# DSC 140B
## Representation Learning

Lecture 16 | Part 1

## From Points to Graphs

# Dimensionality Reduction

- **Given**: $n$ points in $\mathbb{R}^d$, number of dimensions $k \leq d$

- **Map**: each point $\vec{x}$ to new representation $\vec{z} \in \mathbb{R}^k$

# Idea

- Build a similarity graph from points in $\mathbb{R}^2$
- Use approach from last lecture to embed into $\mathbb{R}^k$
- But how do we represent a set of points as a similarity graph?

# Three Approaches

- 1) Epsilon neighbors graph
- 2) *k*-Nearest neighbor graph
- 3) fully connected graph with similarity function

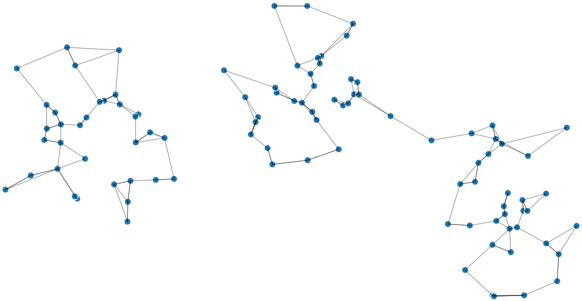
# Epsilon Neighbors Graph

- Input: vectors $\vec{x}^{(1)}, \ldots, \vec{x}^{(n)}$, a number $\varepsilon$
- Create a graph with one node $i$ per point $\vec{x}^{(i)}$
- Add edge between nodes $i$ and $j$ if $\|\vec{x}^{(i)} - \vec{x}^{(j)}\| \leq \varepsilon$
- Result: **unweighted** graph

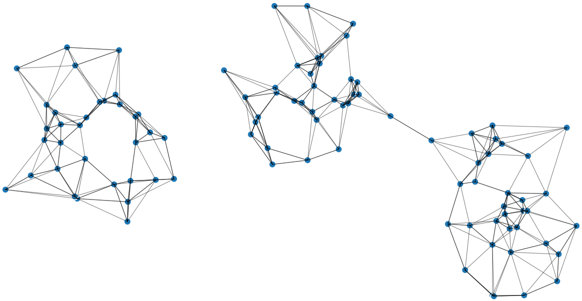
# k-Neighbors Graph

- Input: vectors $\vec{x}^{(1)}, \ldots, \vec{x}^{(n)}$, a number $k$
- Create a graph with one node $i$ per point $\vec{x}^{(i)}$
- Add edge between each node $i$ and its $k$ closest neighbors
- Result: **unweighted** graph

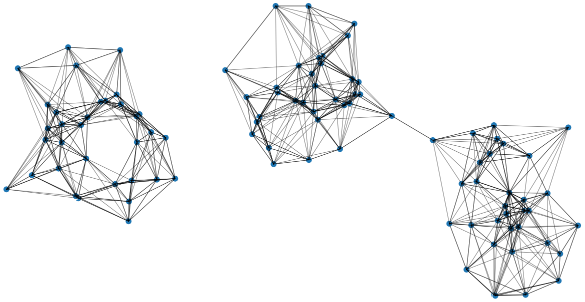
# k-Neighbors Graph



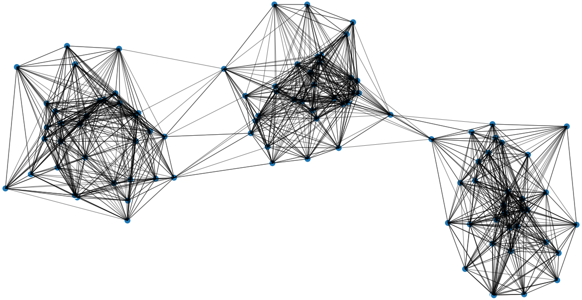
# k-Neighbors Graph



# k-Neighbors Graph



# k-Neighbors Graph



# Fully Connected Graph

- Input: vectors $\vec{x}^{(1)}, \dots, \vec{x}^{(n)}$, a similarity function $h$
- Create a graph with one node $i$ per point $\vec{x}^{(i)}$
- Add edge between every pair of nodes. Assign weight of $h(\vec{x}^{(i)}, \vec{x}^{(j)})$
- Result: **weighted** graph

