Lecture 10
CNNs on Graphs

CMSC 35246: Deep Learning

Shubhendu Trivedi
&
Risi Kondor

University of Chicago

April 26, 2017

Two Scenarios

- For CNNs on graphs, we have two distinct scenarios:
  - Scenario 1: Each data point lives in $\mathbb{R}^d$, but the dataset has an underlying graph structure

Two Scenarios

- For CNNs on graphs, we have two distinct scenarios:
  - Scenario 1: Each data point lives in $\mathbb{R}^d$, but the dataset has an underlying graph structure
    - Each coordinate is a value associated with a vertex of underlying graph

Two Scenarios

- For CNNs on graphs, we have two distinct scenarios:
  - Scenario 1: Each data point lives in $\mathbb{R}^d$, but the dataset has an underlying graph structure
    - Each coordinate is a value associated with a vertex of underlying graph
    - For images: The underlying graph is always a grid of fixed dimensions
  - Scenario 2: Each data point is itself a graph (Example regression task: Molecules as input, boiling points as output)
    - Each graph can be of different size
    - Sub-problem: Given a graph $\mathcal{G}$, find an embedding $\phi : \mathcal{G} \to \mathbb{R}^p$

Scenario 1

CNNs on data in irregular domains

CNNs on Grids

- So far we have defined CNNs on grids

CNNs on Grids

- So far we have defined CNNs on grids
- We model images and feature maps as functions on a rectangular domain

CNNs on Grids

- So far we have defined CNNs on grids
- We model images and feature maps as functions on a rectangular domain

$$f : \mathbb{Z}^2 \to \mathbb{R}^K$$

- In general the grid can be $\mathbb{Z}^d$
- CNNs are able to exploit various structures that reduce sample complexity

CNNs on Grids

- So far we have defined CNNs on grids
- We model images and feature maps as functions on a rectangular domain

$$f : \mathbb{Z}^2 \rightarrow \mathbb{R}^K$$

- In general the grid can be $\mathbb{Z}^d$
- CNNs are able to exploit various structures that reduce sample complexity
  - Translation structure (allowing use of filters)

CNNs on Grids

- So far we have defined CNNs on grids
- We model images and feature maps as functions on a rectangular domain

$$f : \mathbb{Z}^2 \to \mathbb{R}^K$$

- In general the grid can be $\mathbb{Z}^d$
- CNNs are able to exploit various structures that reduce sample complexity
  - Translation structure (allowing use of filters)
  - Metric on the grid (allows compactly supported filters)

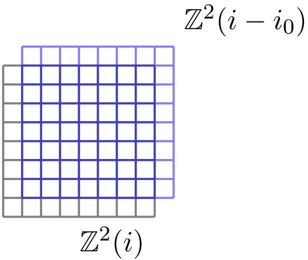
CNNs on Grids

- So far we have defined CNNs on grids
- We model images and feature maps as functions on a rectangular domain

$$f : \mathbb{Z}^2 \to \mathbb{R}^K$$

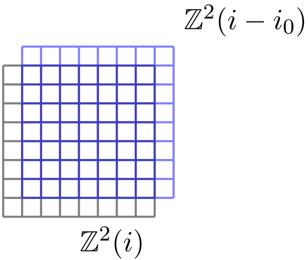
- In general the grid can be $\mathbb{Z}^d$
- CNNs are able to exploit various structures that reduce sample complexity
  - Translation structure (allowing use of filters)
  - Metric on the grid (allows compactly supported filters)
  - Multiscale structure of the grid (allows subsampling)

CNNs on Grids



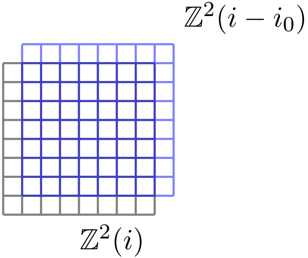
- The translation group acts on $\mathbb{Z}^2$
- We are able to exploit this symmetry of the grid in CNNs

