Graph Neural Networks

Scribed by: Aman Kumar

November 24, 2025

1 Traditional ConvNets and High-Dimensional Learning

ConvNets are powerful architectures designed to solve high-dimensional learning problems. To understand their necessity, we must first address the challenges of high-dimensional data spaces.

1.1 The Curse of Dimensionality

Consider an image of resolution 1024×1024 pixels. This image can be viewed as a single point in a space of $1,000,000$ dimensions:

$$\dim(\text{image}) = 1024 \times 1024 \approx 10^6$$

If we attempt to learn this space using a naive approach with just $N = 10$ samples per dimension, the total number of samples required would be:

$$10^{1,000,000} \text{ points}$$

This number is astronomically high, illustrating the *Curse of Dimensionality*. ConvNets are designed to extract efficient representations from such high-dimensional data without requiring exhaustive sampling of the space.

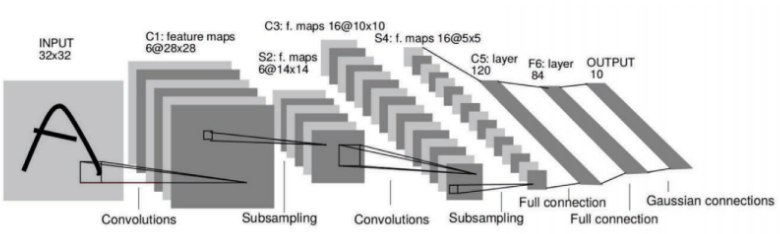

Figure 1: ConvNets extract representation of high-dimensional image data.

2 Main Assumptions of ConvNets

ConvNets function effectively because they rely on specific assumptions about the data structure.

2.1 1. Compositionality

Data such as images, videos, and speech is **compositional**. It is formed of patterns that adhere to three key principles:

- **Local:** A neuron is only connected to adjacent layers (Local Reception Field), not to all neurons in the previous layer.

- **Stationary:** Patterns are similar and shared across the domain (e.g., a specific texture or object part appearing in different locations).
- **Hierarchical:** Low-level features combine to form medium-level features, which subsequently combine to form high-level semantic representations.

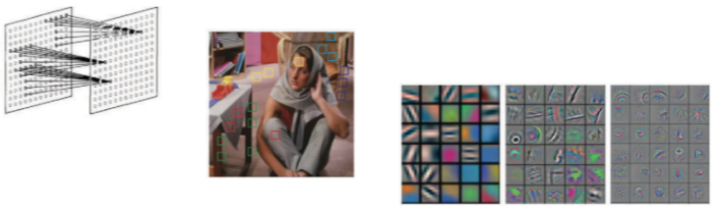

Figure 2: Data is compositional: Local, Stationary, and Hierarchical.

2.2 2. Leveraging Structure

ConvNets leverage this compositionality to extract features which are then fed into downstream tasks such as classifiers or recommendation system.

3 Data Domains: Grid vs. Graph

3.1 Euclidean (Grid) Domain

- **Images/Video:** Lie on 2D or 3D Euclidean grids. Each pixel is a regular vector (e.g., RGB).
- **Sequences (Speech/Text):** Lie on a 1D Euclidean domain.

These domains possess strong, regular spatial structures, allowing operations to be fast and mathematically well-defined.

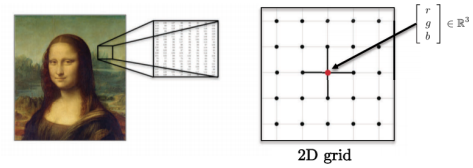

Figure 3: Images lie on regular Euclidean grids.

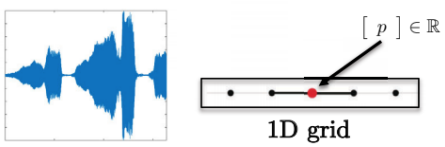

Figure 4: Speech typically lie on regular Euclidean grids.

3.2 Graph Domain

Many real-world datasets do not form a grid and are best captured by graph representations.

