Enhancing Session-Based Recommendation with GNN: A Comprehensive Methodology Analysis

Duoer Gu
Poongkundran Thamaraiselvan
Rhianne Gonsalves
Yash Shah

1 Introduction

1.1 Background

In the e-commerce industry, there is often a large selection of products for customers to choose from. This makes it hard for consumers to find items that align with their preferences. Conventional recommendation systems often struggle to provide personalized and accurate suggestions due to their simplistic methodologies. They tend to focus on single objectives, such as predicting a user's next purchase based solely on past transactions or browsing history.

However, in today's e-commerce context with a huge variety of products easily accessible across categories, users usually complete purchasing in diverse steps, including browsing, clicking, adding items to their cart, and finalizing purchases. Traditional recommendation systems often fail to capture these multidimensional user behaviors effectively.

1.2 Problem Statement

The core challenge lies in the inefficiency of conventional recommendation systems to cater to the diverse and detailed behaviors of online shoppers. Simple objective-focused systems may overlook valuable signals and context, resulting in recommendations that do not fully meet with users' preferences and intentions. This limitation underscores the need for a more comprehensive and adaptive approach to recommendation systems that can accurately predict user actions.

1.3 Objective

The primary objective is to develop a recommender system that enhances the efficacy and relevance of recommendations of online shopping. This system aims to accurately predict users' subsequent products leveraging advanced methodologies capable of capturing the relationships between users, products, and their interactions within the online platform. Specifically, the objective includes:

- Constructing a recommendation system that utilizes graph-based models to capture contextual representations of users, products, and their interactions.
- Leveraging advanced techniques, such as Node2Vec, GraphSAGE, LightGCN to generate high-quality embeddings that preserve both local and global graph structures, enabling the system to perceive subtle patterns in user behavior and product preferences.
- Enhancing the scalability and adaptability of the recommendation system, in order to handle large-scale graphs commonly encountered in real-world e-commerce platforms, while optimizing parameters to suit the characteristics of the dataset.

In summary, the objective is to develop a robust recommender system that is good at capturing the dynamics of user-product interactions within the online shopping environment. Ultimately consumer experience is ameliorated by more accurate, relevant, and personalized recommendations.

2 Dataset Description

- 'train.parquet': This dataset contains 216,716,096 rows and 4 columns, including session IDs, AID IDs, timestamps, and event types. In each row session represent each user's ID, AID id's indicates the corresponding product, Timestamps represent events in Unix epoch format, and finally event type denoting if the action is either a 'click(0)','cart addition(1)' or 'purchase(2)'
- 'test.parquet': This dataset contains 6,928,123 rows and follows the same data structure as train.

3 Methodology

3.1 General Framework

- Extract the session(user) and product embeddings using the below mentioned graph based methods
- Employ 'Cosine similarity' to find the next 20 most relevant products for each session(user)

Now, let's dive into each of the graph based methods used

3.2 Biased Random Walk-based Embedding Generation with node2vec

In our recommendation system, Node2Vec serves as a cornerstone for capturing intricate user-product relationships in the e-commerce platform. This framework excels in learning scalable features for networks by optimizing an objective that preserves neighborhood relations in a low-dimensional space.It is similar to skipgram/Word2vec model used in NLP tasks.

- Algorithm: It is based on the idea that a node's neighbors in a graph can provide valuable information about the node itself.Node2vec uses a biased random walk technique to explore the graph and learn the representations of the nodes, by using the parameters : q and p.The intuition of Node2vec parameters p and q is that they mimic the behavior of two algorithms BFS (Breadth First Strategy) and DFS (Depth First Strategy) to explore the graph in differents ways and get differents resolutions of the network, in summary p represents the Likelihood of immediately revisiting a node in the walk, whereas q is the ratio between BFS and DFS strategies.For instance, if we want to have a high probability of transition to navigate/return back to the node, we set a low palue of p, whereas if we want to navigate farther away from the node we set a low value of q.
  - Compute random walk probabilities.
  - Simulate r random walks of length l starting from each node.
  - Optimize the node2vec objective using SGD.
- Parameters
  - p:0.2 (likelihood of immediately revisiting a node in the walk)
  - q:0.5 (balances the exploration between distant and local nodal structures)
  - embedding dimension:16 (Each node is represented with 16 features)
  - walk length:5 (The length of each random walk)
  - context size:3 (Specifies the window size surrounding the current node that will be considered when performing the embedding updates)

