Lecture 3: Introduction to Graph Neural Networks (GNNs)

Lecture Overview:

- **Topic**: Introduction to Graph Neural Networks (GNNs) and deep learning for graphs.
- **Focus**: Node embeddings, shallow and deep encoders, limitations of simple models, and how GNNs generalize.
- **Structure**: Transition from shallow models to deep encoders like GNNs, covering node representation, loss functions, and practical applications.

---

1. Node Embeddings and Shallow Encoders:

1. **Node Embeddings**:
    - The goal is to transform nodes of a graph into a low-dimensional space (embedding space) that retains structural properties.
    - This transformation, called an **encoder**, maps each node into this space.
2. **Shallow Encoders**:
    - A shallow encoder performs a direct embedding lookup for each node.
    - The embedding matrix has dimensions (N,D) where N is the number of nodes and D is the embedding dimension.
3. **Limitations**:
    - **Linear Parameter Growth**: The number of parameters scales linearly with the graph size O(|V|d) parameters required.
    - **Transductive**: These embeddings cannot generalize to new nodes or unseen graphs since the encoder is specific to a fixed set of nodes.
    - **Don't Incorporate Node features:** usually node features can be helpful.

In conclusion, we see that we desire a way of getting embeddings such that similar nodes in a graph should have similar embeddings. Whereas simple lookup tables can in theory achieve this, in reality that are quite a subpar way of going about this. What if we had an encoder that had multiple layers of non-linear transformations based on graph structure?

---

2. Deep Encoders: Graph Neural Networks (GNNs)

Our current ML toolbox is designed for simple sequences and grids, but our networks are much more complex. For example, there is no fixed node ordering or reference point and data is dynamic and has multimodal features.

1. **We want to solve:**
    - **Node Classification:** prediction of a given node
    - **Link Prediction:** predict if two nodes are linked

- **Community Detection:** identify densely linked cluster of nodes
- **Network Similarity:** predict how similar two networks or subnetworks are

2. **Introduction to Deep Learning:**
   - Loss Function: min L(y, f(x)) where f can be a linear layer, MLP, GNN, etc
   - Minibatches: updates happen in minibatches
   - Forward propagation: Computing L given x
   - Background propagation: Obtaining gradient of L using chain rule
   - Stochastic Gradient Descent: optimizing L for weights over many iterations
3. **Introduction to GNNs**:
   - GNNs are a class of deep learning models designed to process graph-structured data.
   - They extend shallow models by learning how to aggregate information across nodes and their neighbors using a deep neural network structure.
   - Setup: V is vertex set, A is adjacency matrix, X is matrix of node features of size |V| x m, v is a node in V, N(v) is the set of neighbors
4. **Message Passing Neural Networks (MPNN)**:
   - GNNs often implement a form of **message passing**, where nodes receive and aggregate information from their neighbors.
   - Each node updates its representation through multiple layers of non-linear transformations, which depend on the graph structure.
5. **Graph Convolutional Networks (GCNs)**:
   - GCNs are a specific type of GNN where the information aggregation follows a convolutional approach.
   - They aggregate information from neighboring nodes to generate new embeddings for each node at each layer.
   - **Neighborhood Aggregation**: At each layer, node embeddings are updated based on the features of neighboring nodes and the node itself.

