SMORe:
Modularize Graph Embedding for Recommendation

Chih-Ming Chen (CM)
Ting-Hsiang Wang (Sean)
Chuan-Ju Wang (Jennifer)
Ming-Feng Tsai (Victor)
Tutorial Agenda

- Lecture (Sean & CM, 65 minutes)
- Hands-on (CM, 15 minutes)
- Q&A (Sean & CM, 10 minutes)

QR to Slides, Codes, Abstract
DeepWalk: Online Learning of Social Representations

Ting-Hsiang Wang, Texas A&M University
CLIP Lab, National Chengchi University

ABSTRACT
We present latent representations of vertices in a network. These latent representations encode social relations in a continuous vector space, which is easily exploited by statistical models.

Department of Computer Science and unsupervised feature learning (or unsupervised deep learning) techniques, which have proven successful in natural language processing, into network analysis for a global view of the network, especially in the presence of missing information.

In this paper we introduce a novel approach for learning latent representations of a graph's vertices, by modeling a stream of short random walks. Social representations are latent features of the vertices that capture neighborhood similarity and community membership. These latent representations encode social relations in a continuous vector space with a relatively small number of dimensions.

DeepWalk learns a latent representation of social interactions in a network by treating the embedding as an output. The result of applying our method to the well-studied Karate network is shown in Figure 1. The graph, as typically presented by force-directed layouts, is shown in Figure 1a. Figure 1b shows the output of our method with 2 latent dimensions. Beyond the striking correspondence to clusters found through modularity maximization in the input graph (1a) (shown as vertex colors), DeepWalk's representations on several multi-label tasks for social networks such as BlogCatalog, Flickr, and YouTube outperform competing methods up to 10% higher than competing methods when labeled data is sparse.

The sparsity of a network representation is both a strength and a weakness. Sparsity enables the design of efficient algorithms, but can make it harder to generalize in statistical learning. Machine learning applications in networks (such as network classification [15, 37], content recommendation [11], anomaly detection [5], and missing link prediction [22]) must be able to deal with this sparsity in order to survive.

DeepWalk is also scalable. It is an online learning algorithm which builds useful incremental results, and is trivially parallelizable. These qualities make it suitable for a broad class of real world applications such as network classification and anomaly detection.

Categories and Subject Descriptors
H.2.8 [Database Management]: Model - Statistical
I.2.6 [Artificial Intelligence]: Database Applications
I.5.1 [Pattern Recognition]: Model - Clustering

1. INTRODUCTION
The sparsity of a network representation is both a strength and a weakness. Sparsity enables the design of efficient algorithms, but can make it harder to generalize in statistical learning. Machine learning applications in networks must be able to deal with this sparsity in order to survive.

DeepWalk learns a latent representation of social interactions in a network by modeling a stream of short random walks. Social representations are latent features of the vertices that capture neighborhood similarity and community membership. These latent representations encode social relations in a continuous vector space with a relatively small number of dimensions.

DeepWalk takes a graph as input and produces a latent representation encodes community structure so it can be easily exploited by statistical models. The result of applying our method to the well-studied Karate network is shown in Figure 1. The graph, as typically presented by force-directed layouts, is shown in Figure 1a. Figure 1b shows the output of our method with 2 latent dimensions. Beyond the striking correspondence to clusters found through modularity maximization in the input graph (1a) (shown as vertex colors), DeepWalk's representations on several multi-label tasks for social networks such as BlogCatalog, Flickr, and YouTube outperform competing methods up to 10% higher than competing methods when labeled data is sparse.

The sparsity of a network representation is both a strength and a weakness. Sparsity enables the design of efficient algorithms, but can make it harder to generalize in statistical learning. Machine learning applications in networks must be able to deal with this sparsity in order to survive.

DeepWalk is also scalable. It is an online learning algorithm which builds useful incremental results, and is trivially parallelizable. These qualities make it suitable for a broad class of real world applications such as network classification and anomaly detection.
Lecture Agenda

(Sean) **Q0. Recommendation** (REC) and challenges

(Sean) **Q1. Why graph embedding** (GE) for REC

(Sean) **Q2. SMORe** modularization of GE and benefits

(CM) **Q3. Exemplar structural modeling** for REC

(CM) **Q4. REC using SMORe**
Lecture Agenda

(Sean) **Q0. Recommendation** (REC) and challenges

(Sean) **Q1.** Why graph embedding (GE) for REC

(Sean) **Q2. SMORE** modularization of GE and benefits

(CM) **Q3.** Exemplar structural modeling for REC

(CM) **Q4.** REC using SMORE
REC systems are everywhere!

- Amazon: Book, Toy, Electronics, etc.
- Netflix: Movie
- Spotify: Music
- Facebook: Friend, Interest Group
- LinkedIn: Recruiter, Job Seeker
- Twitter: Trend, Following
- Google: News, Ad, Media, etc.
- Yelp: Local Business

And many more ...
Let’s start with Collaborative Filtering (CF)

- REC center around users to provide customized results
- CF assumes *people agree on things are likely to agree on other things*

> Sam and Derek have similar tastes; therefore, if Sam likes Song C, it is likely Derek does, too
Let’s start with Collaborative Filtering (CF)

- REC center around users to provide customized results
- CF assumes **people agree on things are likely to agree on other things**

→ Sam and Derek have similar tastes; therefore, if Sam likes Song C, it is likely Derek does, too
Let's start with Collaborative Filtering (CF)

- REC center around users to provide customized results
- CF assumes **people agree on things are likely to agree on other things**

<table>
<thead>
<tr>
<th>User</th>
<th>Song A</th>
<th>Song B</th>
<th>Song C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sam</td>
<td>5</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Derek</td>
<td>4</td>
<td>5</td>
<td>?</td>
</tr>
<tr>
<td>Liz</td>
<td>-</td>
<td>3</td>
<td>?</td>
</tr>
<tr>
<td>Roger</td>
<td>-</td>
<td>-</td>
<td>?</td>
</tr>
</tbody>
</table>

→ Sam and Derek have similar tastes; therefore, if Sam likes Song C, it is likely Derek does, too.
Let’s start with Collaborative Filtering (CF)

- REC center around users to provide customized results
- CF assumes **people agree on things are likely to agree on other things**

Sam and Derek have similar tastes; therefore, if Sam likes Song C, it is likely Derek does, too.
Challenge (1) : Data Sparsity

- When data sparsity occurs, it is difficult to make accurate REC with CF.
Challenge (1): Data Sparsity

When data sparsity occurs, it is difficult to make accurate REC with CF.

Data sparsity occurs to Liz as there is insufficient ratings from Liz to compare her taste with others’.
Challenge (1): Data Sparsity

- When data sparsity occurs, it is difficult to make accurate REC with CF.

Data sparsity occurs to Liz as there is insufficient ratings from Liz to compare her taste with others’
Challenge (2) : Cold Start

- When **cold start** occurs, it is impossible to conduct CF since there is **no context** for comparison.