CNNs on Grids



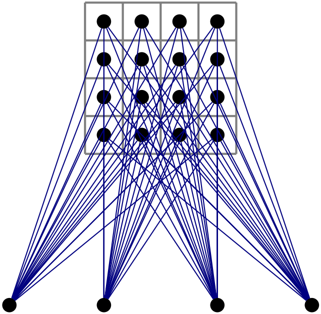
- The translation group acts on $\mathbb{Z}^2$

CNNs on Grids



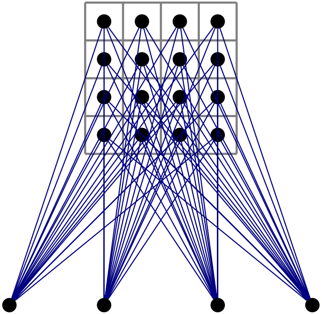
- The translation group acts on $\mathbb{Z}^2$
- We are able to exploit this symmetry of the grid in CNNs

CNNs on Grids



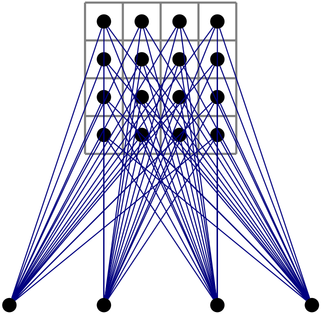
- If we have n input pixels, a fully connected network with m outputs has nm parameters, roughly $O(n^2)$

CNNs on Grids



- If we have n input pixels, a fully connected network with m outputs has nm parameters, roughly $O(n^2)$
- With k filters, each with support S we have $O(kS)$ (independent of n)

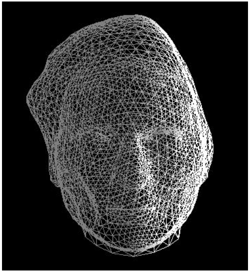
CNNs on Grids



- If we have n input pixels, a fully connected network with m outputs has nm parameters, roughly $O(n^2)$
- With k filters, each with support S we have $O(kS)$ (independent of n)
- Using multiscale nature, we can pool, and reduce the number of parameters further

Data on Irregular Domains

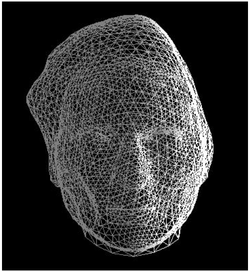
- Often we can have *structured* data defined over coordinates that does not enjoy any of these properties



- Example: 3-D mesh data (each coordinate might be surface tension)
- More: Social network data, protein interaction networks etc.

Data on Irregular Domains

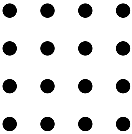
- Often we can have *structured* data defined over coordinates that does not enjoy any of these properties



- Example: 3-D mesh data (each coordinate might be surface tension)
- More: Social network data, protein interaction networks etc.
- In each case we again have n coordinates but which don't live on a regular grid

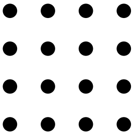
Figure source: Eurocom Face Modeling

Functions on Graphs



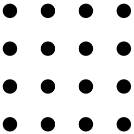
- We can think of a n dimensional image as a function defined on the vertices of a graph $\mathcal{G} = (\Omega, E)$ with $|\Omega| = n$

Functions on Graphs



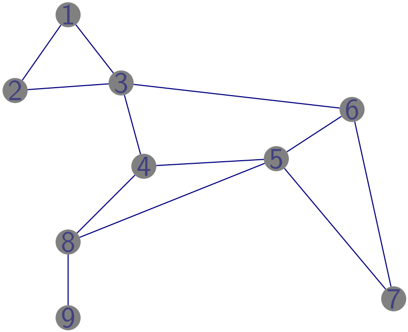
- We can think of a n dimensional image as a function defined on the vertices of a graph $\mathcal{G} = (\Omega, E)$ with $|\Omega| = n$
- $\mathcal{G}$ just happens to be a grid graph with strong local structure which makes CNNs useful