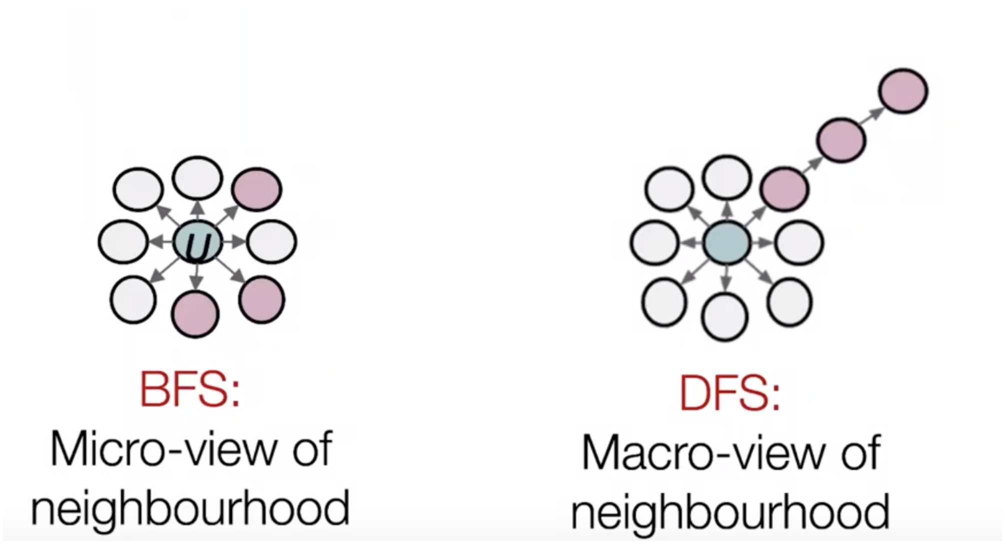


Figure 1: BFS vs DFS in Node2vec

3.3 GraphSAGE

GraphSAGE(Graph Sample and Aggregation), is a novel inductive framework that efficiently generates node embeddings for unseen nodes by leveraging node feature information and aggregating features from a node's local neighborhood. Unlike traditional methods that require re-training for new nodes, GraphSAGE learns a function to generate embeddings, allowing it to adapt to evolving graphs and generalize across different graphs. This capability is particularly useful for e-commerce platforms where new products and user interactions frequently emerge, ensuring our recommender system remains dynamic and scalable

- Node2vec Intergration:The embeddings produced by Node2Vec are used as the initial feature vectors for nodes in GraphSAGE. These embeddings capture the local and global structural information from random walks, providing a comprehensive feature set that could enhance the initial features for each nodes.
- Model Architecture: The model is setup with a two-layer Graph Convolutional Network (GCN) with specific aggregation functions:
  - First Layer: Uses mean aggregation to compute the intermediate representations of nodes.
  - Second Layer: Applies sum aggregation to produce the final node embeddings.
- Parameters
  - Input Dimension:16(node features obatained from Node2vec).
  - Hidden Layer Dimension:60
  - Embedding Dimension: 32 the size of the final node embeddings.
  - Learning Rate: 0.005, setting the speed of convergence during training.
  - Weight Decay: 1e-5, applied as part of regularization to prevent overfitting.
- Initialize Neighborloader for batch processing:
  - Number of neighbors: [10, 10] specifies the number of neighbors sampled from each layer, controlling the breadth of the neighborhood information.
  - Batch Size: 128 the number of nodes processed per iteration.
- Training Process : The training focuses on minimizing the reconstruction loss of graph edges to ensure that connected nodes produce similar embeddings, reinforcing the graph's structural integrity.
  - Epochs: Limited to 3 as its computationally intensive and extended run times required to process large graph.
  - Loss Function: Edge reconstruction loss, vital for maintaining the graph's structural characteristics.

Post training, node embeddings are extracted using the trained GraphSAGE encoder.