> **Cold Start** occurs to Roger as there is **NO** song rating from Roger to compare his taste with others’.
Challenge (3) : Constant Cold Start

• Some Cold Start situations are constant, i.e., never warm up
• E.g., event tickets are sold before user-event interactions can occur
Common Solution: Add Attributes

→ REC systems often mitigate **Complete Cold Start** problems by requiring new users to specify **interests** and items be labeled using **tags**

→ By requiring user interests and item tags to come from **predefined label set**, users and items share context for comparison
To enrich context for comparison ...

- For additional context for comparison, REC systems often borrow auxiliary information, such as item content and user and item attributes
- This forms a Heterogeneous Information Network (HIN)
Add auxiliary information ...

- For additional context for comparison, REC systems often borrow auxiliary information, such as item content and user and item attributes
- This forms a Heterogeneous Information Network (HIN)
And more auxiliary information!

For additional context for comparison, REC systems often borrow auxiliary information, such as item **content** and user and item **attributes**.

- **Item attributes** are added to improve item comparison, e.g., music genre, artist, metadata, etc.

- **User attributes** are added to improve user comparison, e.g., social network, occupation, favorite artist & genre, etc.

This forms a **Heterogeneous Information Network (HIN)**
Graph coordinates all relations!

→ Item attributes are added to improve item comparison, e.g., music genre, artist, metadata, etc.

→ User attributes are added to improve user comparison, e.g., social network, occupation, favorite artist & genre, etc.

→ HIN gives holistic view of complex systems

- For additional context for comparison, REC systems often borrow auxiliary information, such as item content and user and item attributes
- This forms a Heterogeneous Information Network (HIN)
**Lecture Agenda**

(Sean) **Q0. Recommendation** (REC) and challenges

(Sean) **Q1. Why graph embedding** (GE) for REC

(Sean) **Q2. SMORE** modularization of GE and benefits

(CM) **Q3. Exemplar structural modeling** for REC

(CM) **Q4. REC using SMORE**
Why Graph?

1. **Universal language** for describing complex data [1]
   - Many fields have all chosen graph to depict entity interactions

   ![Core Sound Food Web](image1)
   ![Delta Airline Route Map Network](image2)
   ![Twitter Ego-Network](image3)
Why Graph?

1. **Universal language** for describing complex data [1]
   - Many fields have all chosen graph to depict entity interactions

2. **Shared vocabulary** (therefore ideas) between fields [1]
   - E.g., *distributional hypothesis* in linguistics vs. CF in REC systems
Why Graph?

1. **Universal language** for describing complex data [1]
   - Many fields have all chosen graph to depict entity interactions

2. **Shared vocabulary** (therefore ideas) between fields [1]
   - E.g., *distributional hypothesis* in linguistics vs. *CF* in REC systems

3. **Holistic view** of complex systems of interactions
   - Different domains can *easily connect* and be *jointly mined*

→ *HINs* easily integrates information from different domains
Why Graph?

1. **Universal language** for describing complex data [1]
   - Many fields have all chosen graph to depict entity interactions

2. **Shared vocabulary** (therefore ideas) between fields [1]
   - E.g., distributional hypothesis in linguistics vs. CF in REC systems

3. **Holistic view** of complex systems of interactions
   - Different domains can easily connect and be jointly mined

4. **General definition** of contexts used for entity comparison
   - Model **graph structure** instead specific types of relations

→ Different REC approaches, similar structure
Why Graph?

1. **Universal language** for describing complex data [1]
   - Many fields have all chosen graph to depict entity interactions
2. **Shared vocabulary** (therefore ideas) between fields [1]
   - E.g., *distributional hypothesis* in linguistics vs. CF in REC systems
3. **Holistic view** of complex systems of interactions
   - Different domains can easily connect and be jointly mined
4. **General definition** of contexts used for entity comparison
   - Model graph structure instead specific types of relations
5. **Sophisticated models** available from network-related researches
   - Network schema, meta-path, subgraph matching, information propagation, etc.
Why Graph?

1. **Universal language** for describing complex data [1]

2. **Shared vocabulary** (therefore **ideas**) between fields [1]

3. **Holistic view** of complex systems of interactions

4. **General definition** of contexts used for entity comparison

5. **Sophisticated models** available from network-related researches
REC as Link Prediction on Graphs

- Many ways to compare node similarity: entity types, shared neighbors, distance, etc.
- Early REC models, e.g., CF and CBF, define intuitive relations for similarity measurement
- As models mature and data diversity skyrockets nowadays, designing features for REC becomes increasingly challenging
DeepWalk: Online Learning of Social Representations

We present DeepWalk, a novel approach for learning latent representations of vertices in a network. These latent representations encode social relations in a continuous vector space with a relatively small number of dimensions. Note the correspondence between a graph's vertices, by modulating a stream of short random walks. Social representations generalize neural language models to process a sequence of words to graphs.

In this paper we introduce DeepWalk, which builds useful incremental results, and is trivially parallelizable. We demonstrate DeepWalk’s potential in real world scenarios such as network classification [15, 37], content recommendations, e.g., distance and angle similarities, and even logical analogies [28].

• GE for REC exploits observed links as graph structures to predict unobserved links
• Entities are converted into spatial features (embeddings) in the hidden layer and iteratively updated such that their interactions approximate values in the output layer
• Entity relatedness is preserved as spatial properties, e.g., distance and angle
Why Embedding?

1. **Efficient retrieval from approximate nearest neighbor (ANN) search methods**
   - E.g., Spotify ANNOY reduces *curse of dimensionality* during online search by maintaining a binary tree of subdivisions such that **good enough results** can be found in $O(\log n)$ [4]
Why Embedding?

1. **Efficient retrieval** from approximate nearest neighbor (ANN) search methods

   - E.g., Spotify ANNOY reduces *curse of dimensionality* during online search by maintaining a binary tree of subdivisions such that *good enough results* can be found in \( O(\log n) \) [4]

   ![Diagram showing efficient retrieval from ANN search](image)

   - *Brute force exhaustive search* finds the closest nearest neighbors in \( O(\log n!) \)
Why Embedding?

1. **Efficient retrieval** from approximate nearest neighbor (ANN) search methods

   - E.g., Spotify ANNOY reduces curse of dimensionality during online search by maintaining a binary tree of subdivisions such that **good enough results** can be found in $O(\log n)$ [4].

   → **Brute force exhaustive search** finds the closest nearest neighbors in $O(\log n!)$.
1. **Efficient retrieval** from approximate nearest neighbor (ANN) search methods

   - E.g., Spotify ANNOY reduces **curse of dimensionality** during online search by maintaining a binary tree of subdivisions such that **good enough results** can be found in $O(\log n)$ [4]

   → Spotify ANNOY builds binary tree of subdivisions to quickly find closest neighbors

---

**Why Embedding?**

---
Why Embedding?