3.2.1 Motivational Examples

1. **Social Networks:** Nodes represent users, edges represent connections. Pair-wise connections do not form a grid.
2. **Brain Connectivity:** The brain is composed of Regions of Interest (ROIs). The adjacency matrix represents the connection strength between ROIs, useful for predicting neural genetic diseases.
3. **Quantum Chemistry:** Atoms are nodes, bonds are edges. Features include atom charge and bond type.

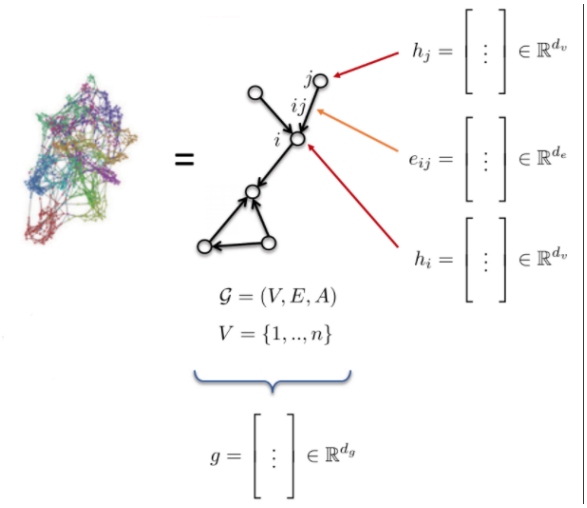

Figure 5: A Graph G with vertices V and edges E.

3.2.2 Graph Definition

A Graph G is defined by:

- Vertices V, Edges E, Adjacency Matrix A.
- **Node Features:** h_i (e.g., atom type).
- **Edge Features:** e_{ij} (e.g., bond type).
- **Graph Features:** g (e.g., molecule energy).

4 Convolution in Traditional ConvNets

4.1 Definition

Abstractly, we define convolution at layer ℓ as:

$$h^{\ell+1} = w^{\ell} * h^{\ell} \tag{1}$$

Where $h^{\ell+1}$ is the feature map at the next layer, and w^{ℓ} is the kernel (usually small, e.g., 3×3).

Mathematically, this is implemented as **Template Matching** (or correlation):

$$h_i^{\ell+1} = (w^\ell * h^\ell)_i \tag{2}$$

$$= \sum_{j \in \Omega} \langle w_j^\ell, h_{i-j}^\ell \rangle \quad \text{(Standard Def)} \tag{3}$$

$$= \sum_{j \in \mathcal{N}_i} \langle w_j^\ell, h_{ij}^\ell \rangle \quad \text{(Local Neighborhood)} \tag{4}$$

Because the kernel is small, we sum over the neighborhood $\mathcal{N}_i$ rather than the whole domain Ω. This results in a complexity of $O(n)$, which can be parallelized on GPUs.

4.2 Why Template Matching Fails on Graphs

If we try to extend this definition to graphs, we encounter two main issues:

1. **No Node Ordering:** In an image, the "top-right" pixel is clearly defined. In a graph, there is no notion of direction (up, down, left, right). Therefore, matching a specific weight w_j to a neighbor is ambiguous.

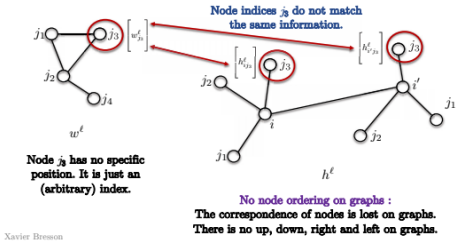

Figure 6: No node ordering in a graph.

2. **Variable Neighborhood Sizes:** A standard CNN kernel has a fixed size (e.g., 9 neighbors in a 3×3 grid). In a graph, one node may have 2 neighbors while another has 100.

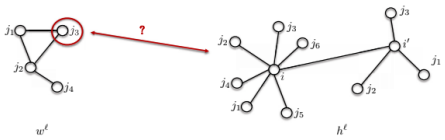

Figure 7: Different neighborhood sizes in a graph..

5 Spectral Graph ConvNets

To resolve the issues of spatial convolution on graphs, we turn to the **Convolution Theorem**.

