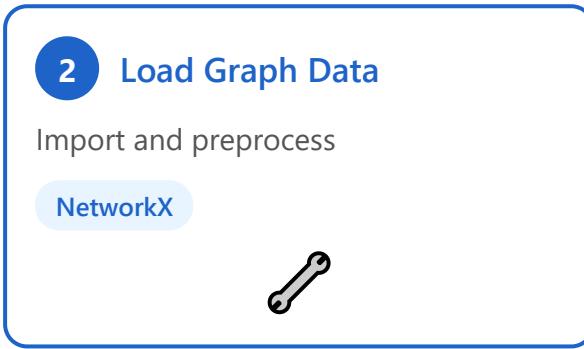
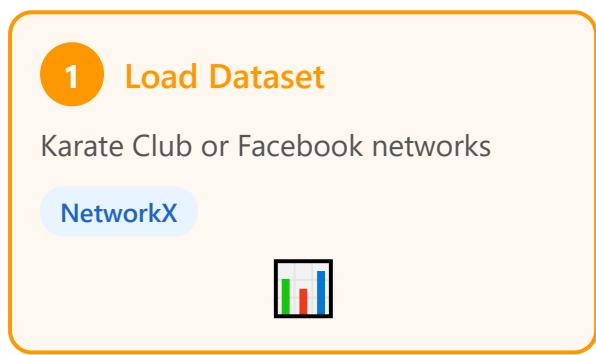
Unsupervised training approach

**Transfer Learning**

Apply to downstream tasks

# Hands-on: Social Network Analysis

Graph Clustering Implementation Workflow



**Part 4/4:**

## **Advanced Topics and Applications**

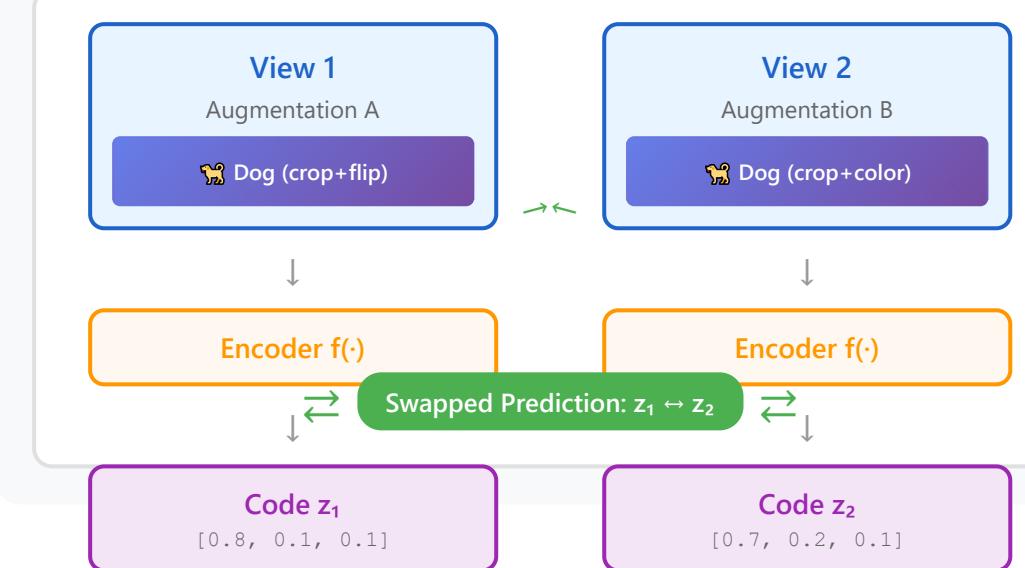
- 21.** Deep Clustering (DeepCluster, SwAV)
- 22.** Multi-modal Clustering
- 23.** Large-Scale Clustering
- 24.** Real-world Applications (by Industry)



## DeepCluster

- ✓ Alternates clustering and representation learning
- ✓ Cluster assignments as pseudo-labels
- ✓ Iterative refinement process