1. **Efficient retrieval from approximate nearest neighbor (ANN) search methods**
   - E.g., Spotify ANNOY reduces curse of dimensionality during online search by maintaining a binary tree of subdivisions such that **good enough results** can be found in $O(\log n)$ [4]

→ Spotify ANNOY builds binary tree of subdivisions to quickly find closest neighbors
Why Embedding?

1. Efficient retrieval from approximate nearest neighbor (ANN) search methods
   - E.g., Spotify ANNOY reduces curse of dimensionality during online search by maintaining a binary tree of subdivisions such that good enough results can be found in $O(\log n)$ [4]

2. Efficient pairwise comparison due to dimensionality reduction (DR)
   - Lower dimension costs less to calculate feature similarity
Why Embedding?

1. **Efficient retrieval from approximate nearest neighbor (ANN) search methods**
   - E.g., Spotify ANNOY reduces curse of dimensionality during online search by maintaining a binary tree of subdivisions such that *good enough results* can be found in $O(\log n)$ [4]

2. **Efficient pairwise comparison due to dimensionality reduction (DR)**
   - Lower dimension costs less to calculate feature similarity

3. **Reduced space complexity due to DR**
   - Lower dimension costs less storage space
Why Embedding?

1. **Efficient retrieval** from approximate nearest neighbor (ANN) search methods
   - E.g., Spotify ANNOY reduces curse of dimensionality during online search by maintaining a binary tree of subdivisions such that good enough results can be found in $O(\log n)$ [4]

2. **Efficient pairwise comparison** due to dimensionality reduction (DR)
   - Lower dimension costs less to calculate feature similarity

3. **Reduced space complexity** due to DR
   - Lower dimension costs less storage space

4. **Transfer learning** with pertained embeddings
   - Pretrained embeddings are better guesses than randomly initialized vectors
GE for REC : Challenges

- **So graph embedding is GREAT for recommendation:**
  - Reduces data sparsity and cold start via integrating auxiliary information
  - Provides holistic view of REC problem and jointly mines different relations in terms of graph structures
  - Trains fast, compares fast, and retrieves fast while taking less space
So **graph embedding** is GREAT for **recommendation**:

- Reduces data sparsity and cold start via integrating auxiliary information
- Provides **holistic view of REC problem** and jointly mines different relations in terms of **graph structures**
- Trains fast, compares fast, and retrieves fast while taking less space

**What’s the catch?**
Lecture Agenda

(Sean) Q0. Recommendation (REC) and challenges

(Sean) Q1. Why graph embedding (GE) for REC

(Sean) Q2. SMORe modularization of GE and benefits

(CM) Q3. Exemplar structural modeling for REC

(CM) Q4. REC using SMORe
Let’s look at GE process …
Let’s look at GE process …

Graph structures

Embedding space
Let’s look at GE process …
Let’s look at GE process …
Let’s look at GE process …

Graph structures

Samples relation

Maps entities to space

Embedding space

Optimizes distance
Challenge (1): Select structure is HARD

Maps entities to space

Embedding space

Optimizes distance

→ Unsure which graph structures best model current REC task
Challenge (1) : Select structure is HARD

Graph structures

Samples relation

Maps entities to space

Embedding space

Optimizes distance

→ Unsure which graph structures best model current REC task

In bipartite, model 9 & 1 by neighborhood is intuitive; but in planar, 5 & 7 are also shown similar in betweenness.
Challenge (1) : Select structure is HARD

Graph structures

Embedding space

Maps entities to space

Samples relation

→ Unsure which graph structures best model current REC task

In bipartite, model 9 & 1 by neighborhood is intuitive; but in planar, 5 & 7 are also shown similar in betweenness.
Challenge (2) : Customize GE model

→ Unsure which graph structures best model current REC task

→ Wanna customize GE methods during each stage of training
Challenge (3) : Make fair comparison

→ Unsure **which graph structures** best model current REC task

→ Wanna **customize GE methods** during each stage of training

→ **Difficult to compare models** as their implementations often vary
Solution: Modularize GE for adaptability!
Solution: Modularize GE for adaptability!
Solution: Modularize GE for adaptability!

- Extracts **graph structures** from dataset while remains **type-agnostic** to sampled entities, i.e., nodes & edges

  \[ G = (V, E, R, \phi) \]

  \[ E : \{(v_1, v_2) | (v_1, v_2) \in V \times V \} \]

  \[ \phi : E \rightarrow R \]
Solution : Modularize GE for adaptability!

- Converts entities into spatial features via embedding stacking operations, e.g., lookup, pooling (average, etc.)

\[ f: (p) \rightarrow \mathbb{R}^d \]

\[ f: (p, q) \rightarrow \mathbb{R}^d \]
Solution: Modularize GE for adaptability!

- Preserves entity relatedness as spatial properties with customizable similarity metrics and loss functions

→ Euclidean distance keeps triangular inequality; dot product does not [5]
Solution: Modularize GE for adaptability!

Graph structures

Samples relation

Maps entities to space

Embedding space

Optimizes distance

Sampler

Mapper

Optimizer for Recommendation
SMORe: Modular GE toolkit for REC

- Sampler
- Mapper
- Optimizer for Recommendation

Graph structures
Samples relation
Maps entities to space
Embedding space
Optimizes distance
SMORe : Modular GE toolkit for REC

SMORe is a modular GE toolkit for REC

S’more is a campfire treat with layers
Benefits of SMORe

SMORe

Sampler
Mapper
Optimizer

Model A
Model B
Model C

Performance level

Modules contribute different levels of performance during different REC tasks

for Recommendation
Benefits of **SMORe**

**SMORe** enables combining the most suitable modules for given REC task.
Benefits of SMORe

SMORe enables combining the most suitable modules for given REC task

SMORe
Sampler
Mapper
Optimizer
for Recommendation
Benefits of SMORe

As toolkit for research:
1. Baseline comparison
2. Ballpark approaches
3. One module at a time

As framework for development:
1. Reduces development time
2. Raises performance limit
3. Adapts to new tasks
References


Lecture Agenda

(Sean)  Q0. Recommendation (REC) and challenges

(Sean)  Q1. Why graph embedding (GE) for REC

(Sean)  Q2. SMORE modularization of GE and benefits

(CM)   Q3. Exemplar structural modeling for REC

(CM)   Q4. REC using SMORE
Example Graph

vertex

edge
Vertex Structure

- **Adjacency**
  - vertices which share the same edge
- **Neighborhood**
  - vertices which share similar connections
- **Community**
  - vertices which share similar communities
- **Centrality**
  - vertices which share similar properties
1. Adjacency

![Graph Diagram]

