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

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. We demonstrate DeepWalk in unsupervised feature learning (or word2vec) from sequences of words to graphs. The algorithm works by treating a graph as a special language composed of a set of randomly-generated walks. These neural language models have been used to capture the semantic and syntactic structure of human language, and even logical analogies. To demonstrate DeepWalk's potential in real world scenarios, we develop an 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, recommendation, anomaly detection, and missing link prediction.

Figure 1: Our proposed method
(a) Input: Karate Graph
(b) Output: Representation

DeepWalk takes a graph as input and produces a latent space representation. The learned representations can provide insights into the structure of the input graph, such as community detection and clustering of the input graph. Beyond the striking similarity, we note that linearly separable portions of (b) correspond to clusters found through modularity maximization in the input graph (a) (shown as vertex colors).
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.

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**

→ 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

→ 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 (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)**.

> Item attributes are added to improve item comparison, e.g., music genre, artist, metadata, etc.
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).
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
- Delta Airline Route Map Network
- Twitter Ego-Network
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

User

Item

The quick brown fox jumps over the lazy dog.
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
• **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
Graph Embedding Pipeline

Supply Graph

Train Neural Network

Return Embeddings (from hidden Layer)

- 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]

→ 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?

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
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

- 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 …

Graph structures

Samples relation

Maps entities to space

Embedding space
Let’s look at GE process …

Graph structures

Samples relation

Maps entities to space

Embedding space

Optimizes distance
Challenge (1) : Select structure is HARD

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

Graph structures

Maps entities to space

Samples relation

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!

- Sampler
- Mapper
- Optimizer

Graph structures
- Maps entities to space
- Embedding space
- Optimizes distance
- Samples relation

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
- Sampler
- Mapper
- Optimizer

Images of graphs and embeddings:
- Samples relation
- Maps entities to space
- Embedding space
- Optimizes distance
**SMORe** : Modular GE toolkit for REC

**Graph structures**
- Samples relation
- Maps entities to space
- Embedding space
- Optimizes distance

**Components**
- **Sampler**
- **Mapper**
- **Optimizer for Recommendation**
**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**

Modules contribute different levels of performance during different REC tasks.
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.
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
Adjacency

vertices share the same edge are treated as similar pairs
Matrix Factorization

\[
\begin{array}{cccc}
\text{U1} & \text{U2} & \text{U3} & \text{I1} \\
\hline
r & & & \\
\end{array}
\quad = \quad
\begin{array}{cccc}
\text{q1} & \text{q2} & \text{q3} & \text{q4} \\
\text{p1} & ? & ? & ? \\
\text{p2} & ? & ? & ? \\
\text{p3} & ? & ? & ? \\
\end{array}
\]

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

(rating - item*user)

---

Matrix Factorization in SMORe

Graph

user-item graph
Matrix Factorization in SMORe

Graph

Sampler

(user, item)
Matrix Factorization in **SMORe**

**Graph**

- User
- Item
- User-item graph

**Sampler**

- Sampled user-item graph

**Mapper**

- User → Embedding
- Item → Embedding

**Embedding Lookup**
Matrix Factorization in **SMORe**

**Graph**

**Sampler**

\[(\text{user}, \text{item})\]

**Optimizer**

\[(\vec{v} \cdot \vec{v} - y)^2\]

**Square Error**

**Mapper**

Embedding Lookup

user-item graph

(user, item)
Sparse Linear Method (SLIM)  

<table>
<thead>
<tr>
<th>I1</th>
<th>I2</th>
<th>I3</th>
<th>I4</th>
</tr>
</thead>
<tbody>
<tr>
<td>U1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>U2</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>U3</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

Transition Matrix \( W \)
SLIM in SMORe

Graph

Sampler

(user, item)
SLIM in SMORe

Graph

Sampler

Mapper

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

**Graph**

- User-item graph

**Sampler**

\((\text{user, item})\)

