Budapest University of Technology and Economics
Faculty of Electrical Engineering and Informatics
Department of Telecommunications and Media Informatics

Efficient embedding of new nodes into existing embedding spaces

Scientific Students' Association Report

Author:

Richárd Kiss

Advisor:

dr. Gábor Szűcs

2022

Contents

Chapter 1

Introduction

Node embedding is the process of mapping nodes of a graph to a vector space such that it preserves structural and positional information of the nodes. Node embedding has many real world applications, for example it is used extensively in recommendation systems, natural language processing, or even for modeling protein-protein interactions.

In our work we focus on the recommendation problem only, specifically the real-time collaborative filtering. The recommendation task can be formulated as a link prediction problem in a bipartite graph with node types user and item. In real-time collaborative filtering task recommendations must be made for users at the time the user interacts with the application. Capturing changes in user behavior real-time poses a great challenge. Not only accurate model learning is important, but adapting the model to changes quickly and fast inference are also important aspects. In production environments the number of items and users in the system often is in the million or even billion scale.

1.1 Real-time collaborative filtering

Traditional (transductive) collaborative filtering approaches like matrix factorization [18][3] or latent semantic analysis [16] are widely used in large scale recommendation systems however, they are not directly applicable in the real-time personalization task. If enough data is collected these methods can accurately model user behavior in the global scale - recommendations are generally accurate - but for the individual users it can become less and less accurate as changes in user behavior occur. These approaches require re-learning the latent representations every once in a while to adapt to behavior shifts, but because the training process is computationally expensive it can not be done real-time.

Adaptation to continuous change in user behavior is a challenging task. Usually, in machine learning there is a trade-off between model accuracy and computational cost. This is no different in real-time recommendation systems. In the real-time setting pre-computing recommendations is not an option, since we want our model to capture changes in user behavior (which is observed through interactions). Evaluation of a complex scoring model is not feasible for each user-item pair in real-time. Thus, in such scenarios the recommendation process is decomposed to two phases, candidate generation and scoring. In the candidate generation phase a fast algorithm is used to narrow down the set of possibly relevant items to several hundred or thousand. Then a more complex model can be evaluated on the candidate pool to select the top-k items that are most likely to be relevant for the user. In this current work we only discuss research to improve the candidate generation phase.

There are two main challenges in the candidate generation process. Firstly, it is necessary to have a fast model that can retrieve relevant items real-time. Secondly, the selected items must maximize recall. No matter how accurate the scoring model is, a relevant item that is not included in the candidate pool will not appear in the recommendation (in the scoring phase only items from the candidate pool are considered).

1.1.1 Fast candidate generation

Modeling user-item interactions by the dot product or cosine-similarity of their latent representations is a proven method in collaborative filtering [3]. Fast approximate and exact algorithms exist for finding vectors that maximize dot product (or cosine similarity) for a given query vector [17]. These algorithms can be easily integrated with node representation learning techniques that model node proximity as dot product of their representations for the candidate generation task.

1.1.2 Own contribution

In our work we aim to build a real-time collaborative filtering algorithm on implicit feedback data that can scale to large user and item bases. We employ transductive node representation learning techniques to obtain meaningful initial representations and use them to train an inductive representation learning model [11] that we can continuously use to update representations as new observations are streaming in. Our algorithm takes the latest observations into consideration when generating recommendations, the items shown to users reflect their latest actions as well as their earlier behavior.

Chapter 2

Theoretical background

2.1 Definitions and notation

To introduce the concepts and algorithms we first declare some necessary definitions and notations.

Let $G = (V_{user} \cup V_{item}, E)$ be a simple undirected bipartite graph with two node types *user* and *item*, also referred as user-item interaction graph. If an interaction is observed between user u and item i, then $(u, i) \in E$.

$\mathcal{N}(z)$ is the set of neighboring nodes of node z in graph G.

Adjacency matrix A_G is a binary square matrix that represents the connections of the graph G:

$$A_G[u, i] = \begin{cases} 1, \text{if}(u, i) \in E \\ 0, \text{otherwise} \end{cases} \tag{2.1}$$

Usually we omit G from the notation as the graph we are referring is known from the context, we simply write A.

As in the recommendation task the graph is bipartite, the adjacency matrix can be constructed as a block matrix:

$$A = \begin{bmatrix} 0 & R \\ R^T & 0 \end{bmatrix} \tag{2.2}$$

where R is a $|V_{users}| \times |V_{items}|$ binary matrix called interaction matrix in the literature. $R_{u,i} = 1$ if user u has interacted with item i or 0 otherwise.

Node degree matrix D is a diagonal square matrix with node degrees in the diagonal:

$$D[i, j] = \begin{cases} |\mathcal{N}(i)|, \text{if} i = j \\ 0, \text{otherwise} \end{cases} \tag{2.3}$$

Graph Laplacian L matrix is defined as $D - A$. Usually a normalized version of the Laplacian is used since it has nicer mathematical properties [14]: $L = D^{-\frac{1}{2}} (D - A) D^{-\frac{1}{2}}$

Node representation matrix. The node representation matrix $X \in \mathbb{R}^{|V| \times k}$ holds k dimensional representations for each node in the graph. For simplicity we refer to the representation of node z as $x_z = X[z]^T$ for convenience.

2.2 Node representation learning

Node representation learning or node embedding is a mapping from nodes in a graph to dense vector representations (embeddings) such that structural and positional information is encoded in the embeddings. Node embedding can be used in downstream machine learning tasks in which relational information is important in the modeling process.