- **A**
  - Connects to **B**, **D**, and **E**
- **B**
  - Connects to **A**, **D**, and **F**
- **C**
  - Connects to **A**, **D**, and **E**
- **D**
  - Connects to **A**, **C**, **E**, and **F**
- **E**
  - Connects to **A**, **C**, **D**, and **F**
- **F**
  - Connects to **B**, **D**, **C**, and **E**
Adjacency

vertices share the same edge are treated as similar pairs

user-item preference
Matrix Factorization


\[
\begin{array}{cccc}
\text{U1} & \text{U2} & \text{U3} & \text{U4} \\
\hline
\text{I1} & \text{I2} & \text{I3} & \text{I4} \\
\end{array}
\]

\[
\begin{array}{cccc}
q_1 & q_2 & q_3 \\
\end{array}
\]

\[
\text{min}_{q^*, p^*} \sum_{(u, i) \in K} (r_{ui} - q_i^T p_u)^2
\]

(rating - item*user)
Matrix Factorization in **SMORe**

Graph

user-item graph
Matrix Factorization in SMORe

Graph

Sampler

(user, item)
Matrix Factorization in SMORe

Graph

Sampler

(user, item)

Mapper

Embedding Lookup
Matrix Factorization in SMORe

Graph

Sampler

Optimizer

(user, item)

(user, item)

Embedding Lookup

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)

(user, item)
Sparse Llinear Method (SLIM)

\[
\begin{align*}
\text{minimize} & \quad \frac{1}{2} \| A - AW \|_F^2 \\
\text{subject to} & \quad W \geq 0 \\
& \quad \text{diag}(W) = 0,
\end{align*}
\]

Transition Matrix \( W \)

SLIM in SMORe

Graph

Sampler

(user, item)
SLIM in **SMORe**

**Graph**
- User-item graph

**Sampler**
- (user, item)

**Mapper**
- \( A \) (fixed)
- \( W \) (trainable)

\[
\text{minimize} \quad \frac{1}{2} \| A - AW \|_F^2 \\
\text{subject to} \quad W \geq 0 \\
\text{diag}(W) = 0,
\]
SLIM in **SMORe**

**Graph**
user-item graph

**Sampler**
(user, item)

**Optimizer**
\((\vec{v} \cdot \vec{v} - y)^2\)

**Square Error**

**Mapper**
A (fixed)  W (trainable)
2. Neighborhood
Neighborhood

vertices share similar connections are treated as similar pairs

common interests

commonly purchased
determines the number of negative examples to be sampled from the unigram distribution raised to the power of the sample frequency of the word. This distribution was found to significantly outperform the unigram distribution, as the empirical study in [8] reported.

To avoid the issues mentioned earlier, we propose to apply SGNS to item-based CF. The goal of this application is to learn item representations that capture the similarity between items, which can be used to recommend items to users. The proposed method handles these scenarios as well, as the experimental results in Section 4 show.

In this work, we used the terms "word" and "item" interchangeably, since in this paper, we assume a static environment where items that share the same set are always considered similar, no matter in what order/time they are observed.

We propose to apply SGNS to item-based CF. The algorithm described in Section 2. We name the proposed method "Item2Vec".


4. Datasets

4.1 Datasets

The first dataset is user-artist data that is retrieved from the Microsoft XBOX Music service. This dataset contains 732K user-artists and 49K distinct artists. The specific artist. The dataset contains 732K user-artists and 49K distinct artists. The dataset consists of 379K orders (that contain more than a single item) and 1706 distinct items. The dataset consists of 379K orders (that contain more than a single item) and 1706 distinct items. The dataset consists of 379K orders (that contain more than a single item) and 1706 distinct items. The dataset consists of 379K orders (that contain more than a single item) and 1706 distinct items. The dataset consists of 379K orders (that contain more than a single item) and 1706 distinct items. The dataset consists of 379K orders (that contain more than a single item) and 1706 distinct items.

Another option is to keep the objective in Eq. (1) as is, and the affinity between a pair of items is computed by the cosine similarity.

Since we ignore the spatial information, we treat the given K items as the final representation of the user.

\[
\frac{1}{K} \sum_{i=1}^{K} \sum_{j \neq i} \log p(w_j | w_i)
\]


To Model Neighborhood

userA

userB

purchasing logs

space for shared neighbor

vertex space

context space

w2v
item2vec in **SMORe**

Graph

Sampler

\[(item_A, item_B)\]

(user-item graph)
item2vec in **SMORe**

**Graph**

**Sampler**

\[(itemA, itemB)\]

**Mapper**

\[\vec{u}_A, \vec{v} \rightarrow \vec{v}, \vec{u}_B \rightarrow \vec{v}_c\]

- **vertex embedding lookup**
- **context embedding lookup**
item2vec in **SMORE**

**Graph**

**Sampler**

\[(\vec{v}_A, \vec{v}_B)\]

(itemA, itemB)

**Optimizer**

\[
\log \sigma (\vec{v}_A \cdot \vec{v}_B)
\]

Log Likelihood

**Mapper**

user-item graph

vertex embedding lookup

context embedding lookup
3. Community
vertices share similar communities are treated as similar pairs

connected to the same group
Heterogeneous Preference Embedding (HPE) RecSys’16

Figure 2: Edge sampling via weighted random walks for Heterogeneous Preference Embedding (HPE). The sampling strategy involves selecting training pairs from the whole network, followed by weighted random walks from the preceding vertex to the target vertex. The training pairs are denoted as \((i, j)\) in the set \(S\), and the weight associated with each pair is denoted as \(w_{i,j}\). The goal is to optimize the objective function as

\[
\sum_{(i, j) \in S} w_{i,j} \log p(v_j | \Phi(v_i))
\]

where \(v_i\) and \(v_j\) represent the embeddings of vertices and \(\Phi\) is the function that maps vertices to their embeddings. The function \(p(v_j | \Phi(v_i))\) represents the probability of observing \(v_j\) given the embedding \(\Phi(v_i)\) of vertex \(v_i\).
HPE in **SMORe**

**Graph**

**Sampler**

(user-item graph)

(V, C) (V, C) (V, C)
HPE in **SMORe**

**Graph**

user-item graph

**Sampler**

\((V, C) (V, C) (V, C)\)

**Optimizer**

\[ \log \sigma(\tilde{v}_V \cdot \tilde{v}_C) \]

**Mapper**

- **vertex embedding lookup**
  - \( V \rightarrow \tilde{v}_V \)

- **context embedding lookup**
  - \( C \rightarrow \tilde{v}_C \)
4. Centrality (Degree)
Centrality

vertices share similar properties are treated as similar pairs
Figure 2: (a) Barbell graph. (b) Roles identified by RolX. Latent representations in $\mathbb{R}^2$ learned by (c) DeepWalk, (d) node2vec and (e,f,g,h) struc2vec. Parameters used for all methods: number of walks per node: 20, walk length: 80, skip-gram window size: 5. For node2vec: $p = 1$ and $q = 2$.

makes it impossible for nodes in $K_1$ and $K_2$ to appear in the same context. struc2vec, on the other hand, learns representations that properly separate the equivalent classes, placing structurally equivalent nodes near one another in the latent space. Note that nodes of the same color are tightly grouped together. Moreover, $p_1$ and $p_{10}$ are placed close to representations for nodes in $K_1$ and $K_2$, as they are the bridges. Finally, note that none of the three optimizations have any significant effect on the quality of the representations. In fact, structurally equivalent nodes are even closer to one another in the latent representations under OPT1.