## SwAV Architecture & Training Process



## SwAV

- ✓ Swapping Assignments between Views
- ✓ Online clustering with prototypes
- ✓ Learnable cluster centroids
- ✓ No pairwise comparisons needed



### Efficient

No pairwise comparisons



### Online

Real-time clustering



### Prototypes

Learnable centroids



### Scalable

Large datasets



State-of-the-Art Self-Supervised Learning Results



# DeepCluster Operating Principle

## Algorithm Flow

1

### Feature Extraction

Transform images into feature vectors using CNN

$$x \rightarrow f(x) \in \mathbb{R}^d$$

2

### K-means Clustering

Group features into K clusters

$$\text{assign: } \operatorname{argmin} \|f(x) - c_k\|^2$$

3

### Pseudo-label Assignment

Use cluster ID as pseudo-label

$$y_{\text{pseudo}} = \text{cluster\_id}$$

4

### Network Update

Train CNN with pseudo-labels (classification)

$$L = -\log P(y_{\text{pseudo}} | x)$$



### Problem: Trivial Solutions

All samples collapse into a single cluster



### Solution: Empty Cluster Reassignment

Fill empty clusters with samples from the largest cluster



# SwAV Operating Principle

## 1. Multi-Crop Augmentation

 224×224 ( $\times 2$ ) 96×96 ( $\times 4$ )

Generate multiple views with various crop sizes

## 2. Prototype Assignment

$$q = \text{softmax}(z \cdot C / \tau)$$

z: feature, C: prototypes,  $\tau$ : temperature

Soft-assign each view to prototypes

## 3. Swapped Prediction

$z_1 \rightarrow \text{predict } q_2$

$z_2 \rightarrow \text{predict } q_1$

Predict View 2 from View 1's code (and vice versa)

## 4. Loss Computation

$$L = \ell(z_1, q_2) + \ell(z_2, q_1)$$

Cross-entropy between predictions



### Online Clustering

Update prototypes every batch (no K-means needed)



### Sinkhorn-Knopp

Ensures balanced cluster assignment  
(equipartition)



### Memory Efficiency

No need to store negative pairs (vs SimCLR)

# Multi-modal Clustering

Clustering Data with Multiple Modalities

## ⚠ Challenge

Different feature spaces and scales across modalities

## Three Clustering Approaches



### Early Fusion

Concatenate features from different modalities



### Late Fusion

Cluster each modality separately, then combine results



### Deep Multi-modal

Learn shared representation across modalities



Image

+



Text

→

### Shared Space

Joint representation



Applications



### Modern Approach

CLIP-style contrastive learning for multi-modal clustering



Video Analysis (visual + audio)



Medical Imaging with Reports

Contrastive Multi-modal Learning

# Large-Scale Clustering

Scalable Techniques for Billions of Samples



Challenges:

Billions of samples

High dimensionality

## Scalability Techniques



### Mini-batch K-means

Process streaming data in small batches

scikit-learn



### Approximate NN

Fast nearest neighbor search

FAISS



### Hierarchical

Multi-level clustering for scalability

scipy



### Distributed

Parallel processing across clusters

MapReduce • Spark MLlib



### GPU Acceleration

Fast distance computations

CUDA • cuML



### Sampling

Representative subset selection

CoreSets



## Critical Trade-off

Speed vs Accuracy

# Real-world Applications by Industry

Clustering Applications Across Different Sectors



## E-commerce

- Customer segmentation
- Product recommendations



## Healthcare

- Patient stratification
- Disease subtyping



## Finance

- Fraud detection
- Portfolio clustering



## Manufacturing

- Anomaly detection
- Predictive maintenance



## Social Media

- Community detection
- Content recommendation



## Genomics

- Gene expression clustering
- Cell type discovery



## Transportation

- Traffic pattern analysis
- Route optimization

# Thank you

Ho-min Park

[homin.park@ghent.ac.kr](mailto:homin.park@ghent.ac.kr)

[powersimmani@gmail.com](mailto:powersimmani@gmail.com)