LightGCN: Simplifying and Powering Graph Convolution Network for Recommendation

Xiangnan He
Kuan Deng
Yongdong Zhang

Xiang Wang
Yan Li
Meng Wang
Outline

- Background: NGCF
  - SIGIR 2019. Neural Graph Collaborative Filtering

- Model: LightGCN

- Fast Loss for LightGCN

- Conclusion & Future Work
Representation Learning in CF

**Model Personal History as User Feature**
- Integrate embeddings of historical items as user embeddings
- Or use autoencoders to generate user behaviors

**Model Single User-Item Pairs**
- Project each user/item ID into an embedding vector

**Model Holistic Interaction Graph**
- Apply embedding smooth constraints on connected nodes
- Perform embedding propagation via graph neural networks

Models:
- Hop-Rec [2018]
- GRMF [2015]
- GC-MC [2017]
- NGCF [2019]
- LightGCN [2020]
- MF [2009]
- BPRMF [2009]
- NCF [2017]
- CML [2017], LRML [2018]
- SVD++ [2008]
- FISM [2013]
- NAIS [2018], ...
- ACF [2017]
- Mult-VAE [2018]
- AutoRec [2015]
- CDAE [2016]

- GRMF [2015]
- GC-MC [2017]
- NGCF [2019]
- LightGCN [2020]
Recap: NGCF [Wang et al, SIGIR’19]

High-level Idea:

• Organize historical interactions as a user-item bipartite graph
• Capture CF signal via high-order connectivity
  • Definition: the paths that reach $u_1$ from any node with the path length $l$ larger than 1.

• E.g., why $u_1$ may like $i_4$?
  • $u_1 \leftarrow i_2 \leftarrow u_2 \leftarrow i_4$
  • $u_1 \leftarrow i_3 \leftarrow u_3 \leftarrow i_4$

NGCF’s contribution: explicitly modeling high-order connectivity in representation space via GNN.
NGCF: First-order Connectivity Modeling

Embedding Propagation, inspired by GNNs

- Propagate embeddings recursively on the graph → high-order connectivity
- Construct information flows in the embedding space → embed CF signal

➢ First-order Propagation
  ➢ Message Construction: generate message from one neighbor

message passed from \( i \) to \( u \)
\[
\mathbf{m}_{u \leftarrow i} = \frac{1}{\sqrt{|\mathcal{N}_u| |\mathcal{N}_i|}} \left( W_1 \mathbf{e}_i + W_2 (\mathbf{e}_i \odot \mathbf{e}_u) \right)
\]

- Discount factor
- Make message dependent on the affinity,
- Pass more information to similar nodes

➢ Message Aggregation: update ego node’s representation by aggregating message from all neighbors

\[
e_u^{(1)} = \text{LeakyReLU} \left( \mathbf{m}_{u \leftarrow u} + \sum_{i \in \mathcal{N}_u} \mathbf{m}_{u \leftarrow i} \right)
\]

self-connections all neighbors of \( u \)
NGCF: Higher-order Connectivity Modeling

- Stack more embedding propagation layers to explore high-order connectivity

$$ e_u^{(l)} = \text{LeakyReLU} \left( m_u^{(l)} + \sum_{i \in N_u} m_i^{(l)} \right), $$

representation of $u$ at the $l$-th layer

- The collaborative signal like $u_1 \leftarrow i_2 \leftarrow u_2 \leftarrow i_4$ can be captured in the embedding propagation process.

- **Final embedding:** concatenate the embedding from all layers
Our Argument

- Designs of NGCF are rather heavy and burdensome
  - Many operations are directly inherited from GCN without justification.

<table>
<thead>
<tr>
<th></th>
<th>GNNs</th>
<th>NGCF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original task</td>
<td>Node classification</td>
<td>Collaborative filtering</td>
</tr>
<tr>
<td>Input data</td>
<td>Rich node features</td>
<td>Only node ID</td>
</tr>
<tr>
<td></td>
<td>• Attributes, text, image data</td>
<td>• One-hot encoding</td>
</tr>
<tr>
<td>Feature</td>
<td>Distill useful information</td>
<td>Generate ID embeddings</td>
</tr>
<tr>
<td>transformation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Neighborhood</td>
<td>Pass messages from neighbors</td>
<td>Pass messages from neighbors to the egos</td>
</tr>
<tr>
<td>aggregation</td>
<td>to the egos</td>
<td></td>
</tr>
<tr>
<td>Nonlinear</td>
<td>Enhance representation ability</td>
<td>Negatively increases the difficulty for model training</td>
</tr>
</tbody>
</table>
Empirical Evidence on Training Difficulty

