Graph Neural Networks in Big Data Analytics: Introduction IV

Alexander Schönhuth

UNIVERSITÄT BIELEFELD
Faculty of Technology

Bielefeld University

November 10, 2022

Contents Today

- Reminder: Message Passing
- Convolution on Graphs
- Polynomial Filters
- Modern GNN's

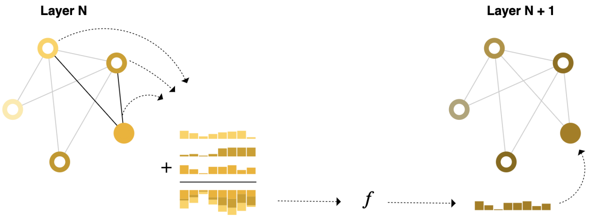
Reminder: Message Passing

Message Passing: Motivation

- Simple GNN's presented earlier
  - do not pool within the GNN layer
  - have learned embeddings unaware of graph connectivity
- *Goal:* Neighboring nodes and edges
  - exchange information
  - influence each other's updated embeddings
- *Solution:* Message passing

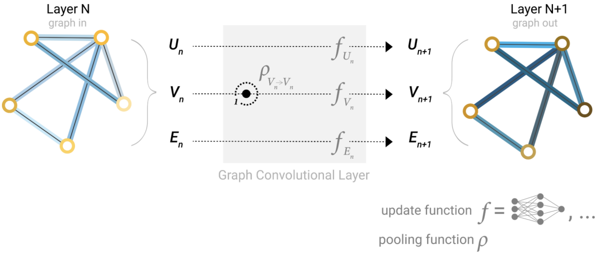
Message Passing: Protocol

1. Each node: gather all embeddings (= *messages*) of neighboring nodes
2. Aggregate all messages using an aggregation function
3. Pooled messages passed through update function (e.g. learned NN)



Message passing: Aggregating information from neighboring nodes
From `https://distill.pub/2021/gnn-intro/`

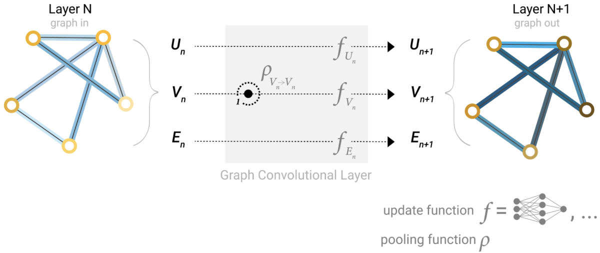
Message Passing and Convolution I



Message passing as convolution on graphs
From `https://distill.pub/2021/gnn-intro/`

- Message passing and convolution are similar in spirit
- *Commonality:* Process element's neighbors to update element
  - *Graphs:* Elements are nodes
  - *Images:* Elements are pixels

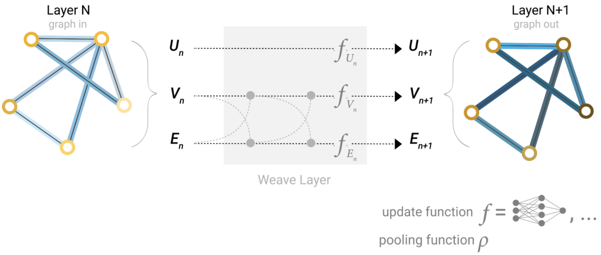
Message Passing and Convolution II



Message passing as convolution on graphs
From `https://distill.pub/2021/gnn-intro/`

- Message passing and convolution are similar in spirit
- *Difference:*
  - *Graphs:* Number of neighbors varies per node
  - *Images:* Number of neighbors constant per pixel

Pooling Within Layers: Reminder I

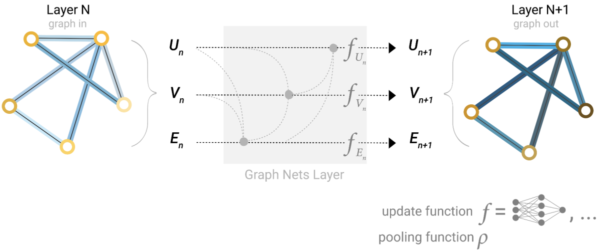


Weave layer: learning node information from edges and learning edge information from nodes