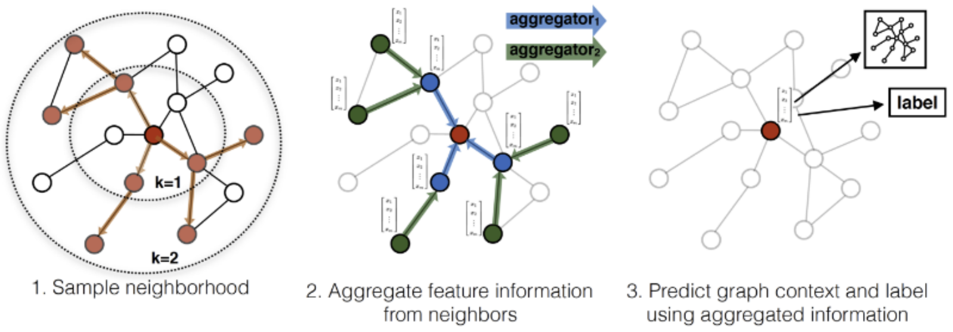

k=1
k=2
1. Sample neighborhood
aggregator$_1$
aggregator$_2$
2. Aggregate feature information from neighbors
label
3. Predict graph context and label using aggregated information

Figure 2: GraphSAGE framework

3.4 Adapted Graph Convolutional Network (GCN)

Traditional GCN models are often transductive, meaning the entire graph must be present during training. However, in our implementation we have incorporates a batching mechanism to handle large-scale graph data efficiently and also due to unavailable resources to process the whole graph. This adaptation allows the GCN to process subsets of the graph, enhancing scalability and reducing memory requirements.

We used the same Neighborloader employed in GraphSAGE for batching.

- Model Architecture
  - First Layer: Uses a spectral-based graph convolution that aggregates features from a node's immediate neighbors. This aggregation involves the normalized adjacency matrix of the graph which in essence computes a weighted average of the features of the node and its neighbors.An Exponential Linear Unit (ELU) activation follows this convolution to introduce non-linearity and aiding in complex feature representation.
  - Second Layer: Output from first layer is processed by another GCN convolution, producing the final embeddings.
- Parameters : Remain the same as in GraphSAGE model
- Batch Processing with NeighborLoader: Similar to the implementation in GraphSAGE, the adapted GCN uses the NeighborLoader to handle large datasets by loading only a subset of nodes and their respective neighborhoods at a time with the same parameters
- Training Process : The training of the adapted GCN is focused on processing subsets of the graph, which allows the model to adapt to large-scale environments without requiring the full graph in memory. The other specifics remain the same to GraphSAGE, in terms of epochs and employed loss function.
- Major differences from GraphSAGE:
  - Transductive vs. Inductive: Current GCN adaptation employs batch processing for scalability reasons, it remains fundamentally transductive as it still relies on the graph structure being static and known during training. In contrast, GraphSAGE which is specifically designed for inductive learning, can handle unseen nodes without retraining.
  - Uniform Aggregation:Our GCN uses the same type of convolution across all nodes, whilst GraphSAGE's utilizes variable aggregation functions. 'Mean' pooling in the first layer and 'Sum' in last and final layer

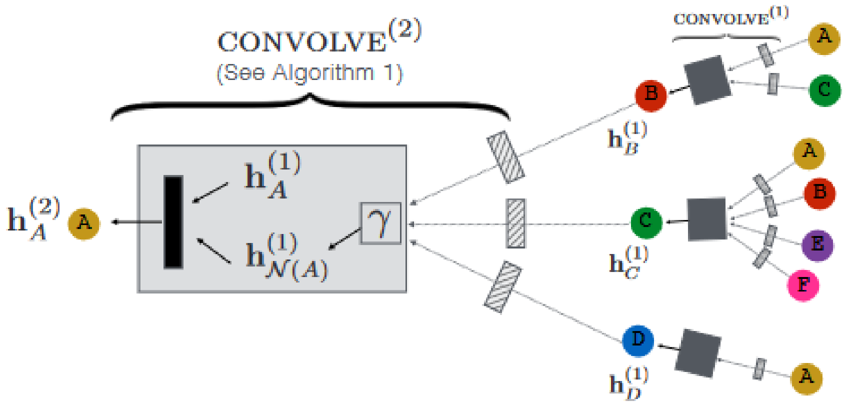


Figure 3: GCN

3.5 LightGCN

LightGCN represents a significant simplification of traditional Graph Convolutional Networks (GCNs) tailored for collaborative filtering in recommender systems. By eliminating feature transformations and nonlinear activations, LightGCN directly leverages the graph structure to aggregate neighbor information efficiently.Our implementation of LightGCN leverages the torchgeometric.nn.models.LightGCN module, which is designed to efficiently handle large-scale graph data.