5.1 The Convolution Theorem

The theorem states that the Fourier transform of the convolution of two functions is the pointwise product of their Fourier transforms:

$$\mathcal{F}(w * h) = \mathcal{F}(w) \odot \mathcal{F}(h) \implies w * h = \mathcal{F}^{-1}\big(\mathcal{F}(w) \odot \mathcal{F}(h)\big) \tag{5}$$

To apply this to graphs, we need to define the Fourier Transform for graphs.

5.2 Step 1: The Graph Laplacian

The core operator in spectral graph theory is the Normalized Laplacian Δ:

$$\Delta = I - D^{-1/2} A D^{-1/2} \tag{6}$$

Note Matrix A is the **adjacency matrix**, and the Δ is the **Laplacian**, which equals to the identity minus the adjacency matrix normalized by Matrix D. D is a diagonal matrix, and each element on the diagonal is the degree of the node. This is called the **normalized Laplacian**, or Laplacian by default in this context.

The Laplacian is interpreted as the measurement of **smoothness of graph**, in other words, the difference between the local value h_i and its neighborhood average value of h_j's. d_i is the degree of node i, and $\mathcal{N}_i$ is the set of all the neighbors of node i.

$$(\Delta h)_i = h_i - \frac{1}{\sqrt{d_i}} \sum_{j \in \mathcal{N}_i} A_{ij} \frac{h_j}{\sqrt{d_j}} \tag{7}$$

5.3 Step 2: Fourier Functions (Eigen-decomposition)

The following is the eigen-decomposition of graph Laplacian:

$$\Delta_{n \times n} = \Phi \Lambda \Phi^\top$$

The Laplacian is factorized into three matrices, $\Phi^\top$, Λ, Φ.

Φ is defined below; it contains column vectors, or Lap eigenvectors, ϕ_1 to ϕ_n, each of size $n \times 1$, and those are also called **Fourier functions**. These Fourier functions form an **orthonormal basis**.

$$\Phi = [\phi_1, \ldots, \phi_n] \quad \text{and} \quad \Phi^\top \Phi = I, \quad \langle \phi_k, \phi_{k'} \rangle = \delta_{kk'}$$

Λ is a diagonal matrix with Laplacian eigenvalues, and on the diagonal are λ_1 to λ_n. From the normalized Laplacian, these values are typically among the range from 0 to 2. Those are also known as the **Spectrum of a graph**.

$$\Lambda_{n \times n} = \text{diag}(\lambda_1, \ldots, \lambda_n) \quad \text{and} \quad 0 \leq \lambda_1 \leq \cdots \leq \lambda_n = \lambda_{\max} \leq 2$$

Below is an example of checking the eigenvalue decomposition. The Laplacian matrix Δ is multiplied by an eigenvector ϕ_k, the shape of the result is $n \times 1$, and the result is $\lambda_k \phi_k$.

$$\Delta \phi_k = \lambda_k \phi_k, \quad k = 1, \ldots, n$$

6 The Graph Fourier Transform

The Graph Fourier Transform is essential for defining convolution in the spectral domain. It projects the graph signal onto the Laplacian eigenvectors (Fourier functions).

6.1 Fourier Series

A graph signal h can be decomposed using the Fourier functions (Laplacian eigenvectors, ϕ_k). This is the graph equivalent of a Fourier series expansion:

$$h = \sum_{k=1}^{n} \hat{h}_k \phi_k = \sum_{k=1}^{n} (\phi_k^\top h) \phi_k = \Phi \hat{h} = \mathcal{F}^{-1}(\hat{h})$$

The coefficient of the Fourier series for mode k, $\hat{h}_k$, is a scalar obtained by projecting the function h onto the Fourier function ϕ_k:

$$\hat{h}_k = \langle \phi_k, h \rangle = \phi_k^\top h$$

6.2 Fourier Transform and Inverse

The Fourier Transform is a linear operation that takes the graph signal h and projects it onto the orthonormal basis Φ to yield the coefficients $\hat{h}$:

- **Fourier Transform / Coefficients of Fourier Series**

$$\mathcal{F}(h)_{n\times 1} = \Phi^\top h = \hat{h}$$

The Fourier Transform is the projection of the graph signal h onto the eigenvectors Φ:

$$\mathcal{F}(h) = \hat{h} = \Phi^T h$$

- **Inverse Fourier Transform**

$$\mathcal{F}^{-1}(\hat{h})_{n\times 1} = \Phi\hat{h} = \Phi\Phi^\top h = h \quad \text{as } \mathcal{F}^{-1} \circ \mathcal{F} = \Phi\Phi^\top = I$$

The Inverse Transform reconstructs the signal:

$$\mathcal{F}^{-1}(\hat{h}) = h = \Phi\hat{h}$$

6.3 Step 4: Spectral Convolution

The foundation for defining convolution on a graph lies in the **Convolution Theorem**: The Fourier transform of the convolution of two functions is the pointwise product of their Fourier transforms, $\mathcal{F}(w * h) = \mathcal{F}(w) \odot \mathcal{F}(h)$.

Let w be the filter and h be the input signal. Using the Graph Fourier Transform ($\mathcal{F}(h) = \Phi^\top h$), the graph convolution ($w *_G h$) is defined as:

$$\begin{aligned}
w *_G h_{n\times 1} &= \mathcal{F}^{-1}\big(\mathcal{F}(w) \odot \mathcal{F}(h)\big) \\
&= \underbrace{\Phi}_{\mathcal{F}^{-1}} \left(\underbrace{\hat{w}_{n\times 1}}_{\mathcal{F}(w)} \odot (\Phi^\top h_{n\times 1}) \right) \\
&= \Phi \left(\hat{w}(\Lambda)_{n\times n} \Phi^\top h_{n\times 1} \right) \\
&= \Phi \hat{w}(\Lambda) \Phi^\top h \\
&= \hat{w}(\underbrace{\Phi\Lambda\Phi^\top}_{\Delta}) h \\
&= \hat{w}(\Delta)_{n\times n} h_{n\times 1}
\end{aligned}$$

The convolution of the two functions w and h on the graph is therefore equivalent to applying the spectral function of w, $\hat{w}$, to the Laplacian, Δ, and multiplying it by the signal h. This is the definition of **Spectral Convolution**:

$$w *_G h = \hat{w}(\Delta)h$$

This definition allows us to perform convolution mathematically rigorously on graphs. However, a key limitation is the computational complexity. The eigen-decomposition ($\Delta = \Phi\Lambda\Phi^\top$) itself is typically $O(n^3)$, and the matrix multiplications involving the dense matrix Φ lead to a complexity of $O(n^2)$ for each convolution layer, which is prohibitively expensive for large graphs where n (the number of nodes) is large.

7 Spectral Graph ConvNets (Spectral GCNs)

In the previous section, we discussed Graph Spectral Theory as a method to define convolution via the Fourier domain. We now explore the evolution of Spectral GCN architectures.

7.1 Vanilla Spectral GCN

The first generation of Spectral GCNs defined the convolution layer such that for a given layer h^l, the activation of the next layer is:

$$h^{l+1} = \eta(w^l * h^l) \tag{8}$$

Here, η is a non-linear activation and w^l is a spatial filter. Using the Convolution Theorem, the RHS is equivalent to $\eta(\hat{w}^l(\Delta)h^l)$. Decomposing the Laplacian $\Delta = \Phi\Lambda\Phi^\top$, we obtain the final activation equation:

$$h^{l+1} = \eta(\Phi\hat{w}^l(\Lambda)\Phi^\top h^l) \tag{9}$$

The objective is to learn the spectral filter $\hat{w}^l(\lambda)$ via backpropagation.

Limitations:

1. **No Spatial Localization:** Filters are not guaranteed to be local in the graph domain.
2. **Parameter Complexity:** Requires learning $O(n)$ parameters per layer $(w(\lambda_1)\dots w(\lambda_n))$.
3. **Computational Complexity:** Backpropagation is $O(n^2)$ because Φ is a dense matrix.

