Link Prediction: Product Recommendation for Instacart

CSE 6240 Web Search and Text Mining Project Final Presentation

Kien Tran, Tanmay Kenjale, Yu-Ching Chen, Chase Harrington
Group 1

Georgia Tech

Agenda

1. Introduction
2. Dataset
3. Methods
4. Experiments and results

Introduction

Introduction and Motivation

- We aim to use the Instacart dataset to develop a user-product recommendation system

- Frame problem as a bipartite graph link prediction problem
  - Two classes of nodes representing users and products
  - Edges between a user node and a product node represent a user purchasing a product

- Successfully predicting edges between nodes, and recommending appropriate items, can increase user satisfaction and potentially increase app usage

Dataset

Dataset Introduction

- [Kaggle – Instacart Market Basket Analysis]
- Our data consists of purchases made by users on Instacart
- 49K product nodes and 101K user nodes
- 9.3 million user-product pairs (edges)
- Data is grouped into orders that each contain a user and the products purchased in the order
- Data contains information about purchase time, time since previous order, add-to-cart sequence, and product repurchase

Georgia Tech

Data Preprocessing

- **Step 1:** Remove users with fewer than 10 purchases
- **Step 2:** Generate user features by aggregating order data (RFM - Recency, Frequency, Monetary) and generate item features by creating text embeddings from product descriptions
- **Step 3:** Employ temporal splitting and only keep new links as positive labels
  - Label edges: The test label contains only the new product in the most recent basket (t) for each user, the validation label in the second most recent basket (t-1) for each user, and the train label in the third most recent basket (t-2) for each user. These are positive labels.
  - Graph edges: All products bought before the basket that was used as labels
- **Step 4:** Generate negative samples with rate = 1, meaning for each user, we generate a random negative sample for each positive labels in the label edges described above.

Methods

Georgia Tech

Models

1) Matrix Factorization (MF) - baseline

2) Node2vec - baseline

3) GraphSAGE

4) Graph Attention Network (GAT)

Matrix Factorization (MF)

- Use train set of positive and negative samples to create a sparse adjacency matrix
  - $A_{u,p}$=1 if an edge exists between user u and product p, 0 otherwise
- Implement Stochastic Gradient Descent on the adjacency matrix to generate 128-dimensional user and product embeddings
- For each edge in the test set, perform a simple cross-product between the user and product embedding to generate a predicted score for the user-product pair
  - If the predicted score is greater than a predetermined threshold value, then we predict a link; otherwise, we do not predict a link
  - We define the threshold value 0<t<1 such that the performance on the test data is maximized; t=0.53
- ROC-AUC score on test data: 0.822

Node2vec

- Trained a Node2vec model for embedding each node
  - Set p=1, q=2 because we expect users would be more interested in neighbor items
  - Set the dimension of embedding as 128
- Generated link embeddings by concatenating Node2vec node embeddings for each link
  - The dimension of link embeddings is 256
- Trained an MLP Classifier to predict the edges existence
  - Using the pre-processing train/val/test set of positive and negative edges
- ROC-AUC score on the test data: 0.935

GraphSAGE vs Graph Attention Network (GAT)

GraphSAGE

- GraphSAGE learned a general transformation function to aggregate information from neighboring nodes and pass-on the processed information to the next layer.
- With more than 2 layers of SAGEConv, the model should be able to learn the patterns equivalent to any collaborative filtering approaches (e.g. user – item – other user).
- GraphSAGE is also easy to train since it take a sample from the neighboring nodes instead of the whole graph.
- We use an implementation of Hamilton et al (2017) in the pytorch-geometric library.