From `https://distill.pub/2021/gnn-intro/`

- f_{V_n} processes node information from edge information and node itself
- f_{E_n} processes edge information from node information and edge itself

Pooling Within Layers: Reminder II



Global information: aggregate from nodes and edges
From `https://distill.pub/2021/gnn-intro/`

- *Issue:* After k layers, nodes can reach k-neighborhoods at most
- *Solution:* Consider *master node* or *global context vector*
- Update global context vector by pooling node and/or edge information

Message Passing and Random Walks

- Let $n := |V|$ be the number of nodes of a graph (V, E)
- Let $A \in \{0, 1\}^{n \times n}$ be its adjacency matrix
- Let m be the length of node information vectors
- Let $X \in \mathbb{R}^{n \times m}$ be the node feature matrix
  - Rows in X are m-dimensional information vectors of nodes

Consider

$$B = AX$$

We obtain

$$B_{ij} = A_{i1}X_{1j} + ... + A_{in}X_{nj} = \sum_{\substack{k=1 \\ A_{ik}>0}}^{n} A_{ik}X_{kj}$$

MESSAGE PASSING AND RANDOM WALKS

Interpretation:

- Each row B_i reflects a new information vector for node v_i
- B_i again has dimension m
- Each B_{ij} is the aggregation of j-th entries of information vectors of neighbors of v_i
  - ☞ Note that $A_{ik} = 1$ if and only if v_i and v_k are neighbors
- Replacing A with A^K yields aggregation of information vectors of K-neighbors
  - ☞ $A^K_{ik} = 1$ iff (sic!) v_i and v_k can be connected by path of length K
- This relates to random walks on the graph
  - ☞ Recall the random walk mechanism for computing PageRank

Graph Attention Networks

Motivation:

- When aggregating one would like to consider weighted sums

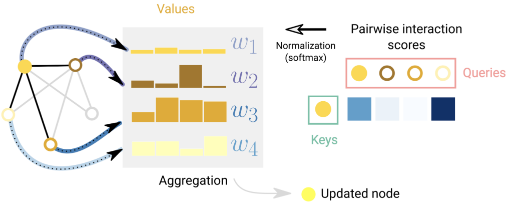
$$B_{ij} = w_{ij,1}A_{i1}X_{1j} + \ldots + w_{ij,n}A_{in}X_{nj} = \sum_{\substack{k=1 \\ A_{ik}>0}}^{n} w_{ij,k}A_{ik}X_{kj}$$

  ☞ Some neighbors are more important than others

- *Challenge:* Compute weights in permutation invariant way
- *Solution:* Base weights on pairs of nodes alone, so

$$w_{ij,k} = f(v_i, v_k)_j$$

Graph Attention Networks



Graph attention network: mechanism
From `https://distill.pub/2021/gnn-intro/`

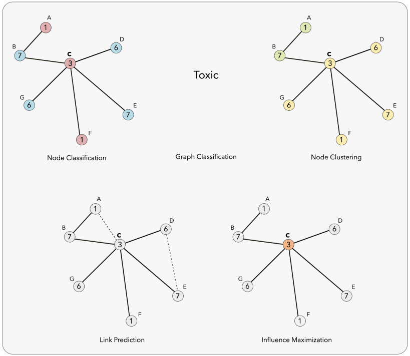
- *Attention Networks:* Compute *value* from comparing *key* and *query*
- *Here:* Compare information vectors of two nodes
  - One node is query, other node is key, weight is value
  - *Example:*

$$f(v_i, v_j) = \langle v_i, v_k \rangle$$

evaluates as scalar product of information vectors of v_i and v_k

Convolution on Graphs

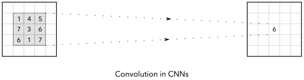
Reminder: Problems on Graphs



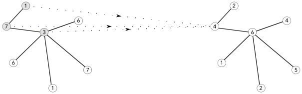
Non-exhaustive list of problems

From `https://distill.pub/2021/understanding-gnns/`

Convolution on Graphs



Convolution in CNNs



Convolution on graphs

From `https://distill.pub/2021/understanding-gnns/`

Issue: **Irregularity of graph**

Polynomial Filters on Graphs

The Graph Laplacian: Definition

Definition [Graph Laplacian]: Let

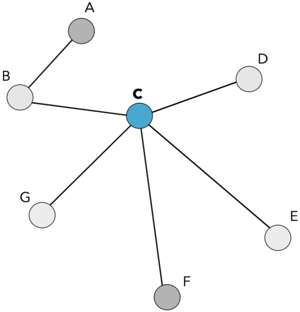
- $G = (V, E)$ be a graph where $|V| = n$
- $A = A(G) \in \{0, 1\}^{n \times n}$ be the adjacency matrix of G
- $D = D(G) \in \mathbb{N}^{n \times n}$ be the diagonal matrix defined by