- Removing feature transformation (NGCF-f) \(\rightarrow\) decrease training loss
- Removing nonlinear activation (NGCF-n) \(\rightarrow\) increase training loss
- But, removing nonlinear activation & feature transformation (NGCF-fn) \(\rightarrow\) significantly decrease training loss
Empirical Evidence on Training Difficulty

- Removing feature transformation (NGCF-f) $\rightarrow$ improve testing accuracy
- Removing nonlinear activation (NGCF-n) $\rightarrow$ hurt testing accuracy
- Removing nonlinear activation & feature transformation (NGCF-fn) $\rightarrow$ significantly improve testing accuracy
Light Graph Convolution

NGCF

- Graph Convolution Layer
  \[ e_u^{(l)} = \text{LeakyReLU}(m_u^{(l)} + \sum_{i \in N_u} m_{u \leftarrow i}^{(l)}) \]
- Layer Combination
  \[ e_u^* = e_u^{(0)} \parallel \cdots \parallel e_u^{(L)} \]
- Matrix Form
  \[ E^{(l)} = \text{LeakyReLU}((\mathcal{L} + D)E^{(l-1)} + \mathcal{L}E^{(l-1)} \odot E^{(l-1)}) \]

LightGCN

- Light Graph Convolution Layer
  \[ e_u^{(k+1)} = \sum_{i \in N_u} \frac{1}{\sqrt{|N_u|} \sqrt{|N_i|}} e_i^{(k)} \]
- Layer Combination
  \[ e_u = \sum_{k=0}^{K} \alpha_k e_u^{(k)} \]
- Matrix Form
  \[ E^{(k+1)} = (D^{-\frac{1}{2}} AD^{-\frac{1}{2}})E^{(k)} \]

Only simple weighted sum aggregator is remained
- No feature transformation
- No nonlinear activation
- No self connection

He et al. LightGCN: Simplifying and Powering Graph Convolution Network for Recommendation. SIGIR 2020
Overall Framework

Light Graph Convolution (LGC)

\[ E = \alpha_0 E^{(0)} + \alpha_1 E^{(1)} + \alpha_2 E^{(2)} + \ldots + \alpha_K E^{(K)} \]

\[ = \alpha_0 E^{(0)} + \alpha_1 \tilde{A} E^{(0)} + \alpha_2 \tilde{A}^2 E^{(0)} + \ldots + \alpha_K \tilde{A}^K E^{(0)} \]

importance of the k-th layer embedding in constituting the final embedding

He et al. LightGCN: Simplifying and Powering Graph Convolution Network for Recommendation. SIGIR 2020
Model Analysis

• Relation with \textbf{SGCN} [Wu et al. ICML 2019]:
  • By doing layer combination, LightGCN subsumes the effect of self-connection \(\Rightarrow\) \textbf{no need to add self-connection in adjacency matrix}.

\[
E^{(K)} = \binom{K}{0}E^{(0)} + \binom{K}{1}AE^{(0)} + \binom{K}{2}A^2E^{(0)} + \ldots + \binom{K}{K}A^KE^{(0)}.
\]

• Relation with \textbf{APPNP} [Klicpera et al. ICLR 2019]:
  • By setting \(\alpha_k\) properly, LightGCN can recover APPNP \(\Rightarrow\) \textbf{use a large} \(K\) \textbf{for long-range modeling with controllable oversmoothing}.

\[
E^{(K)} = \underbrace{\beta E^{(0)}}_{\text{base}} + \underbrace{\beta(1 - \beta)AE^{(0)}}_{\text{first order}} + \underbrace{\beta(1 - \beta)^2A^2E^{(0)}}_{\text{second order}} + \ldots + \underbrace{(1 - \beta)^K\tilde{A}^KE^{(0)}}_{\text{higher order}}.
\]
Experiment Settings

Datasets:

Gowalla, Yelp2018, Amazon-Book

Evaluation Metrics:

recall@20, ndcg@20

Dataset partition: randomly select 80% data for training set, and 20% data for testing set.

Table 2: Statistics of the experimented data.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>User #</th>
<th>Item #</th>
<th>Interaction #</th>
<th>Density</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gowalla</td>
<td>29,858</td>
<td>40,981</td>
<td>1,027,370</td>
<td>0.00084</td>
</tr>
<tr>
<td>Yelp2018</td>
<td>31,668</td>
<td>38,048</td>
<td>1,561,406</td>
<td>0.00130</td>
</tr>
<tr>
<td>Amazon-Book</td>
<td>52,643</td>
<td>91,599</td>
<td>2,984,108</td>
<td>0.00062</td>
</tr>
</tbody>
</table>
Compared Methods