Functions on Graphs



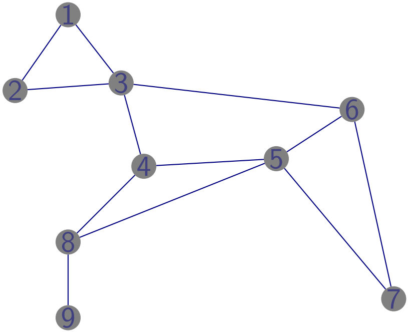
- We can think of a n dimensional image as a function defined on the vertices of a graph $\mathcal{G} = (\Omega, E)$ with $|\Omega| = n$
- $\mathcal{G}$ just happens to be a grid graph with strong local structure which makes CNNs useful
- In general we can have signals defined over a general graph:

Functions on Graphs



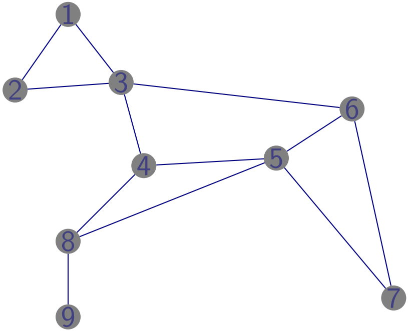
- Ω is the vertex set (input coordinates), $W_{i,j}$ the similarity between any two coordinates i and j

Functions on Graphs



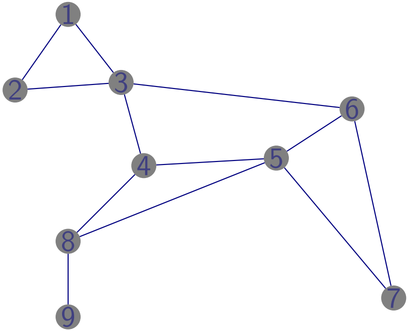
- Ω is the vertex set (input coordinates), $W_{i,j}$ the similarity between any two coordinates i and j
- Note: $W_{i,j}$ is similarity between coordinates, not datapoints

Functions on Graphs



- If the underlying graph structure is known, $W_{i,j}$ will be available

Functions on Graphs



- If the underlying graph structure is known, $W_{i,j}$ will be available
- If unknown: Need to estimate it from training data

Spatial Construction

Locally Connected Networks

Spatial Construction

- So we replace a grid by a general graph $\mathcal{G} = (\Omega, E)$

Spatial Construction

- So we replace a grid by a general graph $\mathcal{G} = (\Omega, E)$
- The notion of locality can be generalized easily via W

Spatial Construction

- So we replace a grid by a general graph $\mathcal{G} = (\Omega, E)$
- The notion of locality can be generalized easily via W
- For given W and threshold δ, we have neighborhoods:

Spatial Construction

- So we replace a grid by a general graph $\mathcal{G} = (\Omega, E)$
- The notion of locality can be generalized easily via W
- For given W and threshold δ, we have neighborhoods:

$$N_\delta(j) = \{i \in \Omega : W_{i,j} > \delta\}$$

Spatial Construction

- So we replace a grid by a general graph $\mathcal{G} = (\Omega, E)$
- The notion of locality can be generalized easily via W
- For given W and threshold δ, we have neighborhoods:

$$N_\delta(j) = \{i \in \Omega : W_{i,j} > \delta\}$$

- Can have filters with receptive fields given by these neighborhoods

Spatial Construction

- So we replace a grid by a general graph $\mathcal{G} = (\Omega, E)$
- The notion of locality can be generalized easily via W
- For given W and threshold δ, we have neighborhoods:

$$N_\delta(j) = \{i \in \Omega : W_{i,j} > \delta\}$$

- Can have filters with receptive fields given by these neighborhoods
- Number of parameters: $O(Sn)$ (S is average neighbhorhood size)

Spatial Construction

- To mimic subsampling and pooling, we can do a multiscale clustering of the graph (K scales)

Spatial Construction

- To mimic subsampling and pooling, we can do a multiscale clustering of the graph (K scales)
- Set $\Omega_0 = \Omega$, at each level $k = 1, \ldots, K$ define Ω_k and Ω_{k-1}

Spatial Construction

- To mimic subsampling and pooling, we can do a multiscale clustering of the graph (K scales)
- Set $\Omega_0 = \Omega$, at each level $k = 1, \ldots, K$ define Ω_k and Ω_{k-1}
- Ω_k is a partition of Ω_{k-1} in d_k clusters