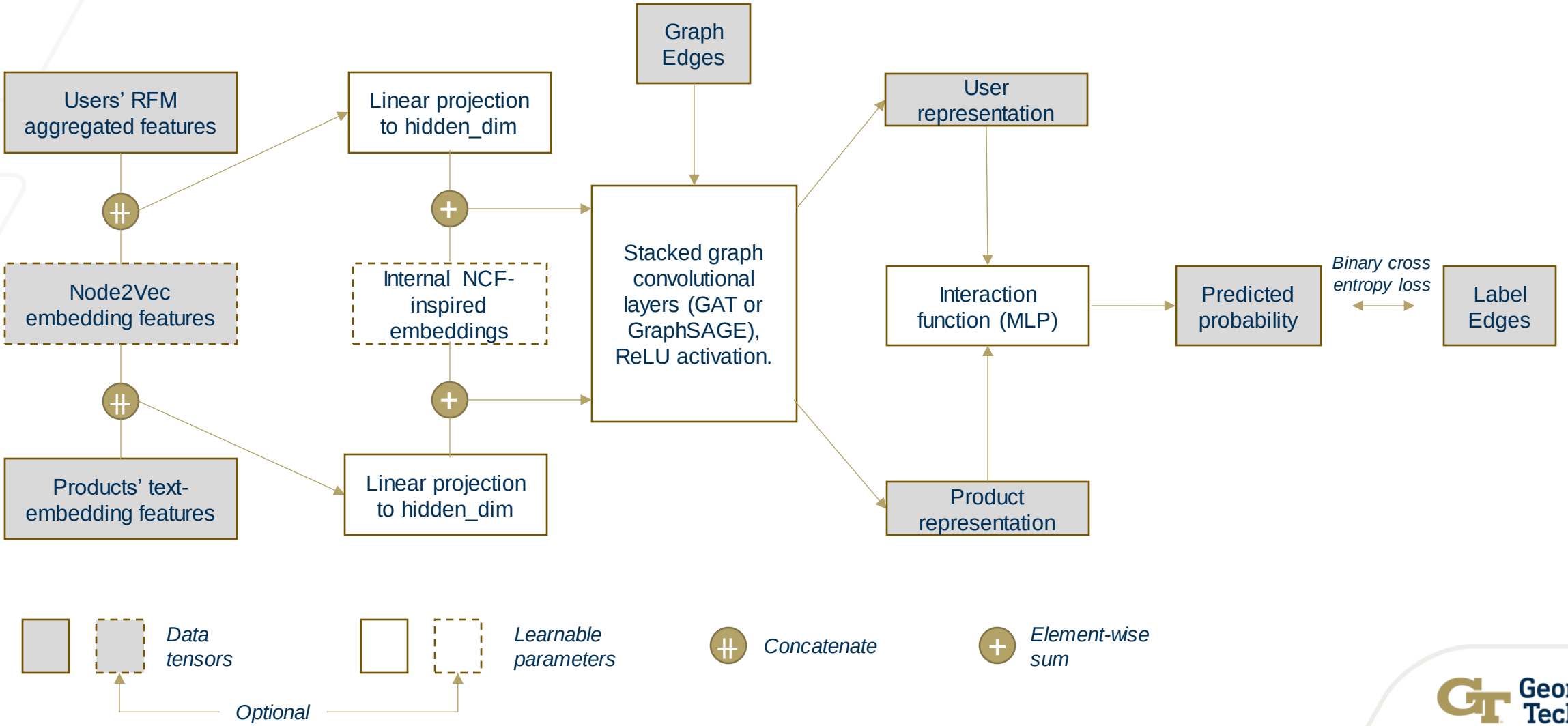
Graph Attention Network

- Similar to GraphSAGE, Graph Attention Convolutional layer (GATConv) takes information from neighboring nodes, process it, and pass on to the next layer.
- The difference is GATConv can use the attention mechanism to learn the importance of neighboring nodes to process accordingly.
- In addition, using multi-head also enable learning more complex patterns => Overall GAT is a more flexible network.
- We use the implementation created by Brody et al (2021), where they generalize the original work from Morris et al (2018) and demonstrated superior performance

We chose GraphSAGE and GAT to implement due to their end-to-end nature, which in theory is superior compared to Node2Vec models, and the flexibility to customize the interaction function, which is superior compared to the matrix factorization approach.