There are three main node representation learning approaches: factorization based, random walk based and deep learning based approaches [29]. Furthermore, algorithms can be either inductive or transductive. With transductive algorithms one can obtain embeddings for a static graph however, if the graph is dynamic - nodes and edges are inserted by time - it is hard to adapt the embeddings to capture these changes. Usually a training process must be performed to obtain the updated embeddings. Inductive approaches on the other hand can easily adapt to these changes.

In our work we experimented with the following node representation learning algorithms:

Table 2.1: Node representation learning algorithms covered in this work.

Algorithm	**Approach**	**Inductive**
GraphFactorization	factorization	✗
RandNE	factorization	✗
Node2Vec	random walk	✗
GraphSAGE	deep learning	✓

2.2.1 Factorization based approaches

Factorization based node representation learning algorithms either factorize a polynomial function of the graph Laplacian or directly the adjacency matrix [4].

Graph Factorization

Simple Graph Factorization [2] (GF) factorizes the adjacency matrix - or equivalently GF models user-item interactions as dot-product of their latent representations. The objective function is as follows:

$$\min_X \frac{1}{2} \sum_{(u,v)\in E} (A_{u,v} - \langle X_u, X_v \rangle)^2 + \frac{\lambda}{2} \sum_i ||X_i||^2 \tag{2.4}$$

where λ is a regularization constant. The rows of $X \in \mathbb{R}^{|V|\times k}$ are dense vector representations of nodes and each column corresponds to one latent feature. Here k is the dimensionality of our latent representations. Non-zero entries of A are approximated with XX^T, where X can be constructed as $X = \begin{bmatrix} U \\ I \end{bmatrix}$, U holds representations of nodes with type user and I holds item type node representations (this composition requires us to sort rows of A so that each $n_1 \in V_{user}$ proceeds $n_2 \in V_{item}$). Using this notation the objective function (Equation 2.4) can be rewritten as:

$$\min_{U,I} \frac{1}{2} \sum_{(u,i)\in E} (R_{u,i} - \langle I_i, U_u \rangle)^2 + \frac{\lambda}{2} \left(\sum_x ||U_x||^2 + \sum_y ||I_y||^2 \right) \tag{2.5}$$

If we fix U, then the objective function is a convex function of I and vice versa. This allows us to find the optimum analytically by setting the gradient to 0. We can compute optimal U with fixed I, then fix U and find optimal I, repeat until convergence. This algorithm is called *Alternating Least Squares* [8] (ALS):

Algorithm 1 Alternating Least Squares

1: Initialize U with small random values
2: **repeat**
3: **for** $i \in V_{item}$ **do** ▷ Fix U and solve for I
4: $I_i \leftarrow \left(\sum_{u \in \mathcal{N}(i)} U_u U_u^T + \lambda \mathbb{I} \right)^{-1} \sum_{u \in \mathcal{N}(i)} R_{u,i} U_u$
5: **end for**
6: **for** $u \in V_{user}$ **do** ▷ Fix I and solve for U
7: $U_u \leftarrow \left(\sum_{i \in \mathcal{N}(u)} I_i I_i^T + \lambda \mathbb{I} \right)^{-1} \sum_{i \in \mathcal{N}(u)} R_{u,i} I_i$
8: **end for**
9: **until** convergence

where $\mathbb{I}$ is the identity matrix. This algorithm can be easily parallelized and is often used due to its fast convergence and numerical stability [28].

Iterative Random Projection Network Embedding

The problem with graph factorization is that it only considers first order node proximities. Previous works [7][27] have shown that high-order proximities are essential to be preserved in network embedding, which can be formulated as a polynomial function of the adjacency matrix or the graph Laplacian:

$$S = \alpha_0 \mathbb{I} + \alpha_1 A + \alpha_2 A^2 + ... + \alpha_q A^q \tag{2.6}$$

where S is the high-order proximity matrix and $\alpha_0, \alpha_1, ..., \alpha_q$ are pre-defined weights. In [30] the authors propose a method which learns node embeddings by optimizing the following objective function:

$$\min_X ||SS^T - XX^T||_2 \tag{2.7}$$

where $X \in \mathbb{R}^{|V| \times k}$ holds representations for each node in the graph. To minimize the objective function the authors use *Gaussian random projection*:

$$X = SP \tag{2.8}$$

where $P \in \mathbb{R}^{|V| \times k}$ is a *Gaussian random matrix* (each element of P follows an i.i.d normal distribution). It can be proven that Gaussian random projection can efficiently minimize the objective function in Equation 2.7 [26]. The problem with factorizing a high-order proximity matrix is that it is no longer sparse, making it impossible to store in memory for larger graphs. The authors suggest an iterative approach to avoid explicitly calculating higher powers of the adjacency matrix. X can be decomposed as:

$$X = \alpha_0 X_0 + \alpha_1 X_1 + \alpha_2 X_2 + ... + \alpha_q X_q \tag{2.9}$$

where $X_i = A^i P$. This way each X_i can be calculated iteratively, using only X_{i-1} (which is $|V| \times k$) and A (which is sparse):

$$X_i = AX_{i-1} \tag{2.10}$$

Algorithm 2 Iterative Random Projection Network Embedding [30]

1: Generate $P \in \mathbb{R}^{|V| \times k}; P_{i,j} \sim \mathcal{N}(0, \frac{1}{k})$
2: Perform *Gram Schmidt* orthogonalization on P to obtain the orthogonal projection matrix X_0
3: **for** i in 1..q **do**
$X_i = AX_{i-1}$
4: **end for**
5: $X = \alpha_0 X_0 + \alpha_1 X_1 + \alpha_2 X_2 + ... + \alpha_q X_q$

Orthogonality of the projection matrix is necessary to more accurately approximate optimal solutions for the objective function [30].