$$D_{ij} = \begin{cases} \sum_{u \in V} A_{v_i u} & i = j \\ 0 & \text{otherwise} \end{cases}$$

- D_{ii} is the *degree* of v_i, i.e. the number of edges connected with v_i
- The *Laplacian* $L = L(G)$ is defined by

$$L(G) := D(G) - A(G)$$

The Graph Laplacian: Example



Input Graph G

	A	B	C	D	E	F	G
A	1	−1					
B	−1	2	−1				
C		−1	5	−1	−1	−1	−1
D			−1	1			
E			−1		1		
F			−1			1	
G			−1				1

Laplacian L of G

Zeros are not displayed. The Laplacian depends only on the graph structure.

From `https://distill.pub/2021/understanding-gnns/`

UNIVERSITÄT BIELEFELD

The Graph Laplacian: Remarks

- The graph Laplacian is the discrete analog of the Laplacian from calculus
- It virtually stores exactly the same information as A, but has interesting properties in its own right
- See `https://csustan.csustan.edu/~tom/Clustering/GraphLaplacian-tutorial.pdf` for further information, if interested

Polynomials of the Laplacian

One can build *polynomials of the Laplacian* of the form

$$p_w(L) = w_0 I_n + w_1 L + w_2 L^2 + ... + w_d L^d = \sum_{i=0}^{d} w_i L^i \tag{1}$$

where I_n is the n-dimensional identity matrix.
Alternatively, each such polynomial can be represented by its *vector of coefficients*

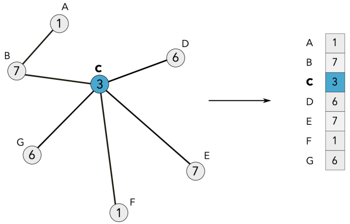
$$w = [w_0, ..., w_d] \tag{2}$$

Remark:

- $p_w(L)$ is an $n \times n$-Matrix for each w, just like L
- The $p_w(L)$ represent the equivalent of filters in CNN's
- We will see why that is...

Polynomials of the Laplacian II

- In the following, each node $v \in V$ stores information $x_v \in \mathbb{R}$
  - For ease of presentation only
  - Everything applies also for multi-dimensional vectors
- Stack real-valued features into vector $x \in \mathbb{R}^n$



Collecting node information into vector.

From `https://distill.pub/2021/understanding-gnns/`

Polynomial Filters: Definition

- In the following, each node $v \in V$ stores information $x_v \in \mathbb{R}$
  - For ease of presentation only
  - Everything applies also for multi-dimensional vectors
- Stack real-valued features into vector $x \in \mathbb{R}^n$
- *Convolution with a polynomial filter* p_w is then defined as

$$x' = p_w(L)x \tag{3}$$

that is, by applying the matrix $p_w(L) \in \mathbb{R}^{n \times n}$ to the vector $x \in \mathbb{R}^n$

POLYNOMIAL FILTERS: EXAMPLES

Examples:

- $w = [w_0, 0, ..., 0]$:

$$x' = p_w(L) = w_0 I_n x + 0 + ... + 0 = w_0 x$$

- $w = [0, 1, 0, ..., 0]$:

$$x' = p_w(L) = Lx$$

  Let $\mathcal{N}(v)$ is the *neighborhood* of v, that is all nodes attached to v via an edge, so

$$x'_v = (Lx)_v = \sum_{u \in G} L_{vu} x_u = \sum_{u \in G} (D_{vu} - A_{vu}) x_u = D_{vv} x_v - \sum_{u \in \mathcal{N}(v)} x_u$$

  - *Interpretation:* Features of v are combined with features of immediate neighbors ☞ message passing