7.2 ChebNets

To resolve the stability issue of LapGCNs, ChebNets replace the monomial basis with **Chebyshev polynomials**, which are orthogonal. We approximate the filter using Chebyshev polynomials $T_k(\tilde{\Delta})$:

$$X_k = 2\tilde{\Delta}X_{k-1} - X_{k-2}, \quad X_0 = h, \quad X_1 = \tilde{\Delta}h \tag{10}$$

where $\tilde{\Delta} = 2\lambda_n^{-1}\Delta - I$ is the rescaled Laplacian. ChebNets provide stable, localized, and linear-time spectral convolutions. However, they are **isotropic** (direction-invariant).

7.3 CayleyNets

CayleyNets use **Cayley rationals** as the orthonormal basis instead of polynomials.

$$\hat{w}(\Delta) = w_0 + 2\,\mathrm{Re}\left\{\sum_{k=0}^{K-1} w_k(z\Delta + i)^k(z\Delta - i)^{-k}\right\} \tag{11}$$

This allows for "spectral zoom," localizing filters in specific frequency bands (useful for detecting communities).

8 Spatial Graph ConvNets

8.1 Template Matching on Graphs

Spatial GCNs rely on the concept of template matching. However, graphs lack a fixed coordinate system (no "top-left" pixel). **Solution:** Use a single template vector w^l shared across all neighbors (Isotropic) or weight edges differently (Anisotropic).

Mathematically, for a feature vector h_j:

$$h_i^{l+1} = \eta\left(\sum_{j\in\mathcal{N}_i}\langle w^l, h_{ij}^l\rangle\right) \tag{12}$$

In matrix form for all nodes:

$$h^{l+1} = \eta(Ah^lW^l) \tag{13}$$

This formulation is node-order invariant and handles varying neighborhood sizes.

Variable Neighborhood Sizes: A standard CNN kernel has a fixed size (e.g., 9 neighbors in a 3×3 grid). In a graph, one node may have 2 neighbors while another has 100.

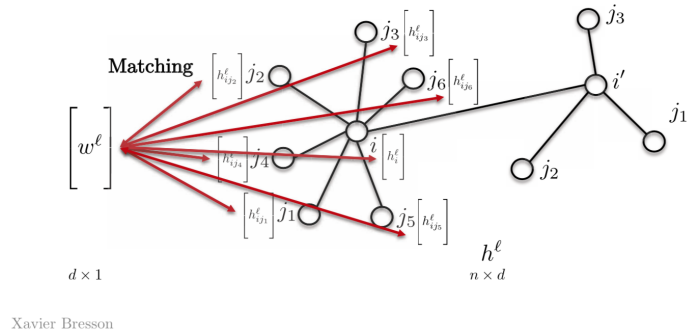

Figure 8: Template Matching using a template vector.

8.2 Isotropic GCNs

Isotropic models treat all neighbors equally (based solely on structure).

8.2.1 Vanilla Spatial GCNs

It has the same definition as before, but we add the Diagonal matrix in the equation, in such a way that we find the mean value of the neighbourhood. The Matrix representation being:

$$h^{l+1} = \eta(D^{-1}Ah^lW^l)$$

where A has the dimensions $n \times n$, h^l has dimensions $n \times d$, and W^l has $d \times d$, which results in an $n \times d$ matrix h^{l+1}.

And the vectorial representation being:

$$h_i^{l+1} = \eta\left(\frac{1}{d_i}\sum_{j \in \mathcal{N}_i} A_{ij}W^l h_j^l\right)$$

where h_i^{l+1} has the dimensions of $d \times 1$.

The vectorial representation is responsible for handling the absence of node ordering, which is invariant of node re-parametrisation. That is, adding on the previous example, if the node has an index of 6 and is changed to 122, this won't change anything in the computation of the activation function of the next layer h^{l+1}.

We can also deal with neighborhoods of different sizes. That is, we can have a neighborhood of 4 nodes or 10 nodes; it wouldn't change anything. We are given the local reception field by design; that is, with Graph Neural Networks we only have to consider the neighbors. We have **weight sharing**; that is, we use the same W^l matrix for all features no matter the position on the graph, which is a Convolution property. This formulation is also independent of the graph size, since all operations are done locally. Since it is an **isotropic** model, the neighbors will have the same W^l matrix.