2.2.2 Random walk based approaches

Random walk based approaches covered in this work use noise contrastive estimation [12] to learn high-order node representations.

DeepWalk

DeepWalk [19] generalizes language modeling and unsupervised feature learning from sequences of words to graphs. It does so by sampling random walks from the graph and treating them as sentences where the nodes correspond to words. It applies the *Skip-Gram with Negative Sampling* (SGNS) algorithm on random walk samples to learn node embeddings. SGNS is a language model that maximizes the co-occurrence probability $p\left(w_1, w_2|X\right) = \sigma\left(x_{w_1}^T x_{w_2}\right)$ among the words that appear within a specified context window in a sentence:

$$\max_X \mathop{\mathbb{E}}_{(w,c)\sim\xi} \log(\sigma(x_w^T x_c)) + \mathop{\mathbb{E}}_{(w,c)\sim\xi'} \log(\sigma(-x_w^T x_c)) \tag{2.11}$$

where w is a word and c is its context, ξ is the training distribution, ξ' is the noise distribution. DeepWalk implicitly factorizes the following matrix [20]:

$$\log\left(\text{vol}(G)\left(\frac{1}{T}\sum_{k=1}^{T}\left(D^{-1}A\right)^k\right)D^{-1}\right) - \log(\beta) \tag{2.12}$$

where $\text{vol}(G) = \sum_i D_i$ is the volume of graph G, $D^{-1}A$ is the row normalized adjacency matrix, T is the context window size and β is the number of negative samples used in NCE. Although directly factorizing the proximity matrix can result in a more accurate representation [5], for large graphs the low memory footprint of random walk based methods is well-founded [29].

Algorithm 3 DeepWalk [19]

1: Initialize X with random values
2: **for** i in $0..\gamma$ **do**
3: $\mathcal{O} = \text{Shuffle}(V)$
4: **for** $v \in \mathcal{O}$ **do**
5: $\mathcal{W}_{v_i} = \text{RandomWalk}(G, v_i, t)$
6: **for** $v_j \in \mathcal{W}_{v_i}$ **do**
7: **for** $u_k \in \mathcal{W}_{v_i}[j-T : j+T]$ **do**
8: $J(X) = -\log \sigma\left(x_{v_j}^T x_{u_k}\right)$
9: $X \leftarrow X - \alpha \nabla_X J$
10: **end for**
11: **end for**
12: **end for**
13: **end for**

where γ is te number of walks per node, t is the walk length and $\text{RandomWalk}(G, v, t)$ samples a random walk (sequence of nodes) in G starting from node v and with length t. The loss function can be extended with negative sampling in line 8 as $J(X) = -\log \sigma\left(x_{v_j}^T x_{u_k}\right) - \log \sigma\left(-x_{n_1}^T x_{n_2}\right)$ where n_1 and n_2 are negative samples.

Node2Vec

Node2Vec [9] uses a similar approach as DeepWalk, it differs only in the sampling strategy. Node2Vec samples second order (also referred as biased) random walks parameterized by p and q parameters. The unnormalized transition probabilities are defined as:

$$\alpha_{p,q}(t,x) = \begin{cases} \frac{1}{p}, & \text{if } d_{tx} = 0 \\ 1, & \text{if } d_{tx} = 1 \\ \frac{1}{q}, & \text{if } d_{tx} = 2 \end{cases} \tag{2.13}$$

where d_{tx} denotes the length of the shortest path between nodes t and x.

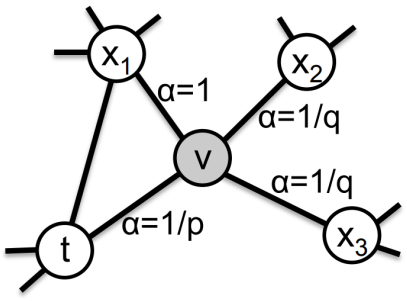

Figure 2.1: The walk just transitioned from t to v and is now evaluating its next step out of node v. Edge labels indicate search biases α.
Figure and caption taken from [9]

With the selection of p (also referred as return parameter) we can control the likelihood of immediately returning to a previous node in the walk. With high p values we can achieve exploration and avoid 2-hop redundancy, while lower values encourage more localized walks. The parameter q (also referred as in-out parameter) allows us to differentiate between "inward" and "outward" nodes.

The Node2Vec algorithm is mostly similar to Algorithm 3 with the only difference that it samples biased random walks instead of first order random walks at step 5 (RandomWalk).

2.2.3 Deep learning based approach

GraphSAGE

GraphSAGE [13] is an inductive node representation learning algorithm that uses graph neural networks to learn a mapping from nodes in the graph to the embedding space. To decouple computational complexity from node degree, GraphSAGE uses stochastic neighbor sampling. This enables GraphSAGE to be used on large graphs with high node degree. The GraphSAGE layer is defined as:

$$h_v^k \leftarrow \sigma\left(\frac{1}{Q} W \left[h_v^{k-1} \parallel m_{\mathcal{N}(v)}^k\right]\right) \tag{2.14}$$

$$m_{\mathcal{N}(v)}^k = \sum_{u \sim \mathcal{N}(v)} h_u^{k-1} \tag{2.15}$$

where $m_{\mathcal{N}(v)}^k$ is the message vector [15], $\parallel$ is the concatenation operator, σ is a nonlinear activation function, W is a learnable parameter of the layer and Q is the number of sampled neighbors. The initial representation h_v^0 can be any feature vector that describes node v.