• LightGCN:
  • BPR optimizer, Uniform layer combination weights
  • Tuning L2 regularizer coefficient only.
• NGCF[Wang et al. SIGIR 2019]
  • Using the paper results
• Mult-VAE[Liang et al. WWW 2018]
  • Parameter setting: dropout ratio $\in \{0, 0.2, 0.5\}$, $\beta \in \{0.2, 0.4, 0.6, 0.8\}$
  • Model architecture: $600 \rightarrow 200 \rightarrow 600$
• GRMF[Rao et al. NIPS 2015]: smooth embeddings with Laplacian regularizer
  • Parameter setting: $\lambda_g \in \{1e^{-5}, 1e^{-4}, \ldots, 1e^{-1}\}$
## Experiment Results

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Gowalla</th>
<th>Yelp2018</th>
<th>Amazon-Book</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>recall</td>
<td>ndcg</td>
<td>recall</td>
</tr>
<tr>
<td>Method</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NGCF</td>
<td>0.1570</td>
<td>0.1327</td>
<td>0.0579</td>
</tr>
<tr>
<td>Mult-VAE</td>
<td>0.1641</td>
<td>0.1335</td>
<td>0.0584</td>
</tr>
<tr>
<td>GRMF</td>
<td>0.1477</td>
<td>0.1205</td>
<td>0.0571</td>
</tr>
<tr>
<td>GRMF-norm</td>
<td>0.1557</td>
<td>0.1261</td>
<td>0.0561</td>
</tr>
<tr>
<td>LightGCN</td>
<td><strong>0.1830</strong></td>
<td><strong>0.1554</strong></td>
<td><strong>0.0649</strong></td>
</tr>
</tbody>
</table>

- LightGCN achieves significant improvements over the state-of-the-art baselines → **outstanding performance**

He et al. LightGCN: Simplifying and Powering Graph Convolution Network for Recommendation. SIGIR 2020
Experiment Results

• Performance comparison between NGCF and LightGCN at different layers:

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Gowalla</th>
<th>Yelp2018</th>
<th>Amazon-Book</th>
</tr>
</thead>
<tbody>
<tr>
<td>Layer #</td>
<td>Method</td>
<td>recall</td>
<td>ndcg</td>
</tr>
<tr>
<td>1 Layer</td>
<td>NGCF</td>
<td>0.1556</td>
<td>0.1315</td>
</tr>
<tr>
<td></td>
<td>LightGCN</td>
<td>0.1755(+12.79%)</td>
<td>0.1492(+13.46%)</td>
</tr>
<tr>
<td>2 Layers</td>
<td>NGCF</td>
<td>0.1547</td>
<td>0.1307</td>
</tr>
<tr>
<td></td>
<td>LightGCN</td>
<td>0.1777(+14.84%)</td>
<td>0.1524(+16.60%)</td>
</tr>
<tr>
<td>3 Layers</td>
<td>NGCF</td>
<td>0.1569</td>
<td>0.1327</td>
</tr>
<tr>
<td></td>
<td>LightGCN</td>
<td>0.1823(+16.19%)</td>
<td>0.1555(+17.18%)</td>
</tr>
<tr>
<td>4 Layers</td>
<td>NGCF</td>
<td>0.1570</td>
<td>0.1327</td>
</tr>
<tr>
<td></td>
<td>LightGCN</td>
<td>0.1830(+16.56%)</td>
<td>0.1550(+16.80%)</td>
</tr>
</tbody>
</table>

• Training curves of LightGCN and NGCF:
Experiment Results

- **LightGCN - single**: Only use the final layer’s output

- **LightGCN-single** performs better than LightGCN on sparser datasets → further simplified
- It implies that the layer combination weights are important to tune for datasets of different properties => better to learn automatically (future work)

He et al. LightGCN: Simplifying and Powering Graph Convolution Network for Recommendation. SIGIR 2020
Embedding Smoothness

- 2-layers Light Graph Convolution:
  \[ e_u^{(2)} = \sum_{i \in N_u} \frac{1}{\sqrt{|N_u|} \sqrt{|N_i|}} e_i^{(1)} = \sum_{i \in N_u} \frac{1}{|N_i|} \sum_{v \in N_i} \frac{1}{\sqrt{|N_u|} \sqrt{|N_v|}} e_v^{(0)} \]

- Coefficient of \( e_v^{(0)} \): \( c_{v \rightarrow u} = \frac{1}{\sqrt{|N_u|} \sqrt{|N_v|}} \sum_{i \in N_u \cap N_v} \frac{1}{|N_i|} \).
  \[ \rightarrow \text{similarity between user } u \text{ and user } v \]

- Definition of user smoothness of user embeddings:
  \[ S_U = \sum_{u=1}^{M} \sum_{v=1}^{M} c_{v \rightarrow u} \left( \frac{e_u}{||e_u||^2} - \frac{e_v}{||e_v||^2} \right)^2, \]

- User and item smoothness between LightGCN-single and MF:

- LightGCN-single has lower smoothness than MF \( \rightarrow \text{ smoother embeddings are more suitable for recommendation} \)

He et al. LightGCN: Simplifying and Powering Graph Convolution Network for Recommendation. SIGIR 2020
Fast Loss

Inefficiency of BPR Loss

• Bayesian Personalized Ranking (BPR) is a widely-used pairwise loss to optimize recommender models.