- Model Architecture with Batching : We used Neighboursampler similar to previous models , allowing LightGCN to train on different portions of the graph sequentially.
- Complete graph loading requirement: Although we had employed sampling techniques, LightGCN architecture requires us to load the entire graph structure. The reason behind this necessity is to compute global properties and ensure embeddings takes into account the overall structure of the graph.
- Convolutional Layers:

  - Neighbor Sampling: The model samples up to 15 neighbors for each node making sure it preserves necessary local structures, yet reducing complexity.
  - Embedding and Aggregation: LightGCN uses a simple, layer-wise aggregation strategy discarding non linear activation and feature transformation, focusing solely on structural information to produce robust embeddings.

- Training Process : Positive link between nodes are used from the sampled nodes(NeighborSampler), whilst negative links are extracted for each of the sampled session nodes using 'structurednegativesampling' module present in torch geometric.utils package.Post that the computes a recommendation loss, contrasting the positive and negative predictions to refine the embeddings. Model finally learns to differentiate between existing and non-existing links.

- Challenges faced in LightGCN:
  - Memory constraints: Since we had to load the entire graph(edge index tensor storing the indices of session to products connections), it occupied most of the memory when loaded into the GPU. When it came to training, we faced issues with CUDA going out of memory multiple times
  - Mitigation attempts: We tried several alternative solution to the circumvent the issue, a few were: used parallel GPU's, implemented LightGCN from scratch to simplify the model, but unfortunately the training phase wasn't successful due to computational limitations.

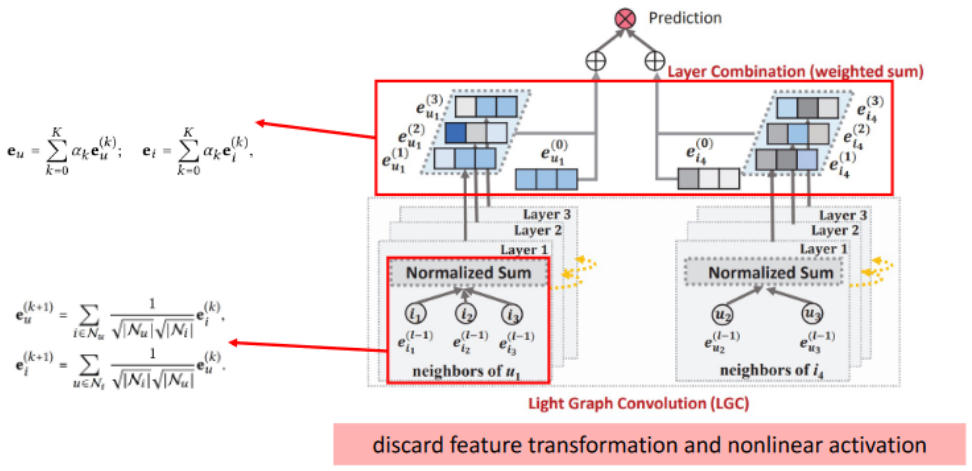

Prediction
Layer Combination (weighted sum)
$\mathbf{e}_u = \sum_{k=0}^{K} \alpha_k \mathbf{e}_u^{(k)}$; $\mathbf{e}_i = \sum_{k=0}^{K} \alpha_k \mathbf{e}_i^{(k)}$,
$\mathbf{e}_u^{(k+1)} = \sum_{i \in N_u} \frac{1}{\sqrt{|N_u|}\sqrt{|N_i|}} \mathbf{e}_i^{(k)}$,
$\mathbf{e}_i^{(k+1)} = \sum_{u \in N_i} \frac{1}{\sqrt{|N_i|}\sqrt{|N_u|}} \mathbf{e}_u^{(k)}$.
Layer 3
Layer 2
Layer 1
Normalized Sum
neighbors of u_1
neighbors of i_4
Light Graph Convolution (LGC)
discard feature transformation and nonlinear activation