The algorithm optimizes a similar objective function as DeepWalk in Equation 2.11 but instead of optimizing with regard to x we optimize for θ, the model parameters:

$$\max_{\theta} \frac{1}{|E|} \sum_{(p_1,p_2)\in E} \log \sigma\left(f_\theta^T(v_{p_1}) f_\theta(v_{p_2})\right) + \mathop{\mathbb{E}}_{(n_1,n_2)\sim E^c} \log \sigma\left(-f_\theta^T(v_{n_1}) f_\theta(v_{n_2})\right) \tag{2.16}$$

where E is the set of edges in G, E^c is a distribution over the complement of E in G. The expectation is approximated using sampling. The authors of [13] also suggest to normalize model outputs to unit length.

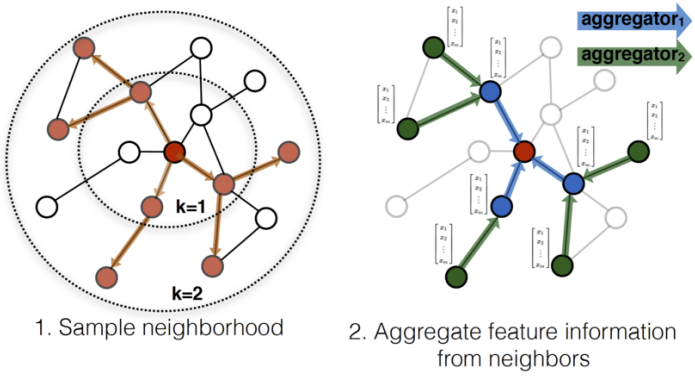

Figure 2.2: Illustration of the sampling and aggregation process used by GraphSAGE. Figure taken from [13]

Chapter 3

Own approach to real-time collaborative filtering

To build a scalable model for real-time collaborative filtering we use a two-stage recommendation framework [31]. The first stage is called candidate generation, in which a fast algorithm is used to construct a smaller set of items (called candidate pool). The size of the candidate pool is in the range of several hundreds. The second stage applies a more complex model to assign scores for each item given a user - model the probability that the user would interact with the given item - and select the most relevant ones from the pool. Quality of the candidate pool predetermines the performance of the recommendations; no matter how accurate the scoring model is, if relevant items are not included in the candidate pool, they can not make it to the final recommendation (only items from the candidate pool can be selected as the final recommendation). In this current work we aim to improve the candidate generation stage.

The input for our approach is a user-item interaction graph $G = (V, E)$, and X^{base} representations for nodes in G. In this work we experimented with representations mentioned in Section 2.2.

Our approach first constructs a representation r_u for user u and uses that representation to build the candidate pool - the top-K closest items are selected in terms of cosine similarity (item representations are drawn from X^{base}).

In this section we propose multiple ways how one can obtain user representations starting from simple, non parametric methods to using graph neural networks.

Lookup

The simplest way of constructing r_u is to just look it up from the node representation matrix: $r_u = X^{base}[u]$. The problem with this is that r_u does not contain any information regarding new edges in the graph, adaptation to changes in user behaviour is not possible.

Neighbor aggregation

To build user representation r_u we can average its neighbors. In this representation we can include all the observed edges (even the ones that were not present when X^{base} was constructed).

$$r_u = \frac{1}{|\mathcal{N}(u)|} \sum_{i \in \mathcal{N}(u)} x_i^{base} \tag{3.1}$$

Here $\mathcal{N}$ contains all the latest observed edges. Note that r_u does not depend on the original representation of the user x_u^{base}, which means that we can construct candidate pools for new users that were not present in the graph when X^{base} was learned.

Neighbor aggregation with importance scores

The importance method was inspired by the attention mechanism [24]. We use the original user representation x_u^{base} to compute importance scores for each item the user has interacted with.

$$r_u = \sum_{i \in \mathcal{N}(u)} \alpha(u,i) x_i^{base} \tag{3.2}$$

$$\alpha(u,i) = \frac{e^{x_u^T x_i}}{\sum_{j \in \mathcal{N}(u)} e^{x_u^T x_j}} \tag{3.3}$$

This way we can reduce the effect of interactions that are not typical for the user. For example let's say we have a user u that mostly consumes action movies. Then his original representation x_u will be similar to the movies he watches. Once he watches a romantic movie however, since the original representation x_u is closer to action movie representations, we will give more weight in the aggregation to the action movies - one highly unlikely, dissimilar neighbor can not alter the representation.

The drawback of this method that it only can be used on users that have representations in X^{base}.

Graph Neural Network

We can use a graph neural network to construct r_u based on representations of surrounding nodes in the graph:

$$r_u = f_\theta(G, X^{base})_u \tag{3.4}$$

where f_θ is a graph neural network with parameters θ and G is the user-item interaction graph. Using transductive embeddings as input in inductive representation learning algorithms can improve link prediction performance according to [11].

Before using this approach, we have to learn the network parameters first. We train the network by maximizing Equation 2.16 on graph G using X^{base} representations as input to the network. It is important that we must use the representations learned by the network instead of X^{base} for similarity search, since the representation space is different to the original representation space. We construct node representations $X^{ind} = f_\theta(G, X^{base})$ and use them in to find top-K similar items to r_u.

Continous representation update

Since the network is not computationally expensive to evaluate (especially if it is shallow) we can use it to update representations in X^{ind} to reflect new observations. This can be done efficiently in batch updates, first a given number of interactions E^{new} is collected and they are included in the graph: $G \leftarrow (V, E \cup E^{new})$. We refer this update strategy as continuous representation update. Our intuition is that we this strategy can improve performance, since not only the user representation will contain information of the latest edges but all the other nodes will have representations that encode new behavior.

