Scaling up by Simplifying GNN Architecture

CS224W: Machine Learning with Graphs

Jure Leskovec, Stanford University

http://cs224w.stanford.edu

Roadmap of Simplifying GCN

- **We start from Graph Convolutional Network** (GCN) [Kipf & Welling ICLR 2017].
- We simplify GCN (LightGCN) by **removing the non-linear activation** from the GCN [Wu et al. ICML 2019].
  - *Wu et al.* demonstrated that the performance on benchmark is not much lower by the simplification.
  - Simplified GCN turns out to be extremely scalable by the model design.
  - **The simplification strategy is very similar to the one used by LightGCN for recommender systems.**

Simple Graph Convolutional Network

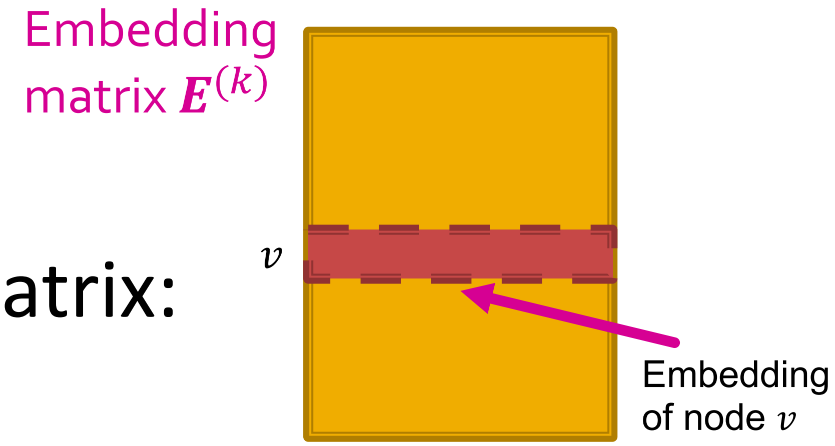
Preview

We have a graph with adjacency matrix **A** and nodes with features **X** and labels **y**

- Initial embeddings: $E^{(0)} = X$
- First convolution layer: $E^{(1)} = \text{relu}(A.E^{(0)}.W^{(1)})$
- Second layer: $E^{(2)} = \text{relu}(A.E^{(1)}.W^{(2)}) = \text{relu}(A.\text{relu}(A.E^{(0)}.W^{(1)}).W^{(2)})$
- If we discard the relu: $E^{(2)} = A.A.E^{(0)}.W^{(1)}.W^{(2)}$ ..... $(\mathit{W} = W^{(1)}.W^{(2)}...)$
- Therefore $E^{(n)} = A^n.E^{(0)}.\mathit{W} = A^n.X.\mathit{W}$
- For classification, we solve $A^n.X.\mathit{W} = y$

Quick Overview of LightGCN (1)

- Adjacency matrix: $\boldsymbol{A}$
- Degree matrix: $\boldsymbol{D}$
- Normalized adjacency matrix: $\widetilde{\boldsymbol{A}} \equiv \boldsymbol{D}^{-1/2}\boldsymbol{A}\boldsymbol{D}^{-1/2}$



- Let $\boldsymbol{E}^{(k)}$ be the embedding matrix at k-th layer.
- Let $\boldsymbol{E}$ be the input embedding matrix.
- GCN's aggregation in the matrix form
  - $\boldsymbol{E}^{(k+1)} = \mathrm{ReLU}\left(\widetilde{\boldsymbol{A}}\boldsymbol{E}^{(k)}\boldsymbol{W}^{(k)}\right)$

Quick Overview of LightGCN (2)

- Removing ReLU non-linearity gives us
  - $\boldsymbol{E}^{(K)} = \boxed{\widetilde{\boldsymbol{A}}^{K}\ \boldsymbol{E}}\,\boldsymbol{W}$, where $\boldsymbol{W} \equiv \boldsymbol{W}^{(0)} \cdots \boldsymbol{W}^{(K-1)}$

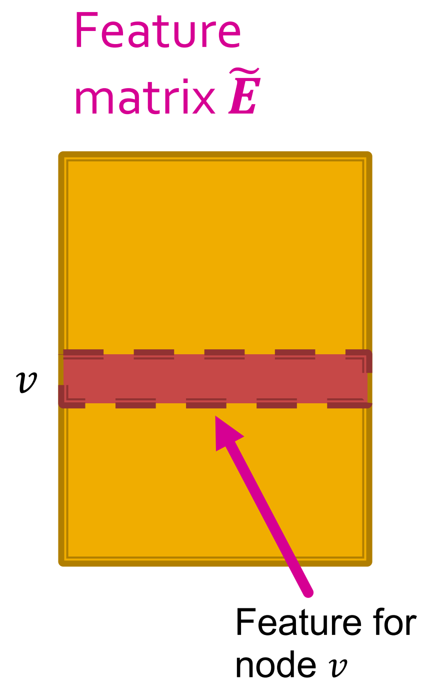
    Diffusing node embeddings along the graph
- Efficient algorithm to obtain $\widetilde{\boldsymbol{A}}^{K}\ \boldsymbol{E}$
  - Start from input embedding matrix $\boldsymbol{E}$.
  - Apply $\boldsymbol{E} \leftarrow \widetilde{\boldsymbol{A}}\ \boldsymbol{E}$ for K times.
- Weight matrix $\boldsymbol{W}$ can be ignored for now.
  - $\boldsymbol{W}$ acts as a linear classifier over the diffused node embeddings $\widetilde{\boldsymbol{A}}^{K}\ \boldsymbol{E}$.