Last, we apply RolX to the barbell graph (results in Figure 2(b)). A total of six roles were identified and some roles indeed precisely captured structural equivalence (roles 1 and 3). However, structurally equivalent nodes (in $K_1$ and $K_2$) were placed in three different roles (role 0, 2, and 5) while role 4 contains all remaining nodes in the path.

Although RolX does capture some notion of structural equivalence when assigning roles to nodes, struc2vec better identifies and separates structural equivalence.

4.2 Karate network

Zachary’s Karate Club is a network composed of 34 nodes and 78 edges, where each node represents a club member and edges denote if two members have interacted outside the club. In this network, edges are commonly interpreted as indications of friendship between members.

We construct a network composed of two copies $G_1$ and $G_2$ of the Karate Club network, where each node $v_i \in V(G_1)$ has a mirror node $u_i \in V(G_2)$. We also connect the two networks by adding an edge between mirrored node pairs 1 and 37. Although this is not necessary for our framework, DeepWalk and node2vec cannot place in the same context nodes in different connected components of the graph. Instead, we add the edge for a more fair comparison.
struc2vec  

KDD’17

struc2vec in **SMORe**

Graph

user-user graph

Sampler

$$S_{fn1()}(V, S)$$

$$S_{fn2()}(V, S)$$
struc2vec in **SMORe**

Graph

**user-user graph**

Sampler

fn1() \[ (V, S) \]

fn2() \[ (V, S) \]

Optimizer

\[
\log \sigma(\overrightarrow{v}_V \cdot \overrightarrow{v}_S) 
\]

Log Likelihood

Mapper

\[ V \rightarrow \overrightarrow{v}_V \]

vertex embedding lookup

\[ S \rightarrow \overrightarrow{v}_S \]

context embedding lookup

note. original work adopts hierarchal softmax for learning
## Recap

<table>
<thead>
<tr>
<th></th>
<th>Sampler</th>
<th>Mapper</th>
<th>Optimizer</th>
</tr>
</thead>
<tbody>
<tr>
<td>MF</td>
<td>Adjacency</td>
<td>Vertex Embedding</td>
<td>Dot product RMSE</td>
</tr>
<tr>
<td>SLIM</td>
<td>Adjacency</td>
<td>Vertex Embedding</td>
<td>Dot product RMSE</td>
</tr>
<tr>
<td>item2vec</td>
<td>Neighborhood</td>
<td>Vertex Embedding, Context Embedding</td>
<td>Sigmoid Dot Product Log Likelihood</td>
</tr>
<tr>
<td>HPE</td>
<td>Community</td>
<td>Vertex Embedding, Context Embedding</td>
<td>Sigmoid Dot Product Log Likelihood</td>
</tr>
<tr>
<td>struc2vec</td>
<td>Centrality</td>
<td>Vertex Embedding, Context Embedding</td>
<td>Sigmoid Dot Product Log Likelihood</td>
</tr>
</tbody>
</table>
Edge Information

- **Weight**
  - reflects strength of interaction

- **Closeness**
  - reflects order of interaction

- **Semantics**
  - sample with grammar

- **Inductiveness**
  - information diffuses through edges
1. Weight
node2vec  KDD’16

edge weight guides the walk

node2vec in **SMORe**

**Sampler**
customized random walk
2. Closeness

strong

weak
High-order Proximity for Recommendations (HOP-Rec)

order-aware interaction

\[
\sum_{1 \leq k \leq K} C(k) \mathbb{E}_{u \sim P^k_u, i' \sim P_N} \left[ \mathcal{F} \left( \theta_u^T \theta_{i'}, \theta_u^T \theta_i \right) \right]
\]

\[
\mathbb{1} \left\{ \theta_u^T \theta_{i'} - \theta_u^T \theta_i > \epsilon_k \right\} \log \left[ \sigma \left( \theta_u^T \theta_{i'} - \theta_u^T \theta_i \right) \right]
\]

Figure 1: High-order proximity between users and items within observed interactions
HOP-Rec in SMORe

Graph

Sampler

user-item graph

\[ 1(\vec{v} \cdot \vec{v} - \vec{\hat{v}} \cdot \vec{\hat{v}} < \text{margin}) \]
HOP-Rec in **SMORe**

**Graph**

- User-item graph

**Sampler**

- Greater margin
- Smaller margin

\[ 1(\vec{v} \cdot \vec{v} - \vec{v} \cdot \vec{v} < \text{margin}) \]

**Optimizer**

\[ \log \sigma(\vec{v} \cdot \vec{v} - \vec{v} \cdot \vec{v}^R) \]

**Mapper**

- Embedding lookup

---

3. Semantics

[Diagram showing connections between user, venue, and other nodes]
We present a general framework, where to model the heterogeneous neighborhood of a node, various heterogeneous network mining tasks. For example, the Heterogeneous Skip-Gram model achieves this and utilizes the skip-gram model to learn the representation of data, wherein it is di

3.2 Heterogeneous Network Embedding: The key to disentangle the heterogeneity, wherein it is difficult to capture the structural and semantic relations among them. Inspired by the P–V model, we can put the meta-path scheme 'OAPVPAO', for example, the walker is biased to generate paths of multiple types of nodes. At step $i$, the next step of a walker is conditioned on the previous node, i.e., $P_r, P_l$. Naturally, we can put the meta-path-based random walk strategy ensures that the walker is conditioned on the previous node, i.e., $P_r, P_l$. In other words, the walker is conditioned on the previous node, i.e., $P_r, P_l$. Therefore, we have the following objective:

\[
\arg \max_\theta \sum_{v \in V} \sum_{t \in T_v} \sum_{c_t \in N_t(v)} \log p(c_t | v; \theta)
\]

different types

neighbors in given type
metapath2vec in SMORe

Sampler

\[ \mathcal{P}: V_1 \xrightarrow{R_1} V_2 \xrightarrow{R_2} \cdots V_t \xrightarrow{R_t} V_{t+1} \cdots \xrightarrow{R_{l-1}} V_l \]

grammar1: Paper -> Author -> Paper

to find context from
the same author

grammar2: Paper -> Venue -> Paper

to find context from
the same conference
4. Inductiveness
Inductiveness

1. vertex = another vertex + relation

2. vertex = pooling(neighbors)
Translation-based Recommendations (TransRec) RecSys’18

Translation operation:
prev. item + user ≈ next item

TransRec in **SMORe**

Graph

Sampler

(user-item graph)

(itemA, user, itemB)
TransRec in **SMORe**

**Graph**

- User-item graph

**Sampler**

\((itemA, user, itemB)\)

**Optimizer**

minimize the distance

\[ d(\vec{v}_A + \vec{v}_B, \vec{v}_B) \]

**Mapper**

- Embedding LookUp

\[ \text{A} \rightarrow \vec{v}_A \]

\[ \text{B} \rightarrow \vec{v}_B \]
Graph Convolution for Recommendation (Pinsage) KDD’18

Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, Jure Leskovec: Graph Convolutional Neural Networks for Web-Scale Recommender Systems. KDD 2018: 974-983
Graph Convolution for Recommendation (Pinsage)  

**Figure 1:** Overview of our model architecture using depth-2 convolutions (best viewed in color). Left: A small example input graph. Right: The 2-layer neural network that computes the embedding of a node of the input graph. Bottom: The neural networks that compute embeddings of each node of the input graph.