Spatial Construction

- To mimic subsampling and pooling, we can do a multiscale clustering of the graph (K scales)
- Set $\Omega_0 = \Omega$, at each level $k = 1, \ldots, K$ define Ω_k and Ω_{k-1}
- Ω_k is a partition of Ω_{k-1} in d_k clusters
- Around every element of Ω_{k-1}, we can define the neighborhood

Spatial Construction

- To mimic subsampling and pooling, we can do a multiscale clustering of the graph (K scales)
- Set $\Omega_0 = \Omega$, at each level $k = 1, \ldots, K$ define Ω_k and Ω_{k-1}
- Ω_k is a partition of Ω_{k-1} in d_k clusters
- Around every element of Ω_{k-1}, we can define the neighborhood

$$N_k = \{N_{k,i} : i = 1 \ldots d_{k-1}\}$$

Spatial Construction

- To mimic subsampling and pooling, we can do a multiscale clustering of the graph (K scales)
- Set $\Omega_0 = \Omega$, at each level $k = 1, \ldots, K$ define Ω_k and Ω_{k-1}
- Ω_k is a partition of Ω_{k-1} in d_k clusters
- Around every element of Ω_{k-1}, we can define the neighborhood

$$N_k = \{N_{k,i} : i = 1 \ldots d_{k-1}\}$$

Defining the Network

- Let number of filters at layer be given by f_k

Defining the Network

- Let number of filters at layer be given by f_k
- Every layer will transform a f_{k-1} dimensional signal, indexed by Ω_{k-1} into a f_k indexed by Ω_k

Defining the Network

- Let number of filters at layer be given by f_k
- Every layer will transform a f_{k-1} dimensional signal, indexed by Ω_{k-1} into a f_k indexed by Ω_k
- If $x_k = (x_{k,i}; i = 1 \ldots f_{k-1})$ is the $d_{k-1} \times f_{k-1}$ dim input to layer k, the output is defined as:

Defining the Network

- Let number of filters at layer be given by f_k
- Every layer will transform a f_{k-1} dimensional signal, indexed by Ω_{k-1} into a f_k indexed by Ω_k
- If $x_k = (x_{k,i}; i = 1 \dots f_{k-1})$ is the $d_{k-1} \times f_{k-1}$ dim input to layer k, the output is defined as:

$$x_{k+1,j} = L_k h\left(\sum_{i=1}^{f_{k-1}} F_{k,i,j} x_{k,i}\right) \text{with } j = 1 \dots f_k$$

Defining the Network

- Let number of filters at layer be given by f_k
- Every layer will transform a f_{k-1} dimensional signal, indexed by Ω_{k-1} into a f_k indexed by Ω_k
- If $x_k = (x_{k,i}; i = 1 \dots f_{k-1})$ is the $d_{k-1} \times f_{k-1}$ dim input to layer k, the output is defined as:

$$x_{k+1,j} = L_k h\left(\sum_{i=1}^{f_{k-1}} F_{k,i,j} x_{k,i}\right) \text{with } j = 1 \dots f_k$$

- $F_{k,i,j}$ is a $d_{k-1} \times d_{k-1}$ sparse matrix with $\mathcal{N}_k$ indicated by zeros

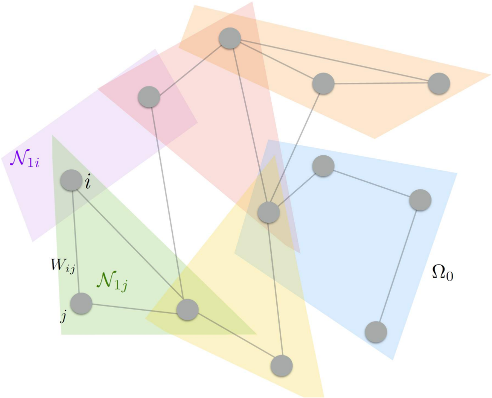
Defining the Network

- Let number of filters at layer be given by f_k
- Every layer will transform a f_{k-1} dimensional signal, indexed by Ω_{k-1} into a f_k indexed by Ω_k
- If $x_k = (x_{k,i}; i = 1 \dots f_{k-1})$ is the $d_{k-1} \times f_{k-1}$ dim input to layer k, the output is defined as:

$$x_{k+1,j} = L_k h\left(\sum_{i=1}^{f_{k-1}} F_{k,i,j} x_{k,i}\right) \text{with } j = 1 \dots f_k$$