Chapter 4

Empirical results

4.1 Details

In this section we present the architecture of the used SAGE model and document parameter values that we used to learn embeddings.

All embedding spaces were 64 dimensional. For the Node2Vec embeddings we used a context window size of 10, we set $p = q = 1$ and sampled 10 walks per node. For RandNE we used $\alpha_i = \frac{1}{7}$ and the order of the polynomial was 7.

The SAGE network consisted of 3 layers - that means information from 3-hop can be included in the constructed representations. All layers transform 64 dimensional inputs to 64 dimensional outputs and each layer contains a bias parameter (b) aside W, the weight matrix:

$$h_v^k \leftarrow \sigma\left(\frac{1}{Q} W \left[h_v^{k-1} \parallel m_{\mathcal{N}(v)}^k\right] + b\right) \tag{4.1}$$

The network consists of 33088 learnable parameters at total - which considered the scale of the problem is not large.

In this section we compare all the approaches from Section 2.2 to construct user representations r_u. We denote the lookup method ($r_u = X^{base}[u]$) with Lookup, the method described in Equation 3.1 is denoted with Neighbor, the one in Equation 3.2 with Importance and lastly, the one using a GraphSAGE network is denoted with SAGE. Additionally, we denote the SAGE variant that uses continuous representation updates as SAGE+.

4.2 Datasets

We evaluated our approach on two different e-commerce recommendation datasets:

Table 4.1: E-commerce datasets used for evaluation.

Dataset	Users	Items	Edges
H&M [1]	$1.37 \cdot 10^6$	$0.11 \cdot 10^6$	$3.18 \cdot 10^7$
RetailRocket [21]	$1.41 \cdot 10^6$	$0.47 \cdot 10^6$	$2.76 \cdot 10^6$

We filtered the graphs to exclude nodes with low degree, and due to limited hardware resources. Nodes with interactions less than 30 were excluded from the H&M graph and this limit was 5 for the RetailRocket data.

Next the edges in the graphs were split into 3 different sets A, B and C. Each edge in A proceeds all edge in B (in time) and each edge in B was observed before all edge in C as illustrated in Figure 4.1.

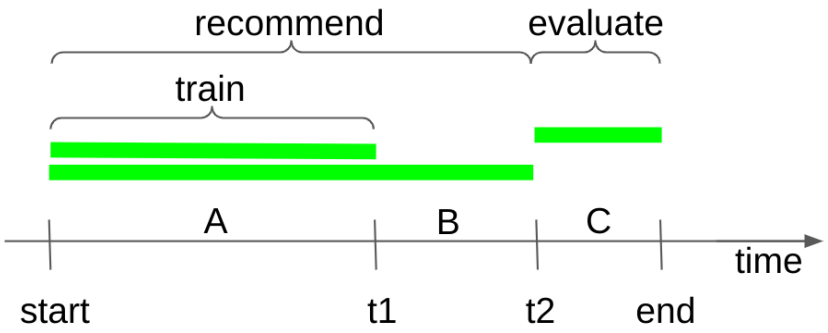

Figure 4.1: Illustration of the decomposition of edges in the graph for training, recommendation and evaluation.

The graph $G_A = (V, A)$ was used for learning transductive representations. The graph $G_B = (V, A \cup B)$ was used for generating user representations and constructing candidate pools (in the neighborhood aggregation and other approaches that rely on neighbor sampling). Finally, a graph $G_C = (V, C)$ was used to evaluate the performance of each method.

Table 4.2: Sub-graphs of original datasets used in evaluation.

Dataset	Graph	Users	Items	Edges
H&M	G_A	306490	55631	$1.99 \cdot 10^7$
	G_B	306490	55631	$2.17 \cdot 10^6$
	G_C	163083	28760	$1.11 \cdot 10^6$
RetailRocket	G_A	60648	35601	703367
	G_B	60648	35601	742443
	G_C	9658	16083	39076

4.3 Quality of candidate pool

As already mentioned in Section 1.1 when constructing a candidate pool Recall needs to be maximized (the ratio of retrieved relevant elements to all relevant elements). We measured Recall on both datasets using different node representation learning methods from Section 2.2 to construct X^{base}. We also perform paired Student's t-test [23] to support our statements.

Regardless the method, Node2Vec seems to be the best representation for candidate pool generation out of the three considered approach. We performed t-test with $p \leq 0.01$ on the RetailRocket dataset, Node2Vec significantly outperformed the other two representations learned with other embedding algorithms in all scenario. The reason for this is that Node2Vec considers high order node proximities while GF only captures first order

Table 4.3: Recall@500 values on the RetailRocket dataset with different node representations. The best results per representation are highlighted in boldface. Superscripts denote significant differences in paired Student's t-test with $p \leq 0.05$.

#	Model	RandNE	GF	Node2Vec
a	Importance	0.326^{bc}	0.324^{bc}	$\mathbf{0.415}^{de}$
b	Lookup	0.256	0.287	0.410^{de}
c	Neighbor	0.318^{b}	0.296	0.415^{de}
d	SAGE	0.331^{bc}	0.315^{bc}	0.382
e	SAGE+	$\mathbf{0.345}^{abcd}$	$\mathbf{0.335}^{bcd}$	0.391^{d}

Table 4.4: Recall@50 values on the H&M dataset with different node representations. The best results per representation are highlighted in boldface. Superscripts denote significant differences in paired Student's t-test with $p \leq 0.05$.