- **TARGET NODE**
- **INPUT GRAPH**
- **Sampler()**
- **CONVOLVE**(2)
  (See Algorithm 1)

![Graph Convolution for Recommendation](image)

**Algorithm 1:**

```plaintext
Sampler()

CONVOLVE(2) (See Algorithm 1)

h_A(2)

Sampler()

CONVOLVE(1)

h_B(1)

h_C(1)

h_D(1)
```

**References:**

Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, Jure Leskovec: Graph Convolutional Neural Networks for Web-Scale Recommender Systems. KDD 2018: 974-983
Graph Convolution for Recommendation (Pinsage)

**Figure 1:** Overview of our model architecture using depth-2 convolutions (best viewed in color). Left: A small example input graph. While neural networks do not explicitly consider neighborhood, the parameters of the convolutional layers (nodes A, B, C, D, E, F) need to be included (Section 3.2). Bottom: The neural networks that compute embeddings of each node of the input graph. Right: The 2-layer neural network that computes the embedding of node A.

**Algorithm 1:**

```python
CONVOLVE^{(2)}(A)
CONVOLVE^{(1)}(B)
Mapper()
Sampler()
```

- **Sampler()**
  - Target Node: A
  - Input Graph: A, B, C, D, E, F

- **CONVOLVE^{(2)}**
  - (See Algorithm 1)
  - self
  - neighbor
  - \( h_A^{(1)} \)
  - \( h_{N(A)}^{(1)} \)
  - Concatenation
  - Importance pooling

- **Mapper()**
  - A: \( h_A^{(1)} \)
  - B: \( h_B^{(1)} \)
  - C: \( h_C^{(1)} \)
  - D: \( h_D^{(1)} \)
  - E: \( h_E^{(1)} \)
  - F: \( h_F^{(1)} \)

- **Sampler()**
  - A: \( h_A^{(2)} \)
  - B: \( h_B^{(2)} \)
  - C: \( h_C^{(2)} \)
  - D: \( h_D^{(2)} \)
  - E: \( h_E^{(2)} \)
  - F: \( h_F^{(2)} \)
## Recap

<table>
<thead>
<tr>
<th>Sampler</th>
<th>Mapper</th>
<th>Optimizer</th>
</tr>
</thead>
<tbody>
<tr>
<td>node2vec</td>
<td>Community (Weight)</td>
<td>Vertex Embedding</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Context Embedding</td>
</tr>
<tr>
<td>HOP-Rec</td>
<td>Community (Closeness)</td>
<td>Vertex Embedding</td>
</tr>
<tr>
<td>metapath2vec</td>
<td>Community (Semantics)</td>
<td>Vertex Embedding</td>
</tr>
<tr>
<td>TransRec</td>
<td>Adjacency (Inductiveness)</td>
<td>Vertex Embedding</td>
</tr>
<tr>
<td>Pinsage</td>
<td>Adjacency (Inductiveness)</td>
<td>Vertex / Context Embedding</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Pooling + Concatenation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Sigmoid Dot Product Log Likelihood</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Dot Product BPR</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Sigmoid Dot Product Log Likelihood</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Dot product BPR</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Fully Connected Layer</td>
</tr>
</tbody>
</table>
Lecture Agenda

(Sean) Q0. Recommendation (REC) and challenges

(Sean) Q1. Why graph embedding (GE) for REC

(Sean) Q2. SMORe modularization of GE and benefits

(CM) Q3. Exemplar structural modeling for REC

(CM) Q4. REC using SMORe
Graph Manipulation

- **Connectivity**
  - modify the edge weights
- **Augmentation**
  - extend the connections from external knowledge
Superhighway for Cross-Domain CF


1 INTRODUCTION

Cross-domain collaborative filtering (CF) aims to alleviate data sparsity in single-domain CF, which is highly susceptible to data sparsity as the method analyzes observed user-item interactions through shared items. To enrich the cross-domain connectivity, the superhighway construction bypasses multi-hop inter-domain paths between cross-domain users and target domain items, otherwise, assuming partially overlapped items (users), superhighway recommendations by enhancing cross-domain connectivity. Specifically, assuming partially overlapped items (users), superhighway recommendations by enhancing cross-domain connectivity. Specifically, assuming partially overlapped items (users), superhighway recommendations by enhancing cross-domain connectivity.

Figure 1: Illustrative example for superhighways
Superhighway in SMORe

Graph

\[
\begin{align*}
\mathcal{U}_S & \quad \mathcal{I}_S \\
\mathcal{I}_T & \quad \mathcal{U}_T \\
\end{align*}
\]

\[
\mathcal{U}^\wedge_S \quad \mathcal{U}^\wedge_T
\]

merged graph
• Multi-Task Learning
  - shared representations for multi-tasks
Collaborative Similarity Embedding (CSE)

1. model preference

2. cluster users

3. cluster items

Neighborhood Similarity Embedding (NSEmbed)

Direct Similarity Embedding (DSEmbed)

\[ \Phi_{UC} \]

\[ \mathcal{L}_{NS} \]

\[ \Phi_{IC} \]

\[ \mathcal{L}_{DS} \]
CSE in **SMORe**

**Graph**

- **Sampler1** (user, item)
- **Sampler2** (user, u_community)
- **Sampler3** (item, i_community)

**Sampler1**

**Sampler2**

**Sampler3**

**User-item graph**
CSE in **SMORe**

Graph

**Mapper1**
- Vertex Embedding LookUp
  - $u_1 \rightarrow \bar{u}_1$
  - $u_1 \rightarrow \bar{u}_1$

**Mapper2**
- Context Embedding LookUp
  - $u_2 \rightarrow \bar{u}_2$

**Mapper3**
- Context Embedding LookUp
  - $u_3 \rightarrow \bar{u}_3$

user-item graph

M1 user-item preference

M2 cluster users

M3 cluster items
CSE in SMORe

\[
\log \sigma(\mathbf{v}_1 \cdot \mathbf{u}_1) + \log \sigma(\mathbf{v}_1 \cdot \mathbf{u}_2) + \log \sigma(\mathbf{v}_1 \cdot \mathbf{u}_3)
\]

Optimizer 1

Log Likelihood

Optimizer 2

Log Likelihood

Optimizer 3

Log Likelihood

M_1 user-item preference

M_2 cluster users

M_3 cluster items
Coming Next …

- Lecture (Sean & CM, 65 minutes)
- Hands-on (CM, 15 minutes)
- Q&A (Sean & CM, 10 minutes)