- $F_{k,i,j}$ is a $d_{k-1} \times d_{k-1}$ sparse matrix with $\mathcal{N}_k$ indicated by zeros
- h is the non-linearity and L_k is the pooling operation

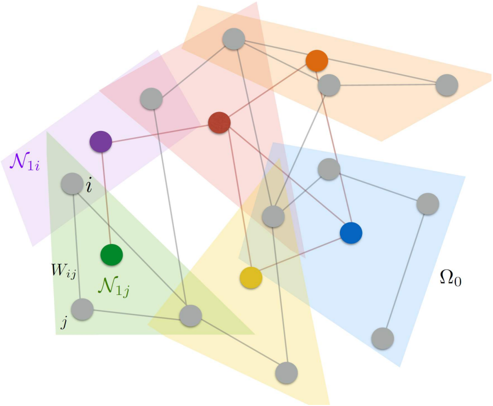
Locally Connected Networks: In Pictures



- Level 1 clustering

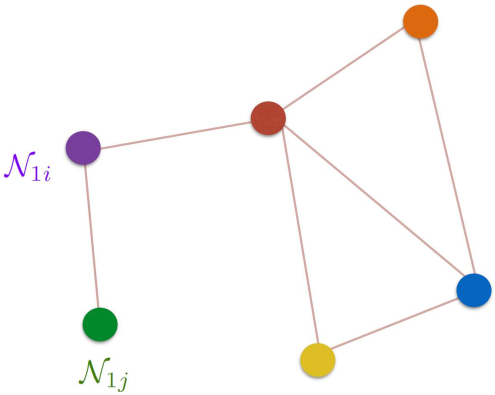
This and next few illustrations are by Joan Bruna