The activation of the next layer h_i^{l+1} is a function of the activation of the previous layer h_i^l at node i and the neighborhood of i:

$$h_i^{l+1} = f_{\text{GCN}}(h_i^l, \{h_j^l : j \to i\})$$

When we change the function f_{GCN}, we get an entire family of graphs.

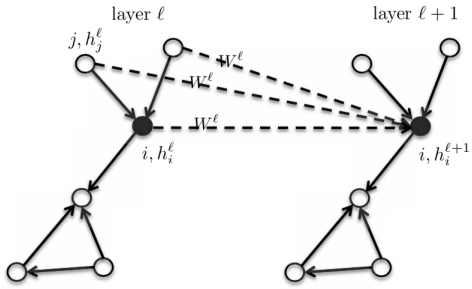

Figure 9: Isotropic model where neighbors share the same template weight.

ChebNets and Vanilla Spatial GCNs The above defined Vanilla Spatial GCN is a simplification of ChebNets. We can truncate the expansion of ChebNet by using the first two Chebyshev functions to end up with:

$$h_i^{l+1} = \eta\left(\frac{1}{\hat{d}_i}\sum_{j\in\mathcal{N}_i}\hat{A}_{ij}W^l h_j^l\right)$$

8.3 Anisotropic GCNs

Standard CNNs have the ability to produce **anisotropic filters** — ones that favour certain directions. This is because the directional structure is based on up, down, left, and right. However, the GCNs described above have no notion of direction, and thus can only produce **isotropic filters**. Anisotropy can be introduced naturally, with **edge features**. For instance, molecules can have single, double, and triple bonds. Graphically, it is introduced by weighting different neighbors differently.

8.3.1 MoNets

MoNets use the degree of the graph to learn the parameters of a **Gaussian Mixture Model (GMM)**.

$$\underset{d\times 1}{h_i^{\ell+1}} = \mathrm{ReLU}\Big(\sum_{k=1}^{K}\sum_{j\in\mathcal{N}_i}\underbrace{e_{ij}^{k,\ell}}_{\text{scalar}}\underbrace{W_1^{k,\ell}}_{d\times d}\underbrace{h_j^\ell}_{d\times 1}\Big)$$

$$\underset{\text{scalar}}{e_{ij}^{k,\ell}} = \exp\Big(-\frac{1}{2}\underbrace{(u_{ij}^\ell-\mu_k^\ell)^T}_{1\times 2}\underbrace{(\Sigma_k^\ell)^{-1}}_{2\times 2}\underbrace{(u_{ij}^\ell-\mu_k^\ell)}_{2\times 1}\Big)$$

$$\underset{2\times 1}{u_{ij}^\ell} = \mathrm{Tanh}\Big(\underbrace{A^\ell}_{2\times 2}\underbrace{(\deg_i^{-1/2},\deg_j^{-1/2})}_{2\times 1}+\underbrace{a^\ell}_{2\times 1}\Big)$$

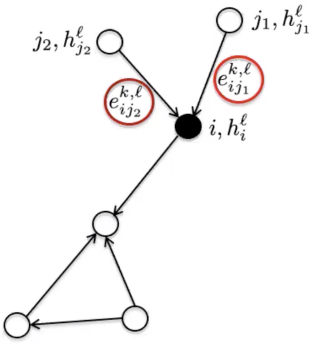

Figure 10: MoNet uses node degree to define filters.

8.3.2 Graph Attention Networks (GAT)

GAT uses the **attention mechanism** to introduce anisotropy in the neighborhood aggregation function.

$$h_i^{\ell+1}_{d\times 1} = \text{Concat}_{k=1}^{K}\Big(\text{ELU}\Big(\sum_{j\in\mathcal{N}_i} e_{ij}^{k,\ell}_{\text{scalar}}\ W_1^{k,\ell}_{\frac{d}{K}\times d}\ h_j^{\ell}_{d\times 1}\Big)\Big)$$

$$e_{ij}^{k,\ell}_{\text{scalar}} = \text{Softmax}_{\mathcal{N}_i}(\hat{e}_{ij}^{k,\ell}) = \frac{\exp(\hat{e}_{ij}^{k,\ell})}{\sum_{j'\in\mathcal{N}_i}\exp(\hat{e}_{ij'}^{k,\ell})}$$

$$\hat{e}_{ij}^{k,\ell} = \text{LeakyReLU}\Big(W_2^{k,\ell}_{1\times\frac{2d}{K}}\ \underbrace{\text{Concat}\big(W_1^{k,\ell}h_i^{\ell},\ W_1^{k,\ell}h_j^{\ell}\big)}_{\frac{2d}{K}\times 1}\Big)$$