#	Model	RandNE	GF	Node2Vec
a	Importance	0.037^{bde}	0.018	$\mathbf{0.100}^{bde}$
b	Lookup	0.001	0.017	0.011
c	Neighbor	$\mathbf{0.049}^{abde}$	0.020^{abd}	0.099^{bde}
d	SAGE	0.013^{be}	0.018	0.020^{b}
e	SAGE+	0.012^{b}	$\mathbf{0.021}^{abcd}$	0.023^{bd}

connections. RandNE also works with higher order polynomial of the Laplacian but it sacrifices the optimality of the solution for scalability by using random projections [6].

Using continuous representation update (SAGE+) results in significantly better performance than using the initial embeddings X^{ind} created with the inductive model (SAGE). This supports our intuition in Section 3. For "weaker" embeddings (GF and RandNE) on the smaller dataset the SAGE+ model (with updated embeddings) significantly outperforms other non-parametric methods in terms of Recall. Even though SAGE (and SAGE+) perform better with Node2Vec embeddings than with any other embedding, the performance of the parametric method is significantly worse than other non-parametric methods. One explanation for this phenomenon can be the strong regularization we employ in the network. The SAGE layer can not distinguish between representations of different neighbors (Equation 2.14). It uses summation as the aggregation function, all neighbors are similarly important - this way we lose the ability to distinguish between more and less important neighbors. One solution to this problem could be to use a layer that employs the attention mechanism to generate importance weights for neighbors, for example the Graph Attention Network layer [25] or UniMP [22], a Transformer layer for GNNs. This is a promising future direction in our research.

For RandNE and GF embeddings using importance scores (Importance method) improved the performance compared to the simple averaging approach (Neighbor) on the smaller dataset while on the bigger dataset the completely different behavior was observed. On the Node2Vec embeddings importance and Neighbor methods performed similarly, there seems to be no advantage of using one over the other.

Average cosine similarity on the RetailRocket dataset

We measured the average cosine similarity per user to neighboring nodes (relevant items) and to randomly selected nodes (not relevant items) and plotted the distribution of these average similarity values. Average cosine similarity to its neighbors per user for user u:

$$\frac{1}{|\mathcal{N}(u)|} \sum_{i \in \mathcal{N}(u)} \cos(r_u, x_i) \tag{4.2}$$

where $\cos(a, b) = \frac{a^T b}{|a||b|}$ is the cosine similarity. Average cosine similarity per user to random items:

$$\mathop{\mathbb{E}}_{i \sim V_{item} \setminus \mathcal{N}(u)} \cos(r_u, x_i) \tag{4.3}$$

where $V_{item} \setminus \mathcal{N}(u)$ is a distribution over item type nodes in the graph with the neighbors of u excluded. We approximate the expectation with sampling.

On the RetailRocket dataset we have experienced that the cosine similarity of r_u to representations of $\mathcal{N}(u)$ are significantly higher than the similarity to randomly selected nodes for some users. The distribution of similarities can be seen on Figure 4.2 for the Importance method on the RetailRocket dataset with Node2Vec embeddings. Other non-parametric methods also produced quite similar distributions.

We have measured even higher seperation using the SAGE embeddings, illustrated on Figure 4.3.

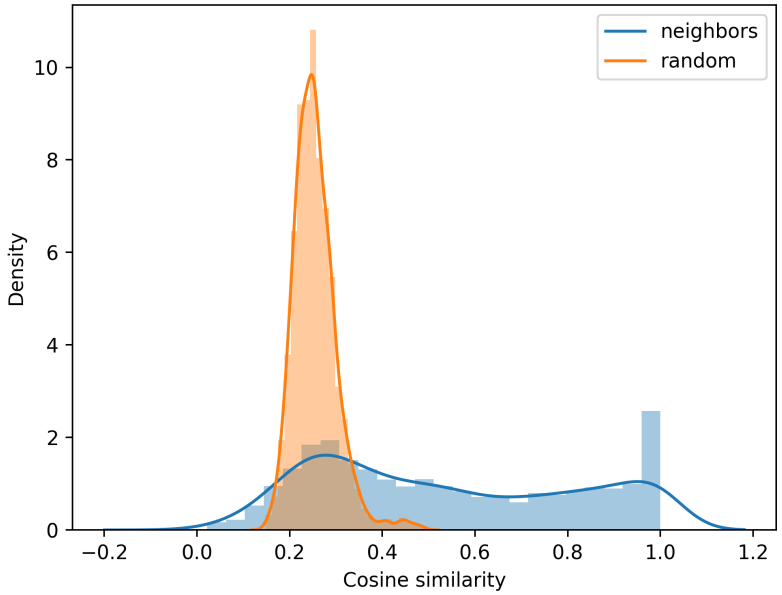

Figure 4.2: Average cosine similarity of r_u to representations of neighboring nodes (blue) and randomly selected nodes (orange) on the RetailRocket dataset using Importance method with Node2Vec embeddings.