Differences to LightGCN

- LightGCN adds **self-loops** to adjacency matrix $\boldsymbol{A}$:
  - $\boldsymbol{A} \leftarrow \boldsymbol{A} + \boldsymbol{I}$
  - Follows the original GCN by Kipf & Welling.
- LightGCN assumes input node embeddings $\boldsymbol{E}$ to be **given as features**.
  - Input embedding matrix $\boldsymbol{E}$ is **fixed** rather than learned.
  - **Important consequence**: $\widetilde{\boldsymbol{A}}^{K}\ \boldsymbol{E}$ needs to be calculated **only once**.
    - Can be treated as a **pre-processing step**.

Simplified GCN

- Let $\widetilde{\boldsymbol{E}} = \widetilde{\boldsymbol{A}}^K \boldsymbol{E}$ be pre-processed feature matrix.
  - Each row stores the pre-processed feature for each node.
  - $\widetilde{\boldsymbol{E}}$ can be used as input to any scalable ML models (e.g., linear model, MLP).
- LightGCN empirically shows learning a linear model over $\widetilde{\boldsymbol{E}}$ often gives performance comparable to GCN!



Comparison with Other Methods

- Compared to neighbor sampling and cluster-GCN, **simplified GCN is much more efficient**.
  - **Simplified GCN computes $\widetilde{\boldsymbol{E}}$ only once at the beginning.**
    - **The pre-processing (sparse matrix vector product, $\boldsymbol{E} \leftarrow \widetilde{\boldsymbol{A}}\ \boldsymbol{E}$) can be performed efficiently on CPU.**
  - Once $\widetilde{\boldsymbol{E}}$ is obtained, getting an embedding for node v only takes **constant time!**
    - Just look up a row for node v in $\widetilde{\boldsymbol{E}}$.
    - No need to build a computational graph or sample a subgraph.
- But the model is **less** expressive (next).

Potential Issue of Simplified GCN

- Compared to the original GNN models, **simplified GCN’s expressive power is limited due to the lack of non-linearity in generating node embeddings.**

Performance of Simplified GCN

- Surprisingly, in semi-supervised node classification benchmark, **simplified GCN works comparably to the original GNNs despite being less expressive.**

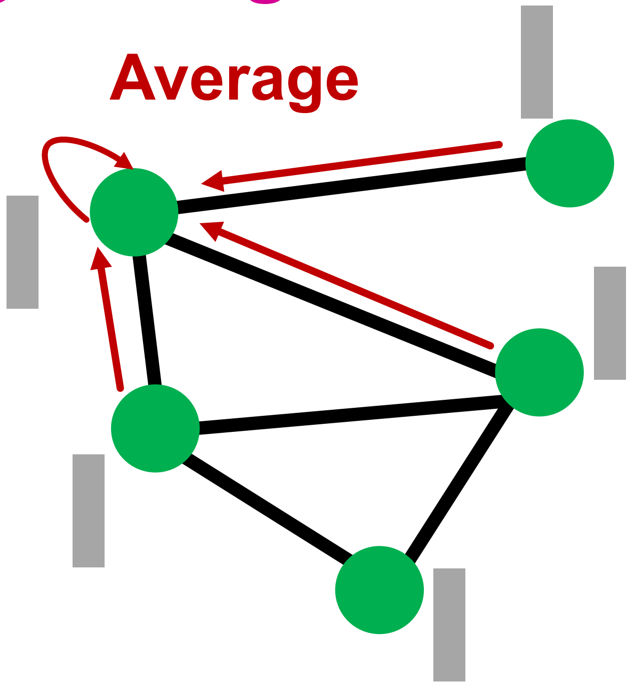
- **Why?**

Graph Homophily

- Many node classification tasks exhibit homophily structure, i.e., **nodes connected by edges tend to share the same target labels.**
- **Examples**:
  - Paper category classification in paper-citation network
    - Two papers tend to share the same category if one cites another.
  - Movie recommendation for users in social networks
    - Two users tend to like the same movie if they are friends in a social network.

When does Simplified GCN Work?

- Recall the preprocessing step of the simplified GCN: **Do $X \leftarrow \tilde{A}\, X$ for K times.**
- Pre-processed features are obtained **by iteratively averaging their neighboring node features.**
- **As a result, nodes connected by edges tend to have similar pre-processed features.**



When does Simplified GCN Work?

- **Premise**: Model uses the pre-processed node features to make prediction.
- Nodes connected by edges tend to get similar pre-processed features.

→ **Nodes connected by edges tend to be predicted the same labels by the model**

- **Simplified SGC's prediction aligns well with the graph homophily in many node classification benchmark datasets.**

Simplified GCN: Summary

- **Simplified GCN removes non-linearity in GCN and reduces to the simple pre-processing of node features.**
- Once the pre-processed features are obtained, scalable mini-batch SGD can be directly applied to optimize the parameters.
- **Simplified GCN works surprisingly well in node classification benchmark.**
  - The feature pre-processing aligns well with graph homophily in real-world prediction tasks.