Polynomial Filters: Polynomial Degree

- Let $\mathrm{dist}(u, v)$ be the length of the shortest path between nodes $u, v \in V$
  - For example, $(u, v) \in E$ corresponds to $\mathrm{dist}(u, v) = 1$
- Basic calculations imply

$$\mathrm{dist}(u, v) > i \quad \text{implies} \quad (L^i)_{uv} = (\underbrace{L \times \ldots \times L}_{i \text{ times}})_{uv} = 0 \tag{4}$$

- Let $p_w(L)$ have polynomial degree d. One obtains

$$x'_v = (p_w(L)x)_v = \sum_{i=0}^{d} w_i \sum_{u \in V} (L^i)_{vu} x_u = \sum_{i=0}^{d} w_i \sum_{\substack{u \in V \\ \mathrm{dist}_G(v,u) \leq i}} (L^i)_{vu} x_u \tag{5}$$

- (5): convolution at node v only with nodes at most d hops away

Summary: Degree of localization governed by degree of polynomial filter

Polynomial Filters: Permutation Invariance

- Let $P \in \{0,1\}^{n \times n}$ be a permutation matrix
  - Applying P to any vector permutes the order of its entries
  - P has exactly one 1 in each row and each column
  - All other entries are zero
  - P is orthogonal, implying $PP^T = P^TP = I_n$ (*)
- A function on $\mathbb{R}^n$ is *node-order invariant* iff $f(Px) = Pf(x)$ for all P
- Permuting order of nodes using P translates into
  - $x \mapsto Px$
  - $L \mapsto PLP^T$
  - $L^i \mapsto (PLP^T)^i = \underbrace{PLP^T \times \ldots \times PLP^T}_{i \text{ times}} \overset{(*)}{=} PL^iP^T$

Polynomial Filters: Permutation Invariance

- A function on $\mathbb{R}^n$ is *node-order invariant* iff $f(Px) = Pf(x)$ for all P
- Permuting order of nodes using P translates into
  - $x \mapsto Px$
  - $L \mapsto PLP^T$
  - $L^i \mapsto (PLP^T)^i = \underbrace{PLP^T \times \ldots \times PLP^T}_{i \text{ times}} \overset{(*)}{=} PL^iP^T$
- For $f(x) = p_w(L)x$ one obtains

$$f(Px) = p_w(L)(Px) = \sum_{i=0}^{d} w_i(PL^iP^T)(Px) = P\sum_{i=0}^{d} w_iL^ix = Pf(x)$$

Summary: Polynomial filters are node-order invariant

UNIVERSITÄT BIELEFELD

Polynomial Filters in Practice: ChebNet

- Let $\tilde{L}$ be the *normalized Laplacian* defined by

$$\tilde{L} := \frac{2L}{\lambda_{\max}(L)} - I_n \tag{6}$$

  where $\lambda_{\max}(L)$ is the largest eigenvalue of L

- ChebNet refined the idea of polynomial filters by re-defining

$$p_w(L) = \sum_{i=0}^{d} w_i T_i(\tilde{L}) \tag{7}$$

  where T_i is the degree-i *Chebyshev polynomial of the first kind*

  - Combining T_i with $\tilde{L}$ established the breakthrough

- *Motivation:*
  - L is positive semi-definite: all eigenvalues are non-negative
  - If $\lambda_{\max}(L) > 1$, entries of powers of L rapidly increase
  - $\tilde{L}$ rescaled version of L with eigenvalues in $[-1, 1]$
  - The T_i behave in a numerically stable manner

Polynomial Filters: Stacking Layers

Start with the original features.

$$h^{(0)} = x$$

Color Codes:

- Computed node embeddings.
- Learnable parameters.

Then iterate, for $k = 1, 2, \ldots$ upto K:

$$p^{(k)} = p_{w^{(k)}}(L)$$

Compute the matrix $p^{(k)}$ as the polynomial defined by the filter weights $w^{(k)}$ evaluated at L.

$$g^{(k)} = p^{(k)} \times h^{(k-1)}$$

Multiply $p^{(k)}$ with $h^{(k-1)}$: a standard matrix-vector multiply operation.

$$h^{(k)} = \sigma\left(g^{(k)}\right)$$

Apply a non-linearity σ to $g^{(k)}$ to get $h^{(k)}$.