Locally Connected Networks: In Pictures



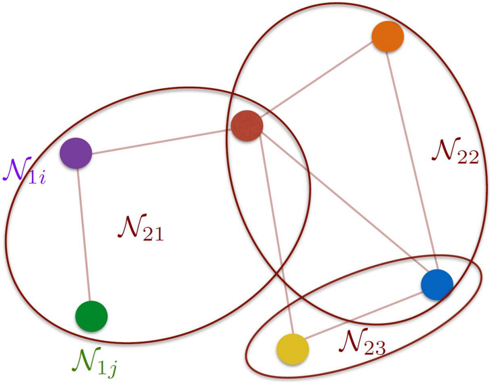
- Pooling to get Ω_1

Locally Connected Networks: In Pictures



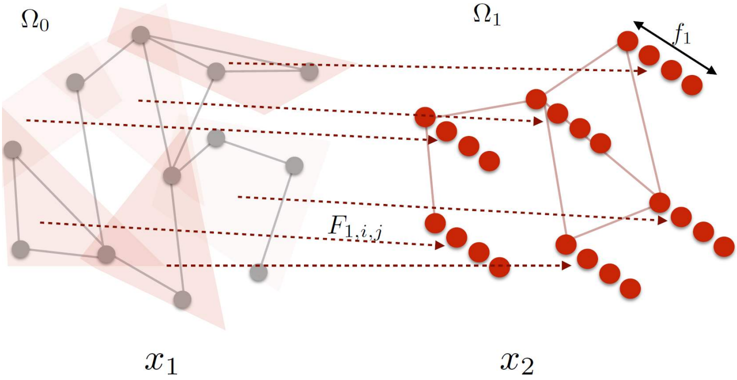
- Pooling to get Ω_1

Locally Connected Networks: In Pictures



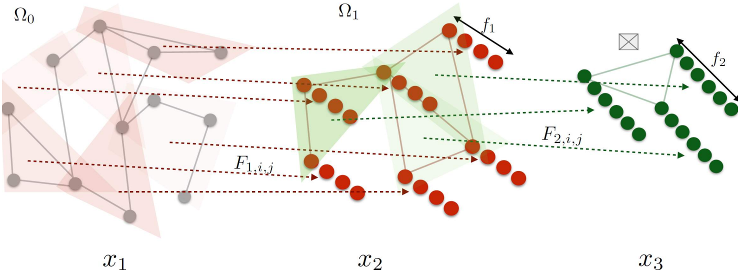
- Level 2 clustering

Locally Connected Networks: In Pictures



- Multiple Feature maps: Level 1

Locally Connected Networks: In Pictures



- Multiple Feature maps: Level 2

Spectral Construction

Spectral Networks

Quick Digression: The Graph Laplacian

Spectral Networks

- Again consider $W \in \mathbb{R}^{d\times d}$, the weighted adjacency matrix for $\mathcal{G} = (\Omega, E)$

Spectral Networks

- Again consider $W \in \mathbb{R}^{d \times d}$, the weighted adjacency matrix for $\mathcal{G} = (\Omega, E)$
- We consider the following definition of the Graph Laplacian:

$$L = I - D^{-1/2} W D^{-1/2}$$

Spectral Networks

- Again consider $W \in \mathbb{R}^{d\times d}$, the weighted adjacency matrix for $\mathcal{G} = (\Omega, E)$
- We consider the following definition of the Graph Laplacian:

$$L = I - D^{-1/2} W D^{-1/2}$$

- D is a diagonal matrix; the degree matrix with $D_{i,i} = \sum_i W_{i,:}$

Spectral Networks

- Again consider $W \in \mathbb{R}^{d \times d}$, the weighted adjacency matrix for $\mathcal{G} = (\Omega, E)$
- We consider the following definition of the Graph Laplacian:

$$L = I - D^{-1/2} W D^{-1/2}$$

- D is a diagonal matrix; the degree matrix with $D_{i,i} = \sum_i W_{i,:}$
- Let $U = [u_1, \ldots, u_d]$ be the eigenvectors of L

Graph Convolution in Frequency Domain

- Define convolution of input signal x with filter g on $\mathcal{G}$ as:

$$x *_{\mathcal{G}} g = U^T (Ux \odot Ug)$$

Graph Convolution in Frequency Domain

- Define convolution of input signal x with filter g on $\mathcal{G}$ as:

$$x *_{\mathcal{G}} g = U^T(Ux \odot Ug)$$

- Learning filters on a graph $\implies$ learning spectral weights:

$$x *_{\mathcal{G}} g = U^T(diag(w_g)Ux) \text{with } w_g = (w_1, \dots, w_d)$$

Local Filters

- Notice that g has support over all vertices

Local Filters

- Notice that g has support over all vertices
- But we want filters that are local

Local Filters

- Notice that g has support over all vertices
- But we want filters that are local
- Observation: Smoothness in frequency domain $\implies$ spatial decay

Local Filters

- Notice that g has support over all vertices
- But we want filters that are local
- Observation: Smoothness in frequency domain $\implies$ spatial decay
- Solution: Consider a smoothing kernel $\mathcal{K} \in \mathbb{R}^{d \times d_0}$ and search for multipliers:

$$w_g = \mathcal{K} \tilde{w}_g$$

Graph Convolution Layer

- **Forward Pass:**
  - For input x, compute interpolated weights $w_{f'f} = \mathcal{K}\tilde{w}_{f'f}$

Graph Convolution Layer

- **Forward Pass:**
  - For input x, compute interpolated weights $w_{f'f} = \mathcal{K}\tilde{w}_{f'f}$
  - Compute the output: $y_{sf'} = U^T(\sum_f Ux_{sf} \odot w_{f'f})$
- **Backward Pass:**
  - Compute gradient w.r.t input Δx_{sf}

Graph Convolution Layer

- **Forward Pass:**
  - For input x, compute interpolated weights $w_{f'f} = \mathcal{K}\tilde{w}_{f'f}$
  - Compute the output: $y_{sf'} = U^T(\sum_f U x_{sf} \odot w_{f'f})$
- **Backward Pass:**
  - Compute gradient w.r.t input Δx_{sf}
  - Compute gradient w.r.t interpolated weights $\Delta w_{f'f}$