On both Figure 4.2 and Figure 4.3 we can observe that the overlap of neighboring and randomly selected nodes is quite small. This means users that lie in the higher average neighbor cosine similarity region are more likely to have their actual neighbors selected

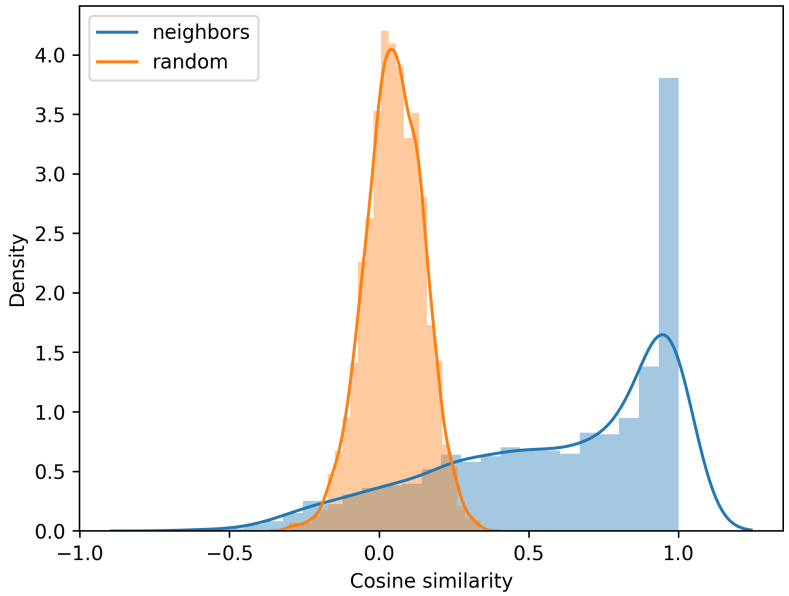

Figure 4.3: Average cosine similarity of r_u to representations of neighboring nodes (blue) and randomly selected nodes (orange) on the RetailRocket dataset using SAGE method with Node2Vec embeddings.

in the candidate pool - similarity to neighboring nodes is significantly higher than to randomly selected nodes, the top-k selected items will be neighbors with high probability.

Explaining peaks on Figure 4.2 and 4.3

Peaks at 1.0 on Figure 4.2 and Figure 4.3 are users that have either only one or very few interaction in G_C and some of those interactions are also present in G_B - if user u has one neighbor n in both G_C and G_B then its representation r_u will be exactly the representation of that specific neighbor resulting in 1.0 cosine similarity on G_C (using Equation 3.2):

$$r_u = \sum_{i \in \mathcal{N}(u)} \alpha(u,i) x_i^{base} \tag{4.4}$$

Since u has one neighbor n:

$$\mathcal{N}(u) = \{n\} \tag{4.5}$$

Softmax for one item is 1:

$$\alpha(u,n) = \frac{e^{x_u^T x_n}}{e^{x_u^T x_n}} = 1 \tag{4.6}$$

$$r_u = 1 x_n^{base} = x_n^{base} \tag{4.7}$$

The cosine similarity of a vector to itself is 1:

$$\cos(r_u, x_n^{base}) = \cos(x_n^{base}, x_n^{base}) = 1 \tag{4.8}$$

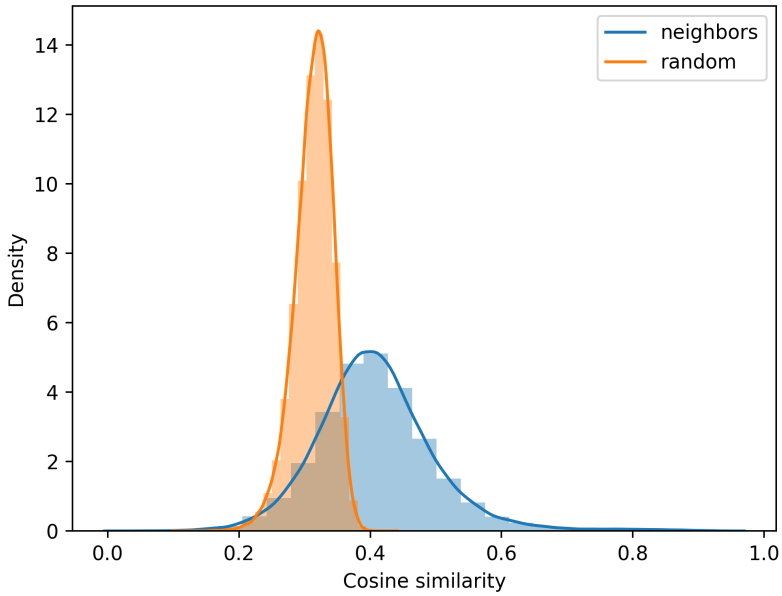

Figure 4.4: Average cosine similarity of r_u to representations of neighboring nodes (blue) and randomly selected nodes (orange) on the H&M dataset using Importance method with Node2Vec embeddings.

Average cosine similarity on the H&M dataset

On Figure 4.4 the separation observed on the smaller dataset is not present, the overlap between similarity to random items and neighbors is much higher. Our intuition is that the dimensionality of the embeddings (64) is not enough for the larger problem, 3 million nodes can not be mapped in 64 dimensions such that all similar nodes are close in this space (in terms of cosine similarity) and dissimilar ones are placed farther apart. Finding the appropriate value for embedding dimensions is an open question, we have to balance between information loss and efficiency [10].

Chapter 5

Conclusion

In our work we have proposed parametric and non-parametric approaches for constructing user representations efficiently to generate candidate pools in real-time collaborative filtering. We have compared these methods on two datasets, where we measured Recall to evaluate the quality of the candidate pool and cosine similarity on neighboring vs randomly selected items to verify if the learned representations are good for distinguishing relevant and not relevant items.

We have shown that incorporating new interactions in the representations is necessary to improve recommendation quality and adapt to changes in user behavior.

Our proposed graph neural network based approach can achieve superior performance on lower quality embeddings than other, non-parametric methods however, on high quality representations non-parametric methods seem to perform better. We used statistical tests to support our statements.

We recognized the weakness of the GraphSAGE graph neural network layer, that it does not differentiate important and not important neighbors. We plan to do research on layers that assign importance scores to neighbors in the aggregation phase like UniMP and Graph Attention Networks.