---

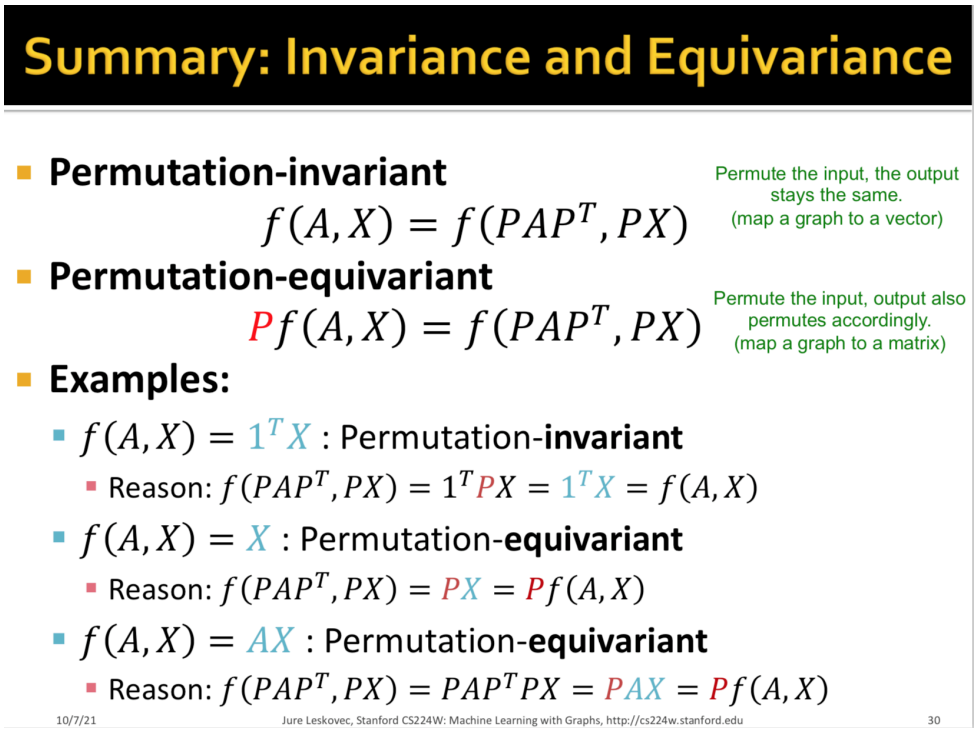
3. Technical Details: Permutation Invariance and Equivariance

Why can't we just use CNNs or feed the adjacency matrix and features into a Neural network?

1. **Permutation Invariance**:
   - Graphs lack a canonical node ordering; thus, the output of a function applied to a graph should not depend on the ordering of nodes.
   - **Definition**: For any graph function f: Rf: $R^{|V| \, x \, |V|} x \, R^{|V| \, x \, m} \; -> \; R^{d}$, f is permutation invariant if $f(A, X) \; = \; f(PAP^{T}, \; PX)$ for any permutation P.

2. **Permutation Equivariance**:

- A function f is **permutation equivariant** if permuting the input graph results in the same permutation of the output. This is essential when learning node-level embeddings.
- **Definition**: For any graph function f: Rf: $R^{|V| \, x \, |V|} x \, R^{|V| \, x \, m} \; -> \; R^{|V| \, x \, d}$, f is permutation equivariant if $Pf(A, X) \; = \; f(PAP^T, \; PX)$ for any permutation P.
- This means that rearranging nodes in the input graph produces correspondingly rearranged embeddings in the output.



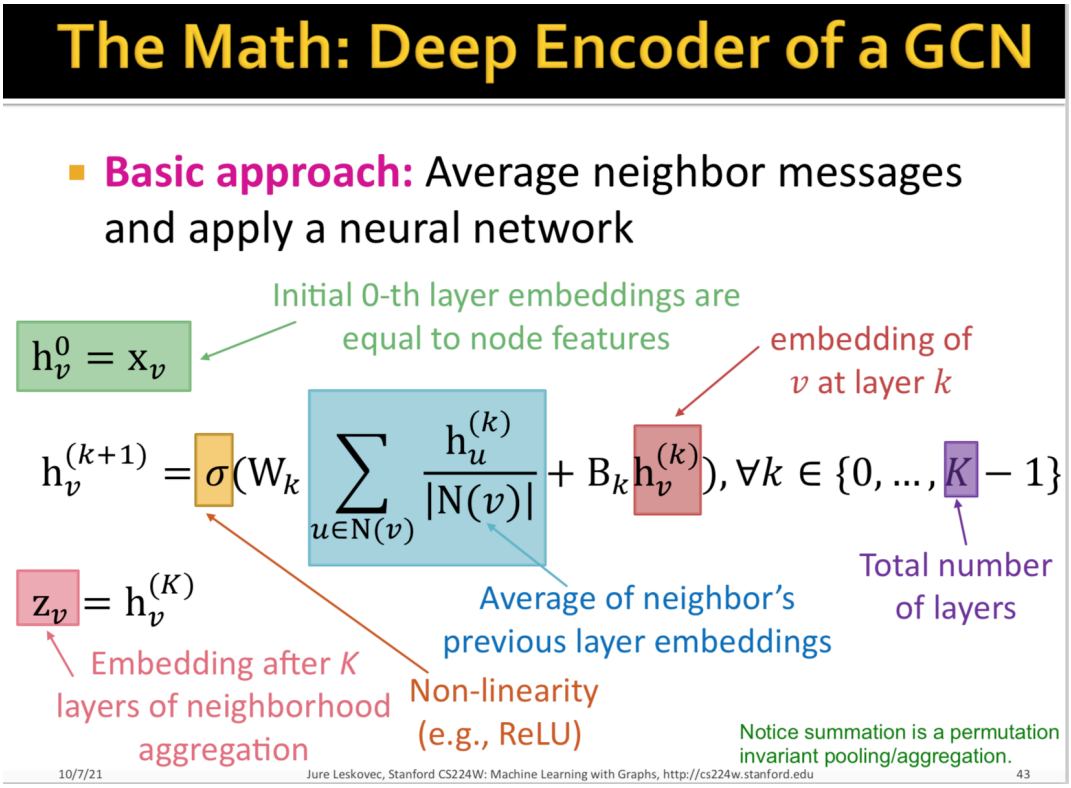
Goal: We want to design graph neural networks that are permutation invariant/equivariant by passing and aggregating information from neighbors.