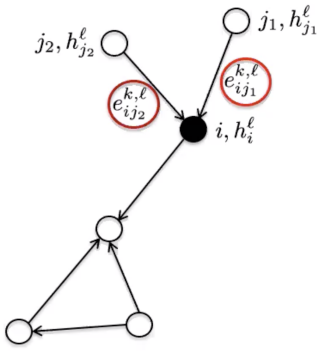

Figure 11: GAT uses attention coefficients α_{ij} to weigh edges.

8.3.3 Gated Graph ConvNets

These use a simple **edge gating mechanism**, which can be seen as a softer attention process compared to the sparse attention mechanism used in GATs.

$$h_i^{\ell+1}_{d\times 1} = h_i^{\ell} + \text{ReLU}\Big(\text{BN}\Big(W_1^{\ell}h_i^{\ell} + \sum_{j\in\mathcal{N}_i} e_{ij}^{\ell}_{d\times 1}\odot W_2^{\ell}h_j^{\ell}\Big)\Big)$$

$$e_{ij}^{\ell}_{d\times 1} = \frac{\sigma(\hat{e}_{ij}^{\ell})}{\sum_{j'\in\mathcal{N}_i}\sigma(\hat{e}_{ij'}^{\ell}) + \varepsilon}$$

$$\hat{e}_{ij}^{\ell}_{d\times 1} = \hat{e}_{ij}^{\ell-1} + \text{ReLU}\Big(\text{BN}\big(V_1^{\ell}h_i^{\ell-1} + V_2^{\ell}h_j^{\ell-1} + V_3^{\ell}\hat{e}_{ij}^{\ell-1}\big)\Big)$$

[1] X Bresson, T Laurent, Residual gated graph convnets, 2017
[2] D Bahdanau, K Cho, Y Bengio, Neural machine translation by jointly learning to align and translate, 2014

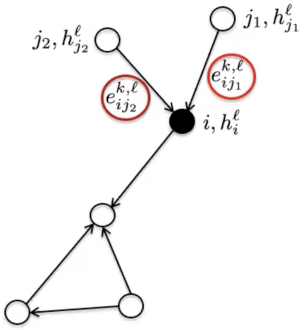

Figure 12: Gated GCNs use soft gating on edges.

8.3.4 Graph Transformers

This is the graph version of the standard transformer, commonly used in NLP. If the graph is **fully connected** (every two nodes share an edge), we recover the definition of a standard transformer. Graphs obtain their structure from sparsity, so the fully connected graph has trivial structure and is essentially a set. Transformers then can be viewed as **Set Neural Networks**, and are in fact the best technique currently to analyze sets/bags of features.

9 Benchmarking GNNs

Recent benchmarks typically evaluate GNNs on four tasks: Graph Classification, Graph Regression, Node Classification, and Edge Classification.

9.1 Graph Regression: Quantum Chemistry

Task: Predict molecular properties (e.g., solubility). **Observation:** Anisotropic GCNs generally outperform Isotropic ones because chemical bonds (edges) have specific types (single, double) that imply directionality/strength.