QR to Slides, Codes, Abstract
SMORe

https://github.com/cnclabs/smore

Compressed Sparse Row (CSR) + Alias Method
SMORE

SMORE: Modularize Graph Embedding for Recommendation

Compilation

1. clone it
   - `git clone https://github.com/cnclabs/somore`
2. enter it
   - `cd smore`
3. compile it
   - `make`
SMORe

- DeepWalk
  - DeepWalk: online learning of social representations
- Walklets
  - Don't Walk, Skip! Online Learning of Multi-scale Network Embeddings
- LINE (Large-scale Information Network Embedding)
  - LINE: Large-scale Information Network Embedding
- HPE (Heterogeneous Preference Embedding)
  - Query-based Music Recommendations via Preference Embedding
- APP (Asymmetric Proximity Preserving graph embedding)
  - Scalable Graph Embedding for Asymmetric Proximity
- MF (Matrix Factorization)
- BPR (Bayesian Personalized Ranking)
  - BPR: Bayesian personalized ranking from implicit feedback
- WARP-like
  - WSABIE: Scaling Up To Large Vocabulary Image Annotation
  - Learning to Rank Recommendations with the k-Order Statistic Loss
- HOP-REC
  - HOP-Rec: High-Order Proximity for Implicit Recommendation
- CSE (named nemf & nerank in cli)
  - Collaborative Similarity Embedding for Recommender Systems
SMORe for Most End Users

Options Description:
- `train <string>`
  Train the Network data
- `save <string>`
  Save the representation data
- `dimensions <int>`
  Dimension of vertex representation; default is 64
- `undirected <int>`
  Whether the edge is undirected; default is 1
- `negative_samples <int>`
  Number of negative examples; default is 5
- `window_size <int>`
  Size of skip-gram window; default is 5
- `walk_times <int>`
  Times of being staring vertex; default is 10
- `walk_steps <int>`
  Step of random walk; default is 40
- `threads <int>`
  Number of training threads; default is 1
- `alpha <float>`
  Init learning rate; default is 0.025

Usage:
```
./deepwalk -train net.txt -save rep.txt -undirected 1 -dimensions 64 -walk_times 10 -walk_steps 40 -window_size 5 -n
```
SMORe for Most End Users

Graph as input (edge list)

Embeddings as output
• On-Going Work
SMORE (another modularized version)

Find the branch *smore*

It's under refactoring …
SMORe for Developers

Graph

Sampler()

Mapper()

Optimizer()

Embeddings for Recommendations
**SMORe** Example Codes (in smore branch)

```cpp
// main
// 0. [Graph] read from file-based graph
FileGraph *file_graph = new FileGraph(path, 0);

// 1. [Sampler] determine what sampler to be used
VCSampler sampler(file_graph);

// 2. [Mapper] define what embedding mapper to be used
LookupMapper mapper(sampler.vertex_size, dimension);

// 3. [Optimizer] claim the optimizer
PairwiseOptimizer optimizer;
```
SMORe Factorization (in smore branch)

```python
while (update < worker_update_times):
    
    // 4.0 reset user batch loss
    user_batch_loss.assign(dimension, 0.0);
    item_loss.assign(dimension, 0.0);

    // 4.1 sample positive (user, item) pair, feed the loss, update
    user = sampler.draw_a_vertex();
    item = sampler.draw_a_context(user);
    optimizer.feed_loglikelihood_loss(mapper[user], mapper[item], 1.0, dimension, user_batch_loss, item_loss);
    mapper.update_with_l2(item, item_loss, alpha, 0.001);
    item_loss.assign(dimension, 0.0);

    // 4.2 sampler negative (user, item) pair, feed the loss, update
    for (int n=0; n<negative; n++)
        
    item = sampler.draw_a_negative();
    optimizer.feed_loglikelihood_loss(mapper[user], mapper[item], 0.0, dimension, user_batch_loss, item_loss);
    mapper.update_with_l2(item, item_loss, alpha, 0.001);
    item_loss.assign(dimension, 0.0);

mapper.update_with_l2(user, user_batch_loss, alpha, 0.01);
```
SMORe Factorization (in smore branch)

```c
while (update < worker_update_times)
{
    // 4.0 reset user batch loss
    user_batch_loss.assign(dimension, 0.0);
    item_loss.assign(dimension, 0.0);

    // 4.1 sample positive (user, item) pair, feed the loss, update
    user = sampler.draw_a_vertex();
    item = sampler.draw_a_context(user);
    optimizer.feed_loglikelihood_loss(mapper[user], mapper[item], 1.0, dimension, user_batch_loss, item_loss);
    mapper.update_with_l2(item, item_loss, alpha, 0.001);
    item_loss.assign(dimension, 0.0);

    // 4.2 sampler negative (user, item) pair, feed the loss, update
    for (int m=0; m<negative; m++)
    {
        item = sampler.draw_a_negative();
        optimizer.feed_loglikelihood_loss(mapper[user], mapper[item], 0.0, dimension, user_batch_loss, item_loss);
        mapper.update_with_l2(item, item_loss, alpha, 0.001);
        item_loss.assign(dimension, 0.0);
    }
    mapper.update_with_l2(user, user_batch_loss, alpha, 0.01);
}
```
SMORe Factorization (in smore branch)

```cpp
while (update < worker_update_times) {
    // 4.0 reset user batch loss
    user_batch_loss.assign(dimension, 0.0);
    item_loss.assign(dimension, 0.0);

    // 4.1 sample positive (user, item) pair, feed the loss, update
    user = sampler.draw_a_vertex();
    item = sampler.draw_a_context(user);
    optimizer.feed_loglikelihood_loss(mapper[user], mapper[item], 1.0, dimension, user_batch_loss, item_loss);
    mapper.update_with_l2(item, item_loss, alpha, 0.001);
    item_loss.assign(dimension, 0.0);

    // 4.2 sampler negative (user, item) pair, feed the loss, update
    for (int n=0; n<negative; n++) {
        item = sampler.draw_a_negative();
        optimizer.feed_loglikelihood_loss(mapper[user], mapper[item], 0.0, dimension, user_batch_loss, item_loss);
        mapper.update_with_l2(item, item_loss, alpha, 0.001);
        item_loss.assign(dimension, 0.0);
    }
    mapper.update_with_l2(user, user_batch_loss, alpha, 0.01);
}
```

run it until conditions hold
sample positive (user, item) pair
SMORe Factorization (in smore branch)

run it until conditions hold

while (update < worker_update_times) {

// 4.0 reset user batch loss
user_batch_loss.assign(dimension, 0.0);
item_loss.assign(dimension, 0.0);

// 4.1 sample positive (user, item) pair, feed the loss, update
user = sampler.draw_a_vertex();
item = sampler.draw_a_context(user);

optimizer.feed_loglikelihood_loss(mapper[user], mapper[item], 1.0, dimension, user_batch_loss, item_loss);
mapper.update_with_l2(item, item_loss, alpha, 0.001);
item_loss.assign(dimension, 0.0);