Georgia Tech

Architecture of GNN models for link prediction



Experiments and results

GNN models experiment setup

Experiments	Model specifications	Training specifications

#1: Test option combinations	• Using GraphSAGE, mean agg, fixed convolutional Depth = 3 and Hidden Size = 256.
	• **Architecture options**:
	  • *Vanilla GraphSAGE*
	  • *GraphSAGE + NCF-inspired Embeddings* (add learnable user and item embeddings to increase expressiveness).
	• **Feature options**:
	  • *Baseline Features Approach:* Only input generated user and product features.
	  • *Advanced Features:* Concatenate with additional features from Node2vec.
		Early stopping: when validation roc_auc is worse than previous epoch
		
		Hardware: Google Colab GPU (15 GB of GPU RAM)

#2: GraphSAGE tuning	• **Depth and Size tuning**:
	  • Depth - number of graph convolutional layers: [2, 3]
	  • Size - hidden layer dimension: [128, 256, 512]	

#3: Graph Attention Network tuning	• **Size and Attention heads tuning**:
	  • Size - hidden layer dimension: [128, 256, 512]
	  • Number of attention heads: [1, 2, 4, 8]
		Hardware: Google Colab premium GPU (40 GB of GPU RAM).

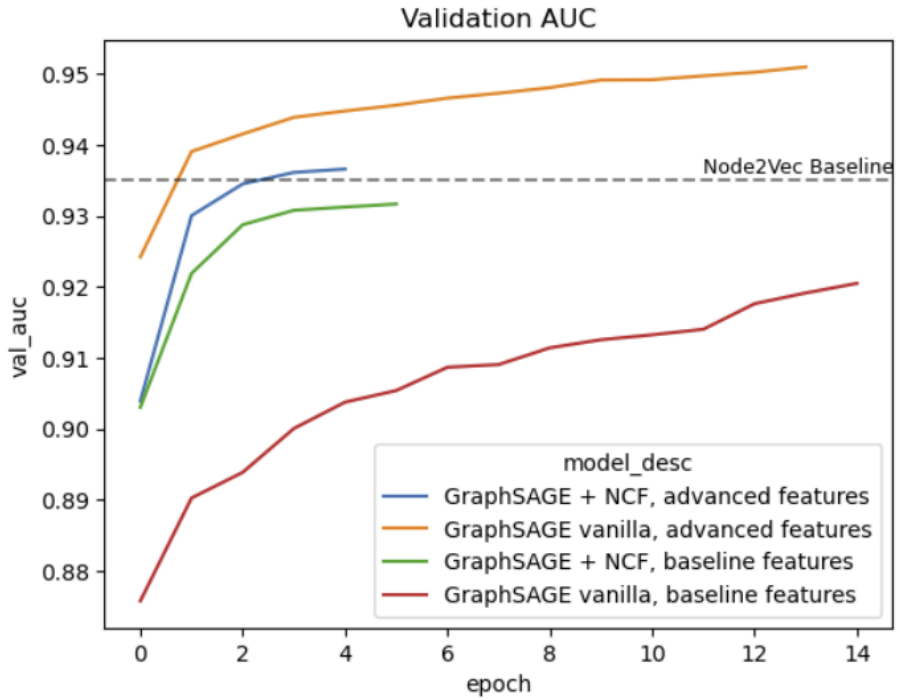
Evaluation metrics: ROC-AUC (main), and other classification metrics: F1-score, Precision, Recall

Ranking metrics such as MRR, MAE were **not** used for evaluation due to the high complexity of predicting and ranking all 50,000 products for each user.

Georgia Tech

Experiment #1 results

	Baseline Features	Advanced Features
Vanilla GraphSAGE	AUC = 0.920	**AUC = 0.951**
GraphSAGE + NCF Embeddings	AUC = 0.932	AUC = 0.937



- Vanilla GraphSAGE + Advanced features seems to be the best configuration. This might stem from the random walk help incorporate long-distance information.
- GraphSAGE + NCF embedding models fit very well and very quick to the training set. But they also easily overfit and stop in just 3-4 epochs.
- Vanilla GraphSAGE is by far the worst performer, not even achieve Node2Vec Baseline performance.
- We also found that the models' performance on validation set closely resemble their performance on the test set.