Graph Convolution Layer

- **Forward Pass:**
  - For input x, compute interpolated weights $w_{f'f} = \mathcal{K}\tilde{w}_{f'f}$
  - Compute the output: $y_{sf'} = U^T(\sum_f Ux_{sf} \odot w_{f'f})$
- **Backward Pass:**
  - Compute gradient w.r.t input Δx_{sf}
  - Compute gradient w.r.t interpolated weights $\Delta w_{f'f}$
  - Compute gradient w.r.t weight $\Delta \tilde{w}_{f'f} = \mathcal{K}^T \Delta w_{f'f}$

What if Graph Structure is unknown?

- Estimate it from data:
- Method 1: **Unsupervised**
  - Given dataset $X \in \mathbb{R}^{N \times d}$, compute distance $d(i,j)$ between features:

$$d(i,j) = \|X_i - X_j\|_2^2$$

What if Graph Structure is unknown?

- Estimate it from data:
- Method 1: **Unsupervised**
  - Given dataset $X \in \mathbb{R}^{N \times d}$, compute distance $d(i,j)$ between features:

$$d(i,j) = \|X_i - X_j\|_2^2$$

  - Then compute $W_{i,j} = \exp^{-\frac{d(i,j)}{\sigma^2}}$

What if Graph Structure is unknown?

- Estimate it from data:
- Method 2: **Supervised**
  - Given dataset $X \in \mathbb{R}^{N \times d}$ and labels $y \in \{1, \ldots, C\}^L$, train a fully connected MLP with K layers, with weights $W_1, \ldots, W_K$

What if Graph Structure is unknown?

- Estimate it from data:
- Method 2: **Supervised**
  - Given dataset $X \in \mathbb{R}^{N \times d}$ and labels $y \in \{1, \ldots, C\}^L$, train a fully connected MLP with K layers, with weights $W_1, \ldots, W_K$
  - Pass data through network, extract K layer features $W_K \in \mathbb{R}^{N \times m_k}$, then compute:

$$d(i, j) = \|W_{ki} - W_{kj}\|_2^2$$

What if Graph Structure is unknown?

- Estimate it from data:
- Method 2: **Supervised**
  - Given dataset $X \in \mathbb{R}^{N \times d}$ and labels $y \in \{1, \ldots, C\}^L$, train a fully connected MLP with K layers, with weights $W_1, \ldots, W_K$
  - Pass data through network, extract K layer features $W_K \in \mathbb{R}^{N \times m_k}$, then compute:

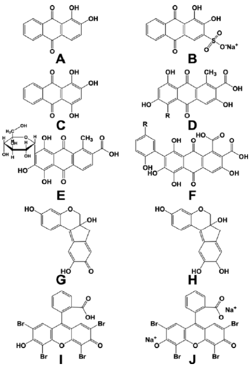
$$d(i, j) = \|W_{ki} - W_{kj}\|_2^2$$

  - Use Gaussian kernel as before to get $W_{i,j}$

Scenario 2

Learning Embeddings of Graphs

Example Task: Regression



- Input: Organic Compounds (graphs)
- Output: Boiling point

Graph Embedding: Simple Algorithm

Algorithm 1 Generation of embedding

Require: $G = (V, E)$, radius δ, Hidden Weights: $H_1^1, \ldots, H_l^\delta$, Output Weights: $W_1, \ldots, W_\delta$

Initialize: Embedding $\phi \leftarrow 0$ **Initialize:** For every vertex $\mathbf{r}_v \leftarrow \Psi(v)$ (local vertex features)

```
for all L = 1 to δ (for every layer) do
    for each vertex v in graph do
        r_1, ..., r_N = neighbors(v)
        v ← r_v + Σ_{i=1}^{N} r_i
        r_v ← σ(v H_L^N)
        i ← softmax(r_v W_L)
        Update: φ ← φ + i
    end for
end for
Output embedding φ
```