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
  • Scenario 2: Each data point is itself a graph (Example regression task: Molecules as input, boiling points as output)
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 the underlying graph

- **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 $G$, find an embedding $\phi: G \rightarrow \mathbb{R}^p$
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 $\phi : \mathcal{G} \rightarrow \mathbb{R}^p$
Scenario 1

CNNs on data in irregular domains
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
  \[ f : \mathbb{Z}^2 \rightarrow \mathbb{R}^K \]
- In general the grid can be \( \mathbb{Z}^d \)
- CNNs are able to exploit various structures that reduce sample complexity
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)
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)
- 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
  \[ f : \mathbb{Z}^2 \rightarrow \mathbb{R}^K \]
- 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)
  - Metric on the grid (allows compactly supported filters)
  - Multiscale structure of the grid (allows subsampling)
The translation group acts on $\mathbb{Z}^2$

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

We are able to exploit this symmetry of the grid in CNNs
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.
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 $n$).
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.
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 an $n$ dimensional image as a function defined on the vertices of a graph $\mathcal{G} = (\Omega, E)$ with $|\Omega| = n$.
We can think of an $n$ 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 \( G = (\Omega, E) \) with \( |\Omega| = n \).

\( G \) 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:
$\Omega$ is the vertex set (input coordinates), $W_{i,j}$ the similarity between any two coordinates $i$ and $j$
Functions on Graphs

- $\Omega$ 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
If the underlying graph structure is known, $W_{i,j}$ will be available.
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 $\delta$, 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: $O(Sn)$ (where $S$ is average neighborhood size).
Spatial Construction

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

\[
N_\delta(j) = \{i \in \Omega : W_{i,j} > \delta\}
\]
Spatial Construction

So we replace a grid by a general graph \( G = (\Omega, E) \)

The notion of locality can be generalized easily via \( W \)

For given \( W \) and threshold \( \delta \), 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 $\delta$, 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 neighborhood size)
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 $\Omega_k$ and $\Omega_{k-1}$
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 $\Omega_k$ and $\Omega_{k-1}$
- $\Omega_k$ is a partition of $\Omega_{k-1}$ 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 $\Omega_k$ and $\Omega_{k-1}$
- $\Omega_k$ is a partition of $\Omega_{k-1}$ in $d_k$ clusters
- Around every element of $\Omega_{k-1}$, 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 $\Omega_k$ and $\Omega_{k-1}$

$\Omega_k$ is a partition of $\Omega_{k-1}$ in $d_k$ clusters

Around every element of $\Omega_{k-1}$, we can define the neighborhood

$$N_k = \{N_{k,i} : i = 1 \ldots d_{k-1}\}$$
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 \(\Omega_k\) and \(\Omega_{k-1}\) 

\(\Omega_k\) is a partition of \(\Omega_{k-1}\) in \(d_k\) clusters 

Around every element of \(\Omega_{k-1}\), we can define the neighborhood 

\[
N_k = \{N_{k,i} : i = 1 \ldots d_{k-1}\}
\]
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 $\Omega_{k-1}$ into a $f_k$ indexed by $\Omega_k$
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 $\Omega_{k-1}$ into a $f_k$ indexed by $\Omega_k$.
- 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 $\Omega_{k-1}$ into a $f_k$ indexed by $\Omega_k$
- 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:

$$x_{k+1,j} = L_k h \left( \sum_{i=1}^{f_{k-1}} F_{k,i,j} x_{k,i} \right) \text{ with } j = 1 \ldots f_k$$
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 $\Omega_{k-1}$ into a $f_k$ indexed by $\Omega_k$
- 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:

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

- $F_{k,i,j}$ 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 $\Omega_{k-1}$ into a $f_k$ indexed by $\Omega_k$
- 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:

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

- $F_{k,i,j}$ is a $d_{k-1} \times d_{k-1}$ sparse matrix with $N_k$ 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 $\Omega_1$
Locally Connected Networks: In Pictures

Pooling to get $\Omega_1$
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
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:

$$L = I - D^{-1/2}WD^{-1/2}$$
Spectral Networks

- Again consider $W \in \mathbb{R}^{d \times d}$, the weighted adjacency matrix for $G = (\Omega, E)$
- 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 $D_{i,i} = \sum_i W_{i,i}$.
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:

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

- $D$ is a diagonal matrix; the degree matrix with $D_{i,i} = \sum_i W_{i,i}$
- Let $U = [u_1, \ldots, u_d]$ be the eigenvectors of $L$
Define convolution of input signal $x$ with filter $g$ on $G$ as:

$$x \ast_G g = U^T (Ux \odot Ug)$$
Graph Convolution in Frequency Domain

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

$$x \ast g = U^T(Ux \odot Ug)$$

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

$$x \ast g = U^T(diag(w_g)Ux) \text{with } w_g = (w_1, \ldots, w_d)$$
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 $w_{f'f} = K\tilde{w}_{f'f}$
Graph Convolution Layer

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

- **Backward Pass:**
  - Compute gradient w.r.t input $\Delta x_{sf}$
Graph Convolution Layer

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

**Backward Pass:**
- Compute gradient w.r.t input $\Delta x_{sf}$
- Compute gradient w.r.t interpolated weights $\Delta w_{f'f}$
Graph Convolution Layer

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

- **Backward Pass:**
  - Compute gradient w.r.t input $\Delta x_{sf}$
  - Compute gradient w.r.t interpolated weights $\Delta w_{f'f}$
  - Compute gradient w.r.t weight $\Delta \tilde{w}_{f'f} = K^T \Delta w_{f'f}$
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 $W_{i,j} = \exp\left(-\frac{d(i,j)}{\sigma^2}\right)$
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 $\delta$, Hidden Weights: $H_1, \ldots, H_\delta$, Output Weights: $W_1, \ldots, W_\delta$

Initialize: Embedding $\phi \leftarrow 0$

(Initialize: For every vertex $r_v \leftarrow \Psi(v)$ (local vertex features))

1: for all $L = 1$ to $\delta$ (for every layer) do
2: for each vertex $v$ in graph do
3: $r_1, \ldots, r_N =$ neighbors($v$)
4: $v \leftarrow r_v + \sum_{i=1}^{N} r_i$
5: $r_v \leftarrow \sigma(vH_L^N)$
6: $i \leftarrow \text{softmax}(r_v W_L)$
7: Update: $\phi \leftarrow \phi + i$
8: end for
9: end for
10: Output embedding $\phi$