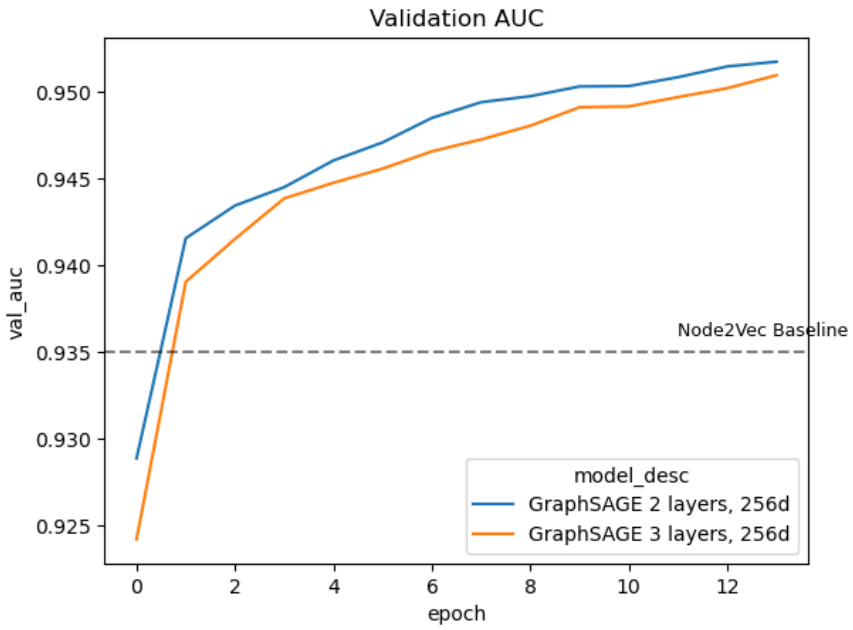
Georgia Tech

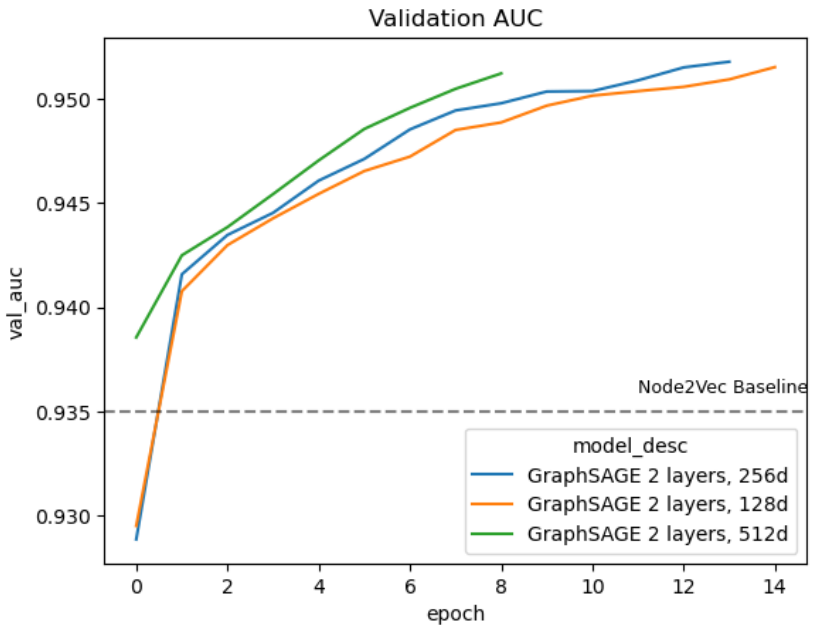
Experiment 2 result: GraphSAGE Depth and Size

We tuned the chosen Vanilla GraphSAGE model w/ Advanced Features further by optimizing the number of layers and the hidden size