**Optimizer**

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

**Square Error**

**Mapper**

- \(A\) (fixed)
- \(W\) (trainable)
2. Neighborhood
Neighborhood

vertices share similar connections are treated as similar pairs

common interests  commonly purchased
item2vec \textit{RecSys’16, MLSP’16}

The given K items

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

purchasing logs

Oren Barkan, Noam Koenigstein: ITEM2VEC: Neural item embedding for collaborative filtering. MLSP 2016: 1-6
To Model Neighborhood

userA

userB

purchasing logs

shared neighbor

vertex space

context space

space for shared neighbor

w2v
item2vec in **SMORe**

Graph

user-item graph

Sampler

\[(itemA, itemB)\]

A
B

A
B
item2vec in **SMORE**

**Graph**

- user-item graph

**Sampler**

- \((itemA, itemB)\)

**Mapper**

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

**Graph**

- User-item graph

**Sampler**

- \((A, B)\)
- \((\text{itemA, itemB})\)

**Optimizer**

- \(\log \sigma (\vec{v}_A^V \cdot \vec{v}_B^C)\)
- Log Likelihood

**Mapper**

- Vertex embedding lookup
- Context embedding lookup
3. Community
Community

vertices share similar communities are treated as similar pairs

not connected

connected to the same group
Heterogeneous Preference Embedding (HPE)  

Heterogeneous Preference Embedding (HPE)

Walk Direction

Edge Sampling from whole network

Weighted Random Walks from preceding vertex

vertex

training pairs

weight

embedding

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

HPE in **SMORe**

**Graph**

**Sampler**

user-item graph

\[ (V, C) (V, C) (V, C) \]
HPE in **SMORe**

**Graph**

user-item graph

**Sampler**

\[(V, C) (V, C) (V, C)\]

**Mapper**

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

vertex embedding lookup

\[C \rightarrow \vec{v}_C\]

context embedding lookup

**Optimizer**

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

Log Likelihood
4. Centrality (Degree)
Centrality

vertices share similar properties are treated as similar pairs
Figure 2: (a) Barbell graph \( B(10, 10) \). (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. \n
struct2vec, 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.

### 4.2 Karate network

Zachary's Karate Club [25] 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 in \( V(G_1) \) has a mirror node 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.

---

Figure 2: (a) Barbell graph $B(10,10)$. (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.

4.2 Karate network

Zachary's Karate Club \cite{zachary1977information} 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 $v \in V(G_1)$ has a mirror node $u \in V(G_2)$. 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.

struc2vec in **SMORe**

**Graph**

**Sampler**

(user-user graph)
struc2vec in **SMORe**

**Graph**

user-user graph

**Sampler**

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

**Optimizer**

Log Likelihood

**Mapper**

vertex embedding lookup

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  

edge weight guides the walk

KDD 2016: 855-864
node2vec in **SMORe**

**Sampler**

*customized random walk*
2. Closeness
High-order Proximity for Recommendations (HOP-Rec)

order-aware interaction

\[
\sum_{1 \leq k \leq K} C(k) \mathbb{E}_{i \sim P_u^k, i' \sim P_N} \left[ \mathcal{F} \left( \theta_u^T \theta_{i'}, \theta_u^T \theta_i \right) \right] \\
\mathbb{1} \{ \theta_u^T \theta_{i'} - \theta_u^T \theta_i > \epsilon_k \} \log \left[ \sigma \left( \theta_u^T \theta_{i'} - \theta_u^T \theta_i \right) \right]
\]

Figure 1: High-order proximity between users and items within observed interactions

**RecSys’18**

Jheng-Hong Yang, Chih-Ming Chen, Chuan-Ju Wang, Ming-Feng Tsai: HOP-rec: high-order proximity for implicit recommendation. RecSys 2018: 140-144

CLIP Lab, National Chengchi University

CFDA Lab, Academia Sinica
HOP-Rec in **SMORe**

**Graph**

**Sampler**

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

(user-item graph)
HOP-Rec in **SMORe**

**Graph**

- user-item graph

**Sampler**

- greater margin
- smaller margin

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

**Optimizer**

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

**Mapper**

- embedding lookup

---

3. Semantics
We present a general framework, where works.

### 3.1 Homogeneous Network Embedding

To learn the distributed representation of nodes and relations, we use **word2vec** [22], the task is to learn the neighborhood of node $v$, with $p \in V$. Inspired by it, DeepWalk [4], the task is to learn the distributed representation of different types of nodes, facilitating random walk strategy ensures that the pre-defined distribution from which a negative node is sampled is close to the distribution of the positive node. Given a heterogeneous network, meta-paths are commonly used in a symmetric way, that is, its recursive guidance for random walkers, i.e., $v \rightarrow t \rightarrow v$.