---

4. Graph Convolutional Networks (GCNs): Layer Mechanics

The main idea of GCN's is that the node's neighborhood defines a computation graph, so we generate node embeddings based on local network neighborhoods. These models can be of arbitrary length, where Layer-0 is the embedding of node v is its input feature and Layer-k embedding gets information from nodes that are k hops away. **Key design decision:** how do you aggregate the information across the layers.

1. **Node Aggregation**:



We see that given a node, GCNs are permutation invariant and permutation equivariant.

2. **Matrix Representation**:
    - For efficient implementation, GCNs use sparse matrix operations.

      We can see that $\Sigma_{u \in N(v)} h_u^{(k-1)} / |N(v)| \rightarrow H^{(k+1)} = D^{-1}AH^{(k)}$

      So:

$$H^{(k+1)} = \sigma(\tilde{A}H^{(k)}W_k^T + H^{(k)}B_k^T)$$

    - **Explanation**:
        - H(k) = [h1(k) … h|v|(k)]: Node feature matrix at layer k.
        - A tilde - modified adjacency matrix with self-loops
        - D^(-1) inverse of degree of the nodes
    - This operation normalizes and aggregates node features using the adjacency matrix.

---

5. Training GNNs

1. **Loss Function**:
    - Depending on the task, the loss function can vary:
        - **Node Classification**: Cross-entropy loss.
        - **Link Prediction**: Binary cross-entropy loss comparing predicted link probabilities with actual links.
2. **Training process**
    - Define a neighborhood aggregation function
    - Define a loss function on the embeddings
    - Train on a set of nodes, ie a batch of compute graphs
    - Generate embeddings for nodes as needed

The same aggregation parameters are shared for all nodes. This means that we can generalize to unseen nodes and/or graphs!

---

6. GNNs, CNNs, Transformers

CNNs can be seen as a special GNN with fixed neighbor size and ordering. The size of the filter is predefined and the advantage of GNN is it processes arbitrary graphs with different degrees for each node. But, CNNs are not permutation invariant nor equivariant - switching the order of the pixels leads to different outputs.

Transformer is one of the most popular architecture due to its key component of self-attention. Attention is defined as a technique to compute a weighted sum of the values, dependent on the query, given a set of vector values and a vector query. Transformer can be viewed as a special GNN that runs on a full connected word graph.

---

7. Summary

- **GCNs**:
    - Powerful for learning on graph-structured data.
    - Built using permutation-invariant functions for aggregation and parameter sharing.
    - Efficient implementation using sparse matrix operations, suitable for small to medium-sized graphs.
    - Run mean aggregation; can be expressed in matrix form
- **Generalization**:
    - CNNs and Transformers can be seen as special cases or related architectures within the GNN framework.

Additional Resource:

- **GNN 101**: An interactive visualization tool developed at the University of Michigan, offering insight into GNN operations. It provides an intuitive understanding of message passing and aggregation functions.
- **Deep Learning Review**: at the end of lecture 3 slides, there is a deep learning review!