# Gaussian Similarity

- A common similarity function: Gaussian

- Must choose $\sigma$ appropriately

$$h(\vec{x}, \vec{y}) = e^{-\|\vec{x}-\vec{y}\|^2/\sigma^2}$$

# Fully Connected: Pseudocode

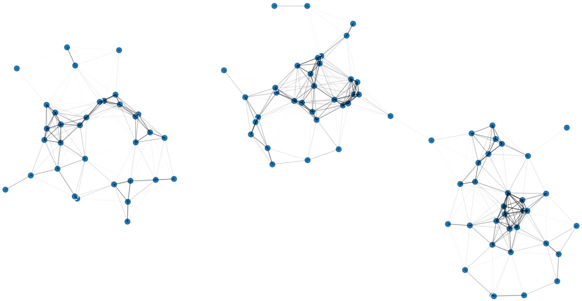
```
def h(x, y):
    dist = np.linalg.norm(x, y)
    return np.exp(-dist**2 / sigma**2)

# assume the data is in X
n = len(X)
w = np.ones_like(X)
for i in range(n):
    for j in range(n):
        w[i, j] = h(X[i], X[j])
```

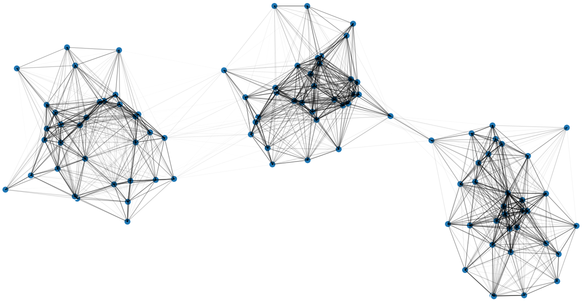
# With SciPy

```
distances = scipy.spatial.distance_matrix(X, X)
w = np.exp(-distances**2 / sigma**2)
```

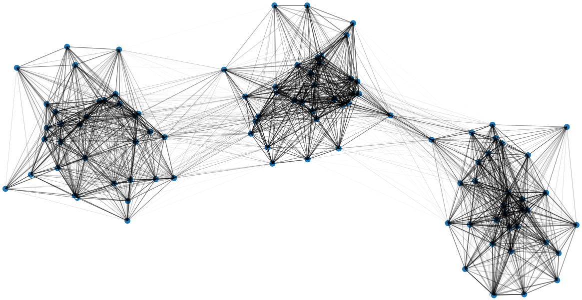
# Gaussian Similarity



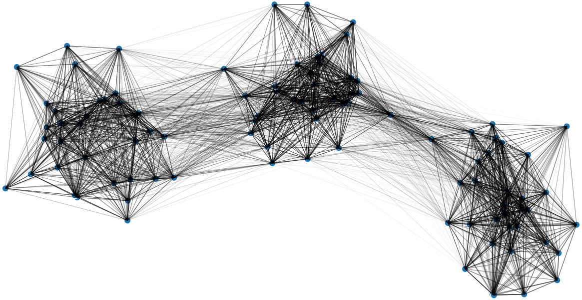
# Gaussian Similarity



# Gaussian Similarity



# Gaussian Similarity



# DSC 140B
## Representation Learning

Lecture 16 | Part 2

## Summary: Laplacian Eigenmaps

# Problem: Graph Embedding

- **Given**: a similarity graph, target dimension $k$
- **Goal**: **embed** the nodes of the graph as points in $\mathbb{R}^k$ so that similar nodes are nearby
- **(One) Solution:** Embed using eigenvectors of the graph Laplacian

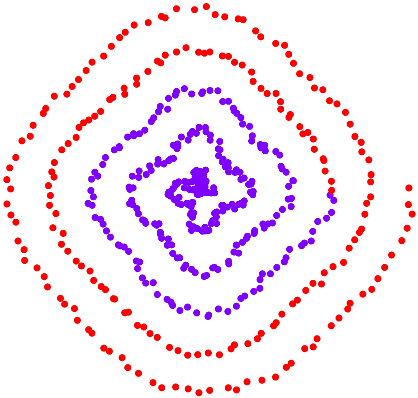
# Problem: Non-linear Dimensionality Reduction

- **Given**: points in $\mathbb{R}^d$, target dimension $k$
- **Goal**: **embed** the points in $\mathbb{R}^k$ so that points that were close in $\mathbb{R}^d$ are close after