We have came to a conclusion that the dimensionality of representations was not sufficient on the larger dataset. In the future we will continue research using higher dimensional embeddings.

Bibliography

[1] H& m personalized fashion recommendations. https://www.kaggle.com/competitions/h-and-m-personalized-fashion-recommendations, 2022.

[2] Amr Ahmed, Nino Shervashidze, Shravan Narayanamurthy, Vanja Josifovski, and Alexander J Smola. Distributed large-scale natural graph factorization. In *Proceedings of the 22nd international conference on World Wide Web*, pages 37–48, 2013.

[3] Robert M Bell and Yehuda Koren. Lessons from the netflix prize challenge. *Acm Sigkdd Explorations Newsletter*, 9(2):75–79, 2007.

[4] Hongyun Cai, Vincent W Zheng, and Kevin Chen-Chuan Chang. A comprehensive survey of graph embedding: Problems, techniques, and applications. *IEEE Transactions on Knowledge and Data Engineering*, 30(9):1616–1637, 2018.

[5] Shaosheng Cao, Wei Lu, and Qiongkai Xu. Grarep: Learning graph representations with global structural information. In *Proceedings of the 24th ACM international on conference on information and knowledge management*, pages 891–900, 2015.

[6] Haochen Chen, Syed Fahad Sultan, Yingtao Tian, Muhao Chen, and Steven Skiena. Fast and accurate network embeddings via very sparse random projection. In *Proceedings of the 28th ACM international conference on information and knowledge management*, pages 399–408, 2019.

[7] Siheng Chen, Sufeng Niu, Leman Akoglu, Jelena Kovačević, and Christos Faloutsos. Fast, warped graph embedding: Unifying framework and one-click algorithm. *arXiv preprint arXiv:1702.05764*, 2017.

[8] Pierre Comon, Xavier Luciani, and André LF De Almeida. Tensor decompositions, alternating least squares and other tales. *Journal of Chemometrics: A Journal of the Chemometrics Society*, 23(7-8):393–405, 2009.

[9] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864, 2016.

[10] Weiwei Gu, Aditya Tandon, Yong-Yeol Ahn, and Filippo Radicchi. Principled approach to the selection of the embedding dimension of networks. *Nature Communications*, 12(1):1–10, 2021.

[11] Chitrank Gupta, Yash Jain, Abir De, and Soumen Chakrabarti. Integrating transductive and inductive embeddings improves link prediction accuracy. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pages 3043–3047, 2021.

[12] Michael Gutmann and Aapo Hyvärinen. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 297–304. JMLR Workshop and Conference Proceedings, 2010.

[13] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.

[14] William L Hamilton. Graph representation learning. *Synthesis Lectures on Artifical Intelligence and Machine Learning*, 14(3):1–159, 2020.

[15] William L Hamilton, Rex Ying, and Jure Leskovec. Representation learning on graphs: Methods and applications. *arXiv preprint arXiv:1709.05584*, 2017.

[16] Thomas Hofmann. Latent semantic models for collaborative filtering. *ACM Transactions on Information Systems (TOIS)*, 22(1):89–115, 2004.

[17] Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, 7(3):535–547, 2019.

[18] Xin Luo, Mengchu Zhou, Yunni Xia, and Qingsheng Zhu. An efficient non-negative matrix-factorization-based approach to collaborative filtering for recommender systems. *IEEE Transactions on Industrial Informatics*, 10(2):1273–1284, 2014.

[19] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710, 2014.

[20] Jiezhong Qiu, Yuxiao Dong, Hao Ma, Jian Li, Kuansan Wang, and Jie Tang. Network embedding as matrix factorization: Unifying deepwalk, line, pte, and node2vec. In *Proceedings of the eleventh ACM international conference on web search and data mining*, pages 459–467, 2018.

[21] Retailrocket. Retailrocket recommender system dataset, Mar 2017. URL `https://www.kaggle.com/datasets/retailrocket/ecommerce-dataset`.

[22] Yunsheng Shi, Zhengjie Huang, Shikun Feng, Hui Zhong, Wenjin Wang, and Yu Sun. Masked label prediction: Unified message passing model for semi-supervised classification. *arXiv preprint arXiv:2009.03509*, 2020.

[23] Mark D Smucker, James Allan, and Ben Carterette. A comparison of statistical significance tests for information retrieval evaluation. In *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*, pages 623–632, 2007.

[24] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[25] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *stat*, 1050:20, 2017.

[26] Santosh S Vempala. *The random projection method*, volume 65. American Mathematical Soc., 2005.

[27] Cheng Yang, Maosong Sun, Zhiyuan Liu, and Cunchao Tu. Fast network embedding enhancement via high order proximity approximation. In *IJCAI*, volume 17, pages 3894–3900, 2017.

[28] R Zadeh, H Li, B He, M Lublin, and Y Perez. Cme 323: Distributed algorithms and optimization, spring 2015. *University Lecture*, 2015.

[29] Daokun Zhang, Jie Yin, Xingquan Zhu, and Chengqi Zhang. Network representation learning: A survey. *IEEE transactions on Big Data*, 6(1):3–28, 2018.

[30] Ziwei Zhang, Peng Cui, Haoyang Li, Xiao Wang, and Wenwu Zhu. Billion-scale network embedding with iterative random projection. In *2018 IEEE International Conference on Data Mining (ICDM)*, pages 787–796. IEEE, 2018.

[31] Yu Zheng, Chen Gao, Liang Chen, Depeng Jin, and Yong Li. Dgcn: Diversified recommendation with graph convolutional networks. In *Proceedings of the Web Conference 2021*, pages 401–412, 2021.