Graph Neural Netowrks

Saeed Saremi

Assigned reading: Chapter 13

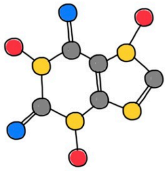
November 21, 2024

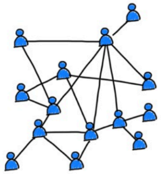
OUTLINE

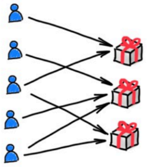
- graph-structured data and graphs
- adjacency matrix and the permutation symmetry
- Graph Neural Networks (GNN)
- expressive power of GNNs and the Weisfeiler-Lehman test

Boris Weisfeiler

GRAPH-STRUCTURED DATA



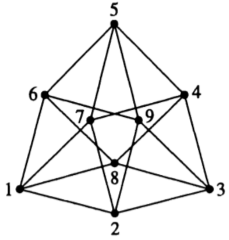




- Graphs are mathematical abstractions that can describe complex systems of interactions.
- Examples include molecular graphs, social networks, and recommender systems.
- Every machine learning problem we have considered so far in the course – e.g., regression, classification, and generative modeling – can be posed for graph-structured data.

VERTICES & EDGES

- A *graph* G an ordered pair of disjoint sets (V, E) such that E is a subset of the set $V^{(2)}$ of unordered pairs of V.
- The set V is the set of *vertices* and E is the set of *edges*. An edge $\{u, v\}$ is said to join the vertices u and v and is denoted by uv (or vu).
- If $uv \in E(G)$, then u and v are adjacent, or neighbouring, vertices of G. The set of all *neighbours* of node v is denoted by $\Gamma(v)$.
- We usually think of a graph as a collection of vertices some of which are joined by edges.
- We like to draw small graphs! Given vertices $V = \{1, 2, ..., 9\}$ and edges $E = \{12, 23, 34, 45, 56, 61, 17, 72, 29, 95, 57, 74, 48, 83, 39, 96, 68, 81\}$, the graph $G = (V, E)$ is immediately comprehended by looking at



ADJACENCY MATRIX

- A natural way to represent a graph is by using the *adjacency matrix*.
- The adjacency matrix $A = A(G) = (a_{uv})$ of a graph G is the $n \times n$ matrix:

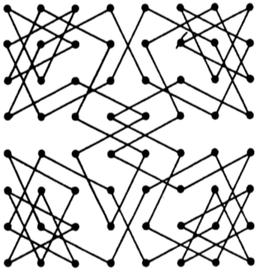
$$a_{uv} = \begin{cases} 1 & \text{if } uv \in E(G), \\ 0 & \text{otherwise.} \end{cases}$$

- An example for a 5 node graph:

$$A = \begin{pmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \end{pmatrix}$$

✎ Take the vertices of a pentagon and draw the graph! (How should we label the vertices?)

CANONICAL ORDERING



PERMUTATION MATRIX

- The adjacency matrix defines the structure of a graph. Naively, we can flatten it and use an MLP to solve our machine learning tasks. The problem is that labeling of nodes is arbitrary and changes the matrix.
- This is formalized by the permutation π which is a bijection from $[n]$ to $[n]$:

$$\pi : [n] \to [n],$$

  which we can represent with the *two-line notation*, e.g.,

$$\pi = \begin{matrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 3 & 1 & 5 & 4 \end{matrix}$$

- We can represent the permutation π with a matrix $P(\pi) = (p_{uv})$:

$$p_{uv} = \begin{cases} 1 & \text{if } \pi(u) = v, \\ 0 & \text{otherwise.} \end{cases}$$

- For the example above:

$$P = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

PERMUTATION EFFECTS

- Let's assume we have feature vectors $x_v \in \mathbb{R}^d$ for all vertices in the graph, which we can represent compactly with the matrix X in $\mathbb{R}^{n\times d}$:

$$X = \begin{pmatrix} x_1^\top \\ \vdots \\ x_n^\top \end{pmatrix}$$

- Now, the relabeling of the graph with the permutation π has the following effect on X:

$$\tilde{X} = PX$$

  (The proof is simple, as P permutes the rows.)

- For the adjacency matrix, both the rows and columns become permuated, therefore the new (post permutation) adjacency matrix is given by:

$$\tilde{A} = PAP^\top$$

PERMUTATION INVARIANCE & EQUIVARIANCE

- We may want to make node-specific predictions g, e.g., *node classification*. Since specific node ordering is arbitrary, under permutations of the node labels our predictions should permute the same way. Therefore, node predictions should be equivariant with respect to node reordering:

$$g(PX, PAP^\top) = Pg(X, A)$$

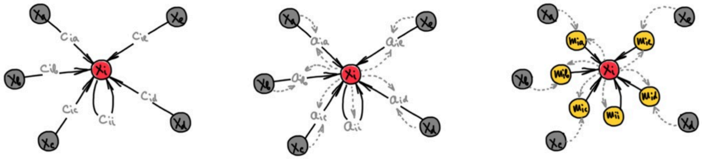
- In *graph classification* we make a graph-wide prediction f given the dataset $\{(G_i, y_i)\}_{i=1}^m$. Since the specific ordering we choose for the graph is arbitrary, our prediction must be invariant to node label reordering:

$$f(PX, PAP^\top) = f(X, A).$$

- Data augmentation is not an an option since the number of allowed permutations is exponentially large:

$$n! \approx (n/e)^n$$

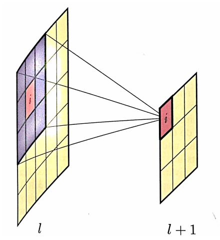
NEURAL MESSAGE PASSING

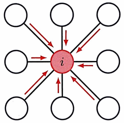


- Inspired by multilayer perceptrons, we define graph neural networks (GNN) by constructing a computational notion of *layer* for graph-structured data that can be applied repeatedly.
- The construction of layer should respect invariance (or equivariance) under permutation. In addition, graphs come in various sizes which our GNN should be able to handle.
- A general framework to construct GNNs goes around *message passing*.[1]

[1] Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., & Dahl, G. E. (2017). "Neural message passing for quantum chemistry," In *International Conference on Machine Learning*.

✎ CONVNETS REVISITED





GRAPH NEURAL NETWORKS

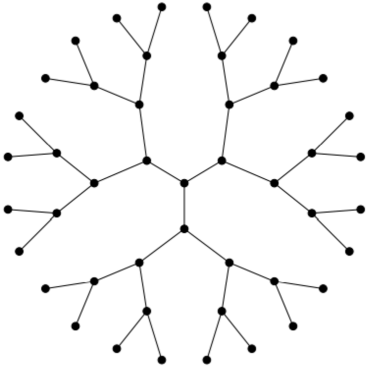
- GNNs use the graph structure and node features x_v to learn a representation vector of a node, h_v, or the entire graph, h_G.
- Inspired by ConvNets (or by WL graph isomorphism test), GNNs follow a neighborhood aggregation strategy, where we iteratively update the representation of a node by aggregating representations of its neighbors.
- After k iterations of aggregation, a node's representation captures the structural information within its k-hop network neighborhood.
- Formally, the k-th layer of a GNN is:[2]

$$a_v^{(k)} = \mathrm{AGGREGATE}^{(k)}\left(\left\{h_u^{(k-1)} : u \in \Gamma(v)\right\}\right),$$
$$h_v^{(k)} = \mathrm{COMBINE}^{(k)}\left(h_v^{(k-1)}, a_v^{(k)}\right),$$

where $h_v^{(k)}$ is the feature vector of node v at the k-th layer; $h_v^{(0)} = x_v$.

[2] Xu, K., Hu, W., Leskovec, J., & Jegelka, S. (2018). How powerful are graph neural networks?

Illustrate the *k*-hop network neighborhood (the "receptive field").



GNN ARCHITECTURES

The functions $\mathrm{AGGREGATE}^{(k)}$ and $\mathrm{COMBINE}^{(k)}$ determine the GNN.

- *Graph convolutional networks*:

  $$h_v^{(k)} = \mathrm{ReLU}\left(W \cdot \mathrm{MEAN}\left(\left\{h_u^{(k-1)} : u \in \Gamma(v) \cup \{v\}\right\}\right)\right),$$

  where W is a learnable matrix.
- For *node classification*, the node representation $h_v^{(K)}$ of the final iteration is used for prediction.
- For *graph classification*, the $\mathrm{READOUT}$ function aggregates node features from the final iteration to obtain the entire graph's representation h_G:

  $$h_G = \mathrm{READOUT}\left(\left\{h_v^{(K)} : v \in G\right\}\right),$$

  where $\mathrm{READOUT}$ is a permutation invariant function such as summation.

✎ Let's come up with *graph attention networks*! Hint: incorporate self-attention into the $\mathrm{AGGREGATE}$ function.

WEISFEILER-LEHMAN TEST & GNNs

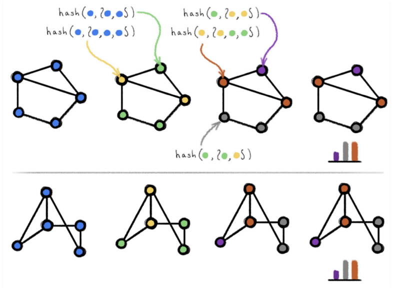


Figure: The WL test iteratively (1) aggregates the labels of nodes and their neighborhoods, and (2) hashes the aggregated labels into *unique* new labels.

- The graph isomorphism problem asks whether two graphs are topologically identical. No polynomial-time algorithm is known for it yet.
- In the WL test for graph isomorphism, the two graphs are non-isomorphic if at some iteration the labels of the nodes between the two graphs differ.
- WL could fail for simple non-isomorphic graphs.
- It turns out GNNs can be *at most* as powerful as WL test!
- This gives rise to higher-order GNNs which mimic 2-WL, 3-WL, . . .