# Idea

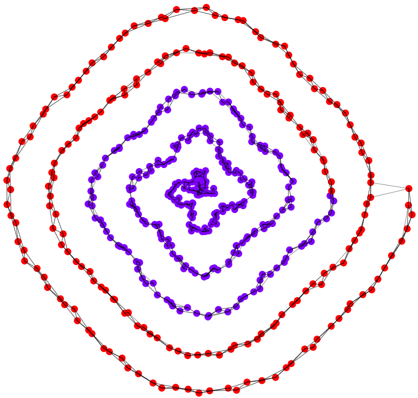
- Build a similarity graph from points in $\mathbb{R}^d$
  - epsilon neighbors, $k$-neighbors, or fully connected
- Embed the similarity graph in $\mathbb{R}^k$ using eigenvectors of graph Laplacian

# Example 1: Spiral



# Example 1: Spiral

- Build a $k$-neighbors graph.
- Note: follows the 1-d shape of the data.



# Example 1: Spectral Embedding

- Let $W$ be the weight matrix ($k$-neighbor adjacency matrix)

- Compute $L = D - W$

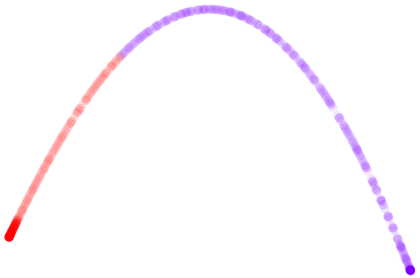
- Compute bottom $k$ non-constant eigenvectors of $L$, use as embedding

# Example 1: Spiral

- Embedding into $\mathbb{R}^1$

# Example 1: Spiral

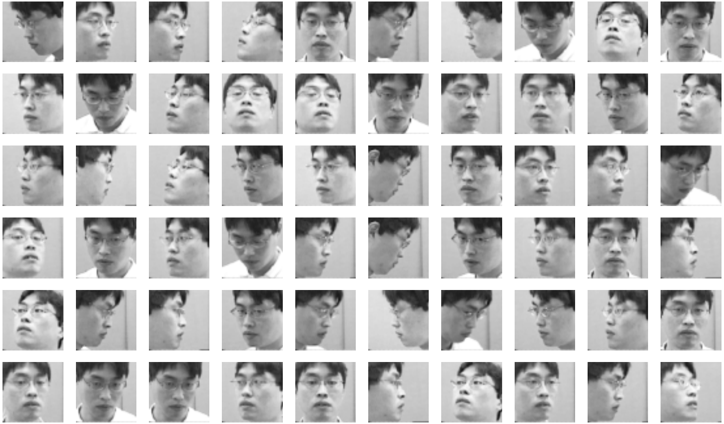
- Embedding into $\mathbb{R}^2$



# Example 1: Spiral

```
import sklearn.neighbors
import sklearn.manifold
W = sklearn.neighbors.kneighbors_graph(
        X, n_neighbors=4
)
embedding = sklearn.manifold.spectral_embedding(
        W, n_components=2
)
```

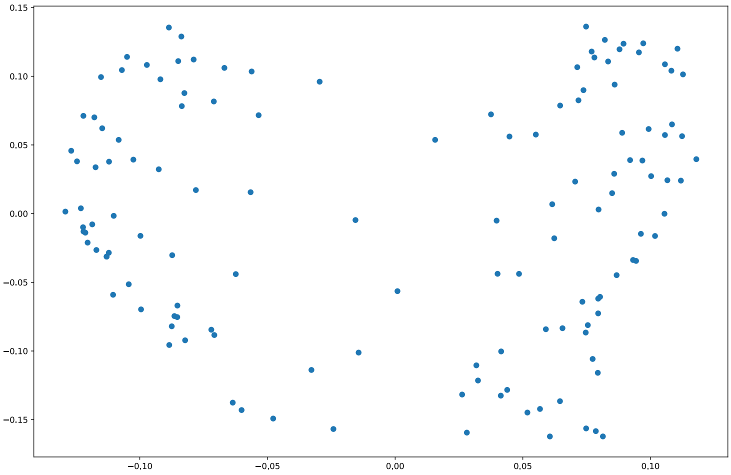
# Example 2: Face Pose



# Example 2: Face Pose

- Construct fully-connected similarity graph with Gaussian similarity
- Embed with Laplacian eigenmaps

# Example 2: Face Pose



# Example 2: Face Pose