Figure 4: LightGCN

3.6 Recommendation phase using the extracted embeddings

We utilized a efficient approach to recommend products based on embeddings generated by GCN, GraphSAGE, and Node2Vec as stated in the general framework at the beginning of methodology. To fasten the retrieval of similar products , we employ Annoyindex, a tree based algorithm for optimized retrieval speeds in finding nearest neighbors using cosine similarity in high dimensional spaces. We have 2 different strategies to recommend products based on the length of the session.

- Sessions with Fewer than 20 Products: In this case, we use Annoyindex technique to find the relevant products for each session.The combined list of original session products and the newly recommended products from Annoy Index forms the final recommendation list, as the objective is to predict the next 20.

- Sessions with 20 or More Products: We leverage a weighted scoring system to rank the products.We calculate the weights of each product in the session using a logarithmic scaling of weights (np.logspace) of base 2, which emphasizes more recent interactions.Products are then ranked based on these adjusted weights, and the top 20 products are selected for recommendation

- Reasoning behind the stated logic:
  - We tried to utilize the embeddings irrespective of length of each session(number of products), but it resulted in worser recall@20 score which is the evaluation metric. For instance, in Node2vec, it resulted in a score reduction from 45 to 27.

4 Evaluation

Basic intuition of Recall: In recommendation systems, recall measures the proportion of relevant items that are successfully retrieved by the system. Specifically, recall quantifies how many of the relevant items are captured in the recommendations out of all possible relevant items. This is especially important in environments where the goal is to ensure that no important items are overlooked, such as in our case of product recommendations.

Evaluation is based on computing Recall@20 for each of clicks, carts, and orders after which the three recall values are weighted-averaged. Recall at 20 is the proportion of relevant items found in the top-20 recommendations

$$\text{score} = 0.10 \cdot R_{\text{clicks}} + 0.30 \cdot R_{\text{carts}} + 0.60 \cdot R_{\text{orders}}$$

where R is calculated as

$$R_{\text{type}} = \frac{\sum_{i=1}^{N} \left| \text{predicted aids}_{i,\text{type}} \cap \text{ground truth aids}_{i,\text{type}} \right|}{\sum_{i=1}^{N} \min\left(20, \left|\text{ground truth aids}_{i,\text{type}}\right|\right)}$$

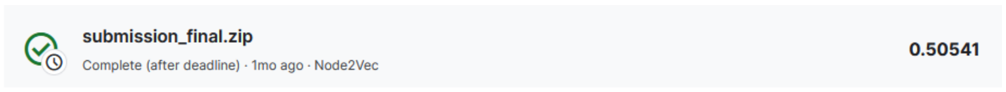
Note: In each session(user's activity), the true outcome for clicks consists of just one value: the subsequent product that the user clicks on. While predictions can include up to 20 potential product, the actual click event is a single item. Conversely, for carts and orders, the ground truth encompasses every product value that is added to a cart or purchased within the session.

In our case, since we didn't employ different weights based on type of interactions(click,add to cart or purchase). Therefore, the same 20 products were recommended for all 3 categories.

Reasons: Although we planned to employ different weighting to each session-product interaction depending on its type and recommend products, due to our computational limitations most of out time frame went in figuring out ways to process large graphs.