# Layers (Hidden = 256)	AUC
2	**0.952**
3	0.951

Hidden Size (# Layers = 2)	AUC
128	0.951
256	**0.952**
512	0.951



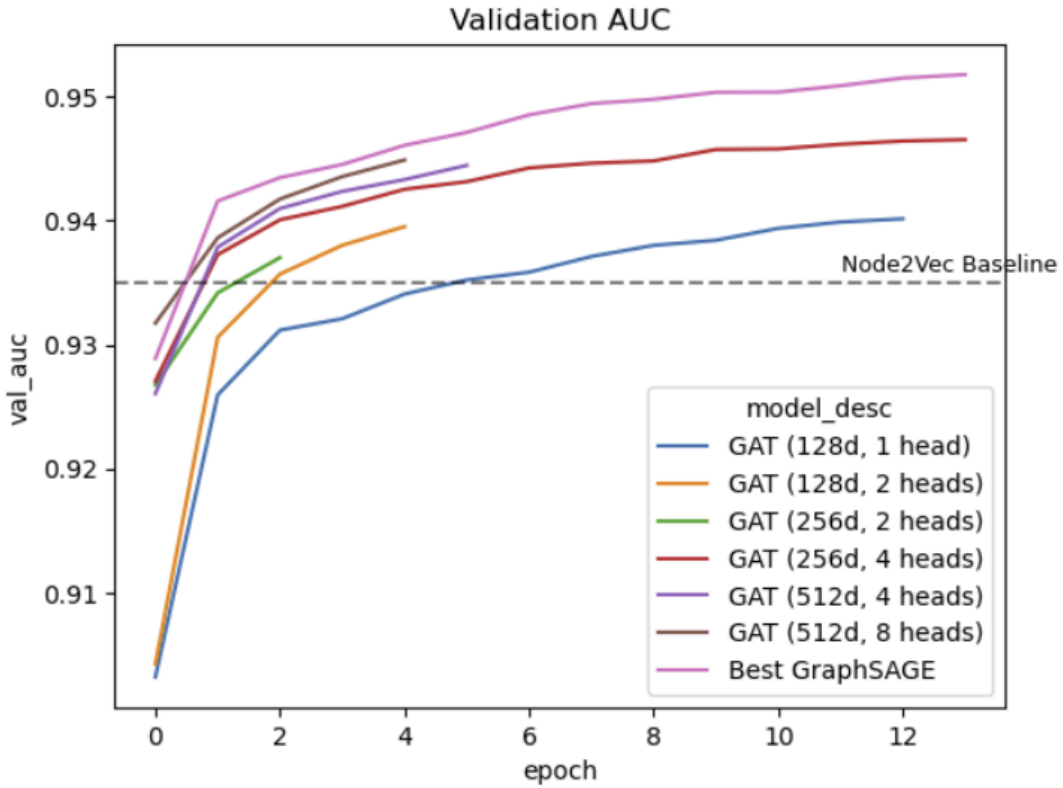


- Two layers and Hidden Size = 256 is the best configuration
- Hidden Size = 128 train faster but slightly more overfit to the train set => worse performance on the validation set

Georgia Tech

Experiment 3: Graph Attention Network (GAT)

We chose to use a Vanilla GAT with advanced features, since we found this worked well with GraphSAGE



- Best GAT had hidden size of 256 and 4 attention heads
- Other GAT configurations stop much earlier with lower performance, which indicate they have lower expressiveness and/or overfit more to the training set
- The best GraphSAGE model outperformed the best GAT model by a large margin

Georgia Tech

Results and Future work

Results

Models	Precision	Recall	F1-measure	AUC scores
MF	0.825	0.819	0.822	0.822
Node2vec + MLP	0.860	0.864	0.862	0.935
GraphSAGE	**0.863**	**0.912**	**0.889**	**0.952**
GAT-based	0.854	0.905	0.879	0.947

- GraphSAGE performs the best across all metrics
- The sparsity of the adjacency matrix is the limitation for MF
- Node2vec can achieve a decent performance, but it only consider the graph structure
- Graph convolutional methods outperform the baseline models, MF and Node2vec