In other words, metapath2vec++ is the neighborhood of node $v$, wherein a meta-path "APA" represents the coauthor relationships on a paper network. In doing so, the heterogeneous network structures into skip-gram, we propose to generate paths of multiple types of nodes. At step $t$, for embedding this network.

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

where $|V|$ is the number of nodes, $|T|$ is the number of types of neighbors—venues, coauthors, and so on. Therefore, we have the following objective:

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

wherein $|V|$ is the number of different types, $|T|$ is the number of neighbors in given type.

---

Yuxiao Dong, Nitesh V. Chawla, Ananthram Swami: metapath2vec: Scalable Representation Learning for Heterogeneous Networks. KDD 2017: 135-144
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) 

Translation operation:

prev. item + user ≈ next item

TransRec in **SMORe**

**Graph**

user-item graph

**Sampler**

\[(\text{item}_A, \text{user}, \text{item}_B)\]
TransRec in **SMORe**

**Graph**

- User-item graph

**Sampler**

- \((\text{item}_A, \text{user}, \text{item}_B)\)

**Optimizer**

- Minimize the distance

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

**Mapper**

- Embedding Lookup
Graph Convolution for Recommendation (Pinsage)

Hamilton, Jure Leskovec: Graph Convolutional Neural Networks for Web-Scale Recommender Systems. KDD 2018: 974-983

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)

Simplifying the representation of nodes in a graph allows for more efficient computation. Pinsage leverages this principle by using a graph convolutional neural network (GCN) to learn representations of nodes. The basic idea is that each node aggregates the features of its neighbors, weighted by the strength of the connections between them.

**Algorithm**

1. **Sampler:** Select a target node.
2. **CONVOLVE:** Perform convolution on the sampled neighborhood to update the node's representation.

**Equations**

\[
\begin{align*}
  h^{(s+1)}_A &= \sum_{B \in N(A)} \alpha_{AB} h^{(s)}_B \\
  h^{(s+1)}_B &= \sum_{A \in N(B)} \alpha_{BA} h^{(s)}_A \\
  h^{(s+1)}_C &= \sum_{A \in N(C)} \alpha_{AC} h^{(s)}_A \\
  h^{(s+1)}_D &= \sum_{B \in N(D)} \alpha_{DB} h^{(s)}_B \\
  h^{(s+1)}_E &= \sum_{C \in N(E)} \alpha_{CE} h^{(s)}_C \\
  h^{(s+1)}_F &= \sum_{D \in N(F)} \alpha_{DF} h^{(s)}_D
\end{align*}
\]

**Notes**

- **Node Feature:** Each node in the graph has an associated feature vector.
- **Neighborhood:** The set of nodes directly connected to a given node.
- **Weight:** \(\alpha_{AB}\) represents the weight of the connection between nodes A and B.

**Application**

1. **Bias:** Integrating domain knowledge into the model can improve performance.
2. **Evaluation:** Performance is often evaluated using metrics like hit rate at K (H@K).

**Conclusion**

Pinsage demonstrates significant improvements over traditional GCNs, achieving state-of-the-art performance in graph-based recommendation systems.
Graph Convolution for Recommendation (Pinsage)  

In addition to these fundamental advancements in scalability, we improve the quality of the representations also introduce new training techniques and algorithmic innovations. These innovations are learned by PinSage, leading significant performance gains in downstream recommendation tasks.

We develop a curriculum training scheme, where the algorithm is fed harder-and-harder examples during training, resulting in a 12% performance gain. We design an efficient MapReduce pipeline that can distribute the trained model to generate embeddings for billions of nodes, while minimizing repeated computations.