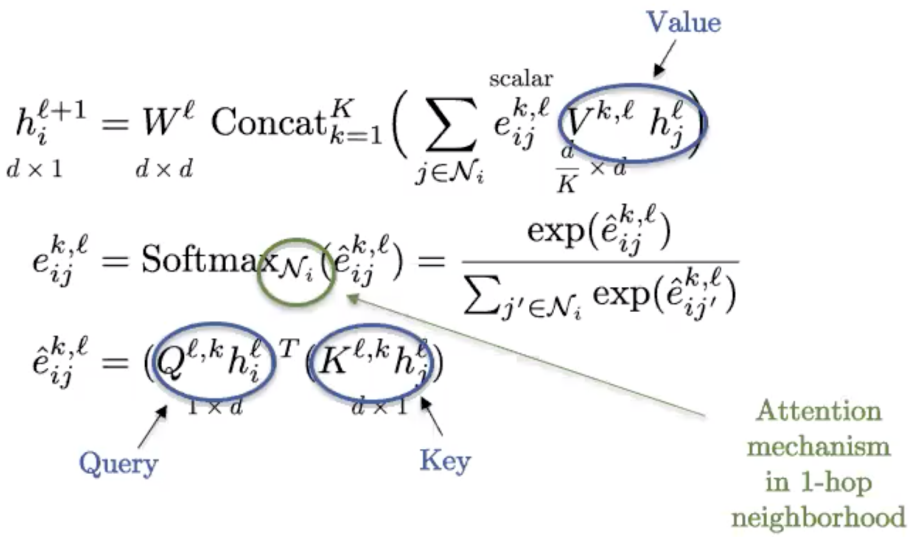

Figure 13: Generalization of the standard transformer to graphs.

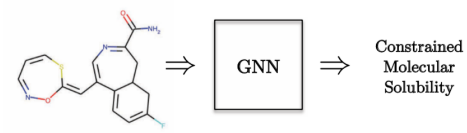

Figure 14: Graph Regression on molecules.

9.2 Edge Classification: TSP

Task: Travelling Salesman Problem (TSP). Classify if an edge belongs to the optimal path. **Observation:** Explicit edge features are crucial here. GatedGCN is often the best performer for such combinatorial optimization tasks.

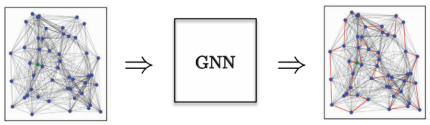

Figure 15: Identifying optimal edges in TSP.

10 Conclusion

GCNs generalize CNNs to non-Euclidean domains.

- **Spectral Approaches:** Evolved from computationally expensive global filters to efficient, localized polynomial approximations (ChebNets).
- **Spatial Approaches:** Evolved from basic averaging (Isotropic) to sophisticated attention-based mechanisms (Anisotropic/GATs).

Modern GCNs are now scalable to large, sparse graphs and are pivotal in fields ranging from drug discovery to recommender systems.

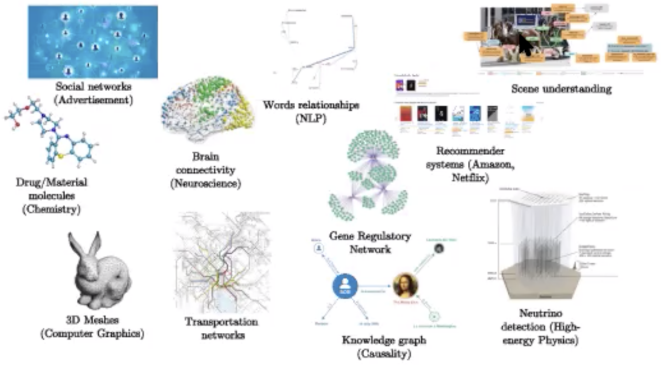

Figure 16: Applications.

References

References

[1] Bruna, J., Zaremba, W., Szlam, A., & LeCun, Y. (2013). *Spectral Networks and Locally Connected Networks on Graphs*. ICLR.

[2] Defferrard, M., Bresson, X., & Vandergheynst, P. (2016). *Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering*. NeurIPS.

[3] Kipf, T. N., & Welling, M. (2017). *Semi-Supervised Classification with Graph Convolutional Networks*. ICLR.

[4] Hamilton, W., Ying, Z., & Leskovec, J. (2017). *Inductive Representation Learning on Large Graphs*. NeurIPS.

[5] Veličković, P., et al. (2018). *Graph Attention Networks*. ICLR.

[6] Xu, K., et al. (2019). *How Powerful are Graph Neural Networks?*. ICLR.

[7] Dwivedi, V. P., et al. (2020). *Benchmarking Graph Neural Networks*. arXiv preprint arXiv:2003.00982.