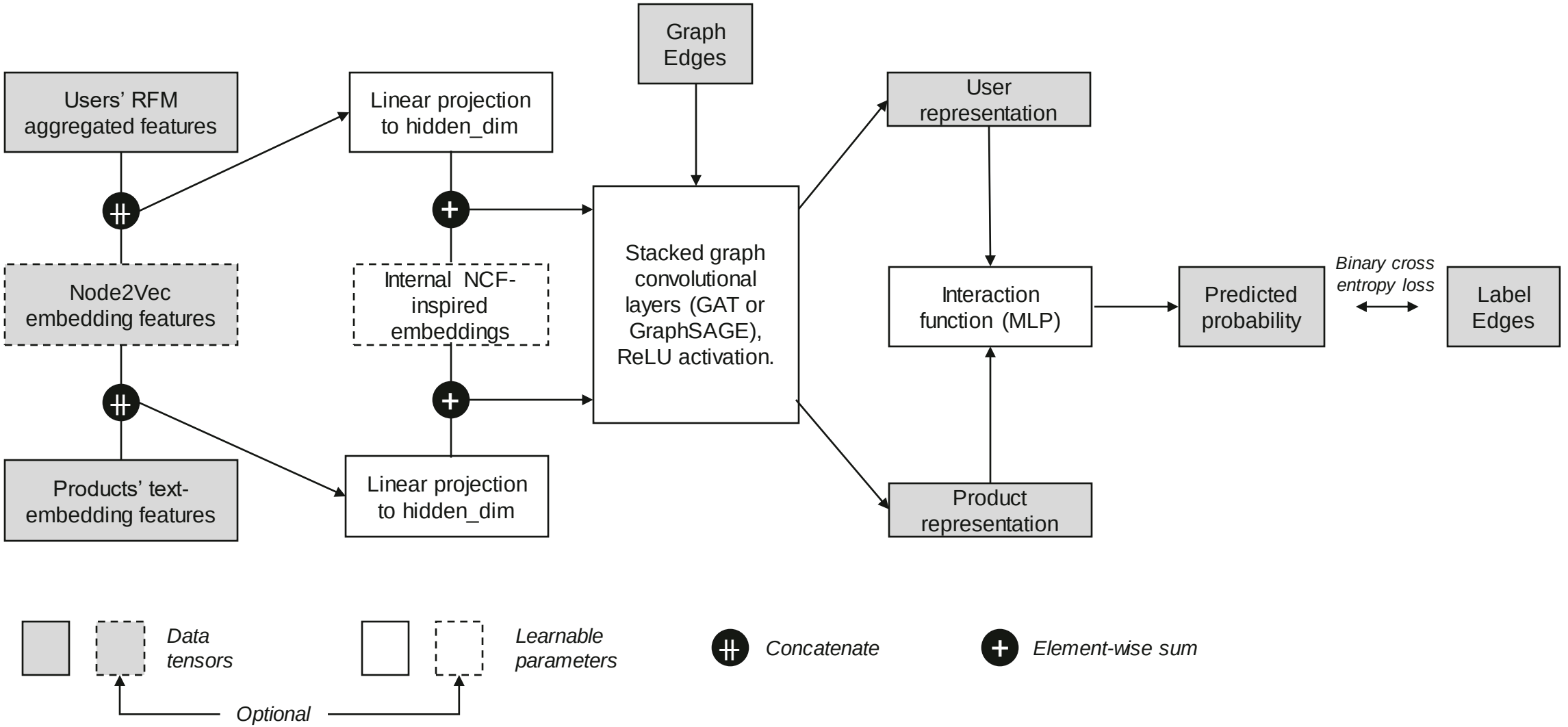
Insights and Future works

- Good input features is pivotal in improving performance of Graph convolutional models => More extensive feature engineering.
- GAT's performance was not as good as GraphSAGE => Additional experimentation with more expressive GAT model and further hyper parameter tuning.
- Combining normal features with Node2Vec embedding features achieve the best results so far, we hypothesized that it's due to long-distance information in the random walks => Worth exploring other architectures that can incorporate both local and long-distance information

Georgia Tech

Thank you for your listening

Georgia Tech


Graph Edges
Users' RFM aggregated features
Linear projection to hidden_dim
User representation
Node2Vec embedding features
Internal NCF-inspired embeddings
Stacked graph convolutional layers (GAT or GraphSAGE), ReLU activation.
Interaction function (MLP)
Predicted probability
Binary cross entropy loss
Label Edges
Products' text-embedding features
Linear projection to hidden_dim
Product representation
Data tensors
Learnable parameters
Concatenate
Element-wise sum
Optional