We have deployed PinSage for a variety of recommendation applications where users interact with online content (e.g., recipes they want to cook, or clothes they like). Through extensive evaluation metrics, controlled user studies, and walk similarity measures, leading to a 46% performance gain in downstream metrics.
<table>
<thead>
<tr>
<th>Model</th>
<th>Sampler</th>
<th>Mapper</th>
<th>Optimizer</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>node2vec</strong></td>
<td>Community (Weight)</td>
<td>Vertex Embedding</td>
<td>Sigmoid Dot Product</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Context Embedding</td>
<td>Log Likelihood</td>
</tr>
<tr>
<td><strong>HOP-Rec</strong></td>
<td>Community (Closeness)</td>
<td>Vertex Embedding</td>
<td>Dot Product</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>BPR</td>
</tr>
<tr>
<td><strong>metapath2vec</strong></td>
<td>Community (Semantics)</td>
<td>Vertex Embedding</td>
<td>Sigmoid Dot Product</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Context Embedding</td>
<td>Log Likelihood</td>
</tr>
<tr>
<td><strong>TransRec</strong></td>
<td>Adjacency (Inductiveness)</td>
<td>Vertex Embedding</td>
<td>Dot product</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>BPR</td>
</tr>
<tr>
<td><strong>Pinsage</strong></td>
<td>Adjacency (Inductiveness)</td>
<td>Vertex / Context Embedding Pooling + Concatenation</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

Collaborative filtering (CF) aims to alleviate data sparsity in single-domain CF by leveraging knowledge transferred from related source domains. Cross-domain collaborative filtering (C-D-CF) is proposed to enhance recommender systems in cross-domain settings, where users and items are shared in the target domain but not in the source domain.

A vital issue in C-D-CF is the data sparsity problem. In this paper, we propose an alternative explicit relation-enrichment procedure to improve recommendation performance in both target and source domains. The superhighway construction is established to bypass data sparsity in C-D-CF.

Superhighway Construction

- In the source domain, users and items are represented by nodes $U_S$ and $I_S$, respectively.
- In the target domain, users and items are represented by nodes $U_T$ and $I_T$, respectively.
- The superhighway construction is represented by a function $\mathcal{F}$, connecting the source domain and the target domain.

Figure 1: Illustrative example for superhighways

\( \mathcal{F} \) is a function that maps the source domain to the target domain, establishing direct relations between candidate users and items. The function $\mathcal{F}$ is defined as a path between $U_S$ and $U_T$, denoted by $\tilde{\mathcal{F}}$.

The superhighway construction bypasses multi-hop inter-domain paths between cross-domain users and target domain items by leveraging knowledge transferred from related source domains.
Superhighway in SMORe
• Multi-Task Learning
  - shared representations for multi-tasks
2. cluster users

Neighborhood Similarity Embedding (NSEmbed)

\[ \Phi_{UC} \]

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

Direct Similarity Embedding (DSEmbed)

\[ \Phi \]

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

3. cluster items

Chih-Ming Chen, Chuan-Ju Wang, Ming-Feng Tsai, Yi-Hsuan Yang: Collaborative Similarity Embedding for Recommender Systems. WWW 2019: 2637-2643
CSE in **SMORe**

Graph

**Sampler1**
((user, item))

**Sampler2**
((user, u_community))

**Sampler3**
((item, i_community))
CSE in **SMORe**

Graph

User-item graph

Mapper 1

- Vertex Embedding Lookup
  - $\overrightarrow{V_1} \leftarrow \overrightarrow{V_1}$

Mapper 2

- Context Embedding Lookup
  - $\overrightarrow{V_2} \leftarrow \overrightarrow{V_2}$

Mapper 3

- Context Embedding Lookup
  - $\overrightarrow{V_3} \leftarrow \overrightarrow{V_3}$

User-item preference

$M_1$

Cluster users

$M_2$

Cluster items

$M_3$
CSE in **SMORe**

**