5 Results

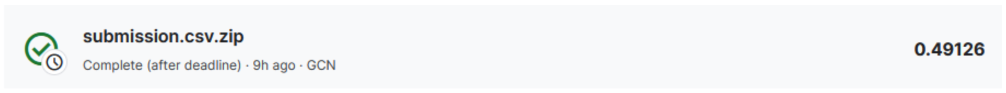
- Node2vec: The Node2vec model achieved a recall score of 0.50, which is a respectable outcome given the model did not factor in the type of interactions(clicks, cart additions or purchases). The top recorded score is 0.60 as this dataset was part of Kaggle competition named 'OTTO – Multi-Objective Recommender System'.



submission_final.zip
Complete (after deadline) · 1mo ago · Node2Vec
0.50541

Figure 5: Node2vec recall score

- GraphSAGE and GCN: Both the models produced almost the same score of 0.49 slightly lower than Node2vec, which was contrary to expectation. This result is interesting as it deviates from the initial hypothesis that leveraging Node2vec embeddings as initial points would enhance model performance
- Reasons:
  - Effectiveness of Initial Embeddings: The strategy of using Node2vec embeddings to initialize GraphSAGE and GCN did not produce the expected improvement in performance. This suggests that while Node2vec provided a good foundation, the additional complexity and nuances of the dataset might require more refined to capture the complex interactions.
  - Computational Intensity and Training Time:Major limiting factor hindering further improvements is the inherent computational demands and increased training time per epoch required by GraphSAGE and GCN for large graphs. Therefore we had trained it only for 3 epoch. Increasing the epochs could have yielded stable estimates and ultimately better recall scores.
  - Convergence Issues with the usage of NeighborSampler: An another factor impacting the models' performance was the use of NeighborSampler, technique that we used to construct batches by sampling a fixed number of neighbors. This method which was crucial in the first place to train the model, might have hindered the models' ability to converge to an optimal solution. The stochastic nature did not provide comprehensive coverage of the graph's global context, leading to insufficient.


submission.csv.zip

Complete (after deadline) · 9h ago · GCN

0.49126

Figure 6: GCN recall score

6 Conclusion

Our project illuminates various aspects in utilized graph based methods in the realm of recommeder systems. A major challenge was the initial struggle to find a way to train a graph based model on a large dataset, which is often the case in real world scenarios where there are numerous users and their respective preferences. Understood how training can be achieved by sampling parts of graph, yet this approach carries its own set of limitations.

Another aspect we explored was learning different graph based frameworks such as Node2vec, GraphSAGE, GCN and LightGCN. Although, we didn't manage to train the LightGCN model, we gained significant insights into its internal working and how to optimize and train given our limited computational resources.

In terms of results, out first model-Node2vec produced the best result contrary to our expectations at the start.On the flip side, training the other models(GraphSAGE and GCN) to its convergence would have improved the recall score by at least a certain margin.

Future directions would certainly involve implementing our initial plan to devise these models accounting for the type of interactions (clicks, cart additions, and purchases). We plan to search for methods to optimally implement LightGCN with limited computational resources. Additionally, we intend to incorporate the timestamp information provided in the dataset to add another layer of temporal complexity to enhance the recommendations. We will also explore approaches such as UltraGCN, which has garnered interest more recently, and consider developing hybrid models later.

7 References

Grover, A. and Leskovec, J., 2016, August. node2vec: Scalable feature learning for networks. In Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining (pp. 855-864)

Hamilton, W., Ying, Z. and Leskovec, J., 2017. Inductive representation learning on large graphs. Advances in neural information processing systems, 30.

He, X., Deng, K., Wang, X., Li, Y., Zhang, Y. and Wang, M., 2020, July. Lightgcn: Simplifying and powering graph convolution network for recommendation. In Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval (pp. 639-648).

Ying, R., He, R., Chen, K., Eksombatchai, P., Hamilton, W.L. and Leskovec, J., 2018, July. Graph convolutional neural networks for web-scale recommender systems. In Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining (pp. 974-983).

Rossi, E., Chamberlain, B., Frasca, F., Eynard, D., Monti, F. and Bronstein, M., 2020. Temporal graph networks for deep learning on dynamic graphs. arXiv preprint arXiv:2006.10637.

Mao, K., Zhu, J., Xiao, X., Lu, B., Wang, Z. and He, X., 2021, October. UltraGCN: ultra simplification of graph convolutional networks for recommendation. In Proceedings of the 30th ACM International Conference on Information & Knowledge Management (pp. 1253-1262).

Zhang, M., Wu, S., Yu, X., Liu, Q. and Wang, L., 2022. Dynamic graph neural networks for sequential recommendation. IEEE Transactions on Knowledge and Data Engineering, 35(5), pp.4741-4753.