// 4.2 sampler negative (user, item) pair, feed the loss, update
for (int n=0; n<negative; n++) {

item = sampler.draw_a_negative();
optimizer.feed_loglikelihood_loss(mapper[user], mapper[item], 0.0, dimension, user_batch_loss, item_loss);
mapper.update_with_l2(item, item_loss, alpha, 0.001);
item_loss.assign(dimension, 0.0);
}
mapper.update_with_l2(user, user_batch_loss, alpha, 0.01);

sample positive (user, item) pair
map user/item to its embedding
SMORe Factorization (in smore branch)

```java
while (update < worker_update_times) {
    // 4.0 reset user batch loss
    user_batch_loss.assign(dimensions, 0.0);
    item_loss.assign(dimensions, 0.0);

    // 4.1 sample positive (user, item) pair, feed the loss, update
    user = sampler.draw_a_vertex();
    item = sampler.draw_a_context(user);
    optimizer.feed_loglikelihood_loss(mapper[user], mapper[item], 1.0, dimension, user_batch_loss, item_loss);
    mapper.update_with_l2(item, item_loss, alpha, 0.001);
    item_loss.assign(dimensions, 0.0);

    // 4.2 sampler negative (user, item) pair, feed the loss, update
    for (int n=0; n<negative; n++) {
        item = sampler.draw_a_negative();
        optimizer.feed_loglikelihood_loss(mapper[user], mapper[item], 0.0, dimension, user_batch_loss, item_loss);
        mapper.update_with_l2(item, item_loss, alpha, 0.001);
        item_loss.assign(dimensions, 0.0);
    }
    mapper.update_with_l2(user, user_batch_loss, alpha, 0.01);
}
```
**SMORe** Factorization (in smore branch)

```java
while (update < worker_update_times) {
    // 4.0 reset user batch loss
    user_batch_loss.assign(dimension, 0.0);
    item_loss.assign(dimension, 0.0);
    // 4.1 sample positive (user, item) pair, feed the loss, update
    user = sampler.draw_a_vertex();
    item = sampler.draw_a_context(user);
    optimizer.feed_loglikelihood_loss(mapper[user], mapper[item], 1.0, dimension, user_batch_loss, item_loss);
    mapper.update_with_l2(item, item_loss, alpha, 0.001);
    item_loss.assign(dimension, 0.0);
    // 4.2 sampler negative (user, item) pair, feed the loss, update
    for (int n=0; n<negative; n++) {
        item = sampler.draw_a_negative();
        optimizer.feed_loglikelihood_loss(mapper[user], mapper[item], 0.0, dimension, user_batch_loss, item_loss);
        mapper.update_with_l2(item, item_loss, alpha, 0.001);
        item_loss.assign(dimension, 0.0);
    }
    mapper.update_with_l2(user, user_batch_loss, alpha, 0.01);
}
```

run it until conditions hold

sample positive (user, item) pair
map user/item to its embedding
estimate the loss from log likelihood
update embedding
**SMORe Factorization (in smore branch)**

```java
while (update < worker_update_times) {
    // 4.0 reset user batch loss
    user_batch_loss.assign(dimension, 0.0);
    item_loss.assign(dimension, 0.0);

    // 4.1 sample positive (user, item) pair, feed the loss, update
    user = sampler.draw_a_vertex();
    item = sampler.draw_a_context(user);
    optimizer.feed_loglikelihood_loss(mapper[user], mapper[item], 1.0, dimension, user_batch_loss, item_loss);
    mapper.update_with_l2(item, item_loss, alpha, 0.001);
    item_loss.assign(dimension, 0.0);

    // 4.2 sample negative (user, item) pair, feed the loss, update
    for (int n=0; n<negative; n++) {
        item = sampler.draw_a_negative();
        optimizer.feed_loglikelihood_loss(mapper[user], mapper[item], 0.0, dimension, user_batch_loss, item_loss);
        mapper.update_with_l2(item, item_loss, alpha, 0.001);
        item_loss.assign(dimension, 0.0);
    }
}
```

Sample positive (user, item) pair

Map user/item to its embedding

Estimate the loss from log likelihood

Update embedding

Mode negative (user, item) pair
Graph

Sampler()

Mapper()

Optimizer()

Embeddings

[0.08 0.02 0.28]

[-0.31 -0.1 0.1]

[-0.15 -0.3 0.2]
Graph

unit1
Sampler()
Mapper()
Optimizer()

unit2
Sampler()
Mapper()
Optimizer()

unit3
Sampler()
Mapper()
Optimizer()

unit4
Sampler()
Mapper()
Optimizer()

Embeddings

[0.08 0.02 0.28]

[-0.31 -0.1 0.1]

[-0.15 -0.3 0.2]
Multi-threading

Practice

Embeddings

[0.08 0.02 0.28]

[-0.31 -0.1 0.1]

[-0.15 -0.3 0.2]

HOGWILD!


https://www.reddit.com/r/aww/comments/2oagj8/multithreaded_programming_theory_and_practice/
Characteristics of **SMORe**

Handles complex systems of interactions as a unified graph structure, allowing **joint mining** of diverse information to address core problems for REC such as **data sparsity** and **cold start**

Generalizes relations as **graph structures** composed of **vertices** and **edges**. Models can explore a spectrum of complex structures and their combinations for any given REC tasks

Breaks GE into **sampler**, **mapper**, and **optimizer**; which extracts interactions as structures, converts entities into spatial features, and preserves relatedness as spatial properties, respectively
Benefits of **SMORe**

**Speedy**

**Speeds development** by reusing codes; **provides model toolkit** for REC and fair baseline comparison; and **accelerates training process** using CSAR and HOGWILD!

**Effective**

**Adapts to different REC needs** on module-level for embedding and structure-level for relations; also **opens to deep methods**, which continue to churn out SOTA models over the past years.

**Multi-task**

**Jointly captures different relations** from HINs by selecting graph structures using sampler and combining embeddings using mapper.
Coming Next …

Lecture (Sean & CM, 65 minutes)

Hands-on (CM, 15 minutes)

Q&A (Sean & CM, 10 minutes)

QR to Slides, Codes, Abstract
SMORe:

SMORe Members at CFDA & CLIP Labs: https://cfda.csie.org/

Chih-Ming Chen
104761501@nccu.edu.tw

Ting-Hsiang Wang
thwang1231@tamu.edu

Chuan-Ju Wang
cjwang@citi.sinica.edu.tw

Ming-Feng Tsai
mftsai@nccu.edu.tw

Special Thanks to:

Chih-Ming Chen
104761501@nccu.edu.tw

Ting-Hsiang Wang
thwang1231@tamu.edu

Chuan-Ju Wang
cjwang@citi.sinica.edu.tw

Ming-Feng Tsai
mftsai@nccu.edu.tw

Link to Codes, Slides, Abstract:
https://github.com/cnclabs/smore/