• However, BPR randomly samples user-item interactions to form a mini-batch → failing to fully leverage parallel computing ability of GPU.

• BPR samples the interactions to form a mini-batch, and the data cannot form a well-structured matrix

<table>
<thead>
<tr>
<th></th>
<th>C++ (CPU)</th>
<th>TensorFlow (GPU)</th>
</tr>
</thead>
<tbody>
<tr>
<td>time/epoch</td>
<td>1.1s</td>
<td>55s</td>
</tr>
</tbody>
</table>

• We run BPRMF with amazon-book dataset using C++ on CPU(i9 9000kf) and TensorFlow on GPU(2080Ti)
Fast Loss on LightGCN

\[
\sum \sum c_{ui}(y_{ui} - \hat{y}_{ui})^2
\]

- \(c_{ui} = \alpha, y_{ui} = 1; \) if u and i have interaction
- \(c_{ui} = 1, y_{ui} = 0; \) otherwise

All possible user-item pairs

Mathematical transformations

\[
\sum \sum [(\alpha - 1)\hat{y}_{ui}^2 - 2\alpha \hat{y}_{ui}] + \sum \sum \left( \sum \sum p_{us}p_{ut}(\sum q_{is}q_{it}) \right)
\]

Historical user-item pairs

The s-th term of user embedding \(e_u\)

The s-th entry of item embedding of \(e_i\).

**Good Characteristics:**

- Can support any model of inner product structure
- Time complexity is \(O(|R|d + |N|d^2)\).
- Linear to the number of observed interactions

He et al. Fast Matrix Factorization for Online Recommendation with Implicit Feedback. SIGIR 2016
### Good Characteristics of Fast Loss

1. Distinct from BPR that samples interactions as a batch, Fast Loss samples **rows (users) as a batch**
   - The data of a batch is well-structured.

2. Adv: allow better use of the speed-up of GPU/CPU, and the computation is linear to #observations.

**Fast-loss brings 2~3 magnitude speed-up compared with BPR**

<table>
<thead>
<tr>
<th></th>
<th>BPR</th>
<th></th>
<th>Fast-loss</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>t</td>
<td>N</td>
<td>T</td>
</tr>
<tr>
<td>Gowalla</td>
<td>65s</td>
<td>840</td>
<td>15.1h</td>
</tr>
<tr>
<td>Yelp2018</td>
<td>115s</td>
<td>520</td>
<td>16.6h</td>
</tr>
<tr>
<td>Amazon-book</td>
<td>435s</td>
<td>340</td>
<td>41.8h</td>
</tr>
</tbody>
</table>

Able to train GNN on 100K users and 10M interactions in single GPU in 1 hour
Recommendation Accuracy

- LightGCN optimized with Fast Loss can achieve comparable performance to that with BPR loss.
  - Which loss is better depends on the data characteristics
  - Fast Loss seems better on long-tail users/items (we are still exploring)

<table>
<thead>
<tr>
<th></th>
<th>BPR</th>
<th>Fast-loss</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>recall@20</td>
<td>NDCG@20</td>
</tr>
<tr>
<td>Gowalla</td>
<td>0.1823</td>
<td>0.1547</td>
</tr>
<tr>
<td>Yelp2018</td>
<td>0.0640</td>
<td>0.0531</td>
</tr>
<tr>
<td>Amazon-book</td>
<td>0.0416</td>
<td>0.0311</td>
</tr>
<tr>
<td>Amazon-office</td>
<td>0.0822</td>
<td>0.0413</td>
</tr>
<tr>
<td>Amazon-cellphone</td>
<td>0.0520</td>
<td>0.0234</td>
</tr>
</tbody>
</table>

Fast loss are better
Conclusion & Future Work

• Conclusion
  • Feature transformation and nonlinear activation increase the training difficulty and hurt the model accuracy
  • Smoother embeddings are more suitable for recommendation
  • Fast-loss brings comparable performance and great efficiency improvement compared with BPR

• Future work
  • Personalize the layer combination weights $\alpha_k$
  • Streaming LightGCN for online industrial scenarios
THANK YOU!

Code(Tensorflow):  https://github.com/kuandeng/LightGCN
Code(Pytorch):  https://github.com/gusye1234/LightGCN-PyTorch