Note: weights re-used at every node, as in CNN's.

From `https://distill.pub/2021/understanding-gnns/`

MODERN GNN'S

- Re-consider $p_w(L) = L$, yielding

$$(Lx)_v = D_v x_v - \sum_{u \in \mathcal{N}(v)} x_u \tag{8}$$

- (8) decomposes into
  - Aggregating over immediate neighbor features $x_u, u \in \mathcal{N}(v)$
  - Combining with node v's own feature x_v
- *Idea:* Generalize by considering different kinds of aggregation and combination steps
- *Caveat:* Aggregation needs to be node-order invariant
- Iteratively repeating 1-hop localized convolutions K times: receptive field including all nodes up to K hops away

Graph Convolutional Networks (GCN's)

For $k = 1, \ldots, K$

$$h_v^{(k)} = f^{(k)}\left(W^{(k)} \cdot \frac{\sum_{u \in \mathcal{N}(v)} h_u^{(k-1)}}{|\mathcal{N}(v)|} + B^{(k)} \cdot h_v^{(k-1)} \right) \quad \text{for all } v \in V.$$

Node v's embedding at step k.

Mean of v's neighbour's embeddings at step $k-1$.

Node v's embedding at step $k-1$.

Color Codes:

- Embedding of node v.
- Embedding of a neighbour of node v.
- (Potentially) Learnable parameters.

From https://distill.pub/2021/understanding-gnns/

Graph Convolutional Networks (GCN's) I

$$h_v^{(k)} = f^{(k)}\left(W^{(k)} \cdot \frac{\sum_{u \in \mathcal{N}(v)} h_u^{(k-1)}}{|\mathcal{N}(v)|} + B^{(k)} \cdot h_v^{(k-1)}\right) \quad \text{for all } v \in V.$$

From `https://distill.pub/2021/understanding-gnns/`

- Derive predictions from $h_v^{(K)}$
- Function $f^{(k)}$, matrices $W^{(k)}, B^{(k)}$ shared across nodes
- Dividing by $|\mathcal{N}(v)|$ implements normalization; alternative normalization schemes conceivable

Graph Attention Networks (GAN's)

$$h_v^{(k)} = f^{(k)}\left(W^{(k)} \cdot \left[\sum_{u \in \mathcal{N}(v)} \alpha_{vu}^{(k-1)} h_u^{(k-1)} + \alpha_{vv}^{(k-1)} h_v^{(k-1)}\right]\right) \quad \text{for all } v \in V.$$

Node v's embedding at step k.

Weighted mean of v's neighbour's embeddings at step $k-1$.

Node v's embedding at step $k-1$.

for $k = 1, \ldots, K$, where normalized attention weights $\alpha^{(k)}$ are generated by $A^{(k)}$

$$\alpha_{vu}^{(k)} = \frac{A^{(k)}(h_v^{(k)}, h_u^{(k)})}{\sum_{w \in \mathcal{N}(v)} A^{(k)}(h_v^{(k)}, h_w^{(k)})} \quad \text{for all } (v, u) \in E.$$

Color Codes:

- Embedding of node v.
- Embedding of a neighbour of node v.
- (Potentially) Learnable parameters.

From `https://distill.pub/2021/understanding-gnns/`

Graph Attention Networks (GAN's) II

$$h_v^{(k)} = f^{(k)}\left(W^{(k)} \cdot \left[\sum_{u \in \mathcal{N}(v)} \alpha_{vu}^{(k-1)} h_u^{(k-1)} + \alpha_{vv}^{(k-1)} h_v^{(k-1)}\right]\right) \quad \text{for all } v \in V.$$

From `https://distill.pub/2021/understanding-gnns/`

- Derive predictions from $h_v^{(K)}$
- Function $f^{(k)}$, matrices $W^{(k)}$ and attention mechanism $A^{(k)}$ (generally another neural network) shared across nodes
- Here: single-headed attention; multi-headed attention similar

References

- ChebNet: https://proceedings.neurips.cc/paper/2016/file/04df4d434d481c5bb723be1b6df1ee65-Paper.pdf
- Graph Convolutional Networks (GCN's): https://openreview.net/forum?id=SJU4ayYgl
- Graph Attention Networks (GAN's): https://openreview.net/forum?id=rJXMpikCZ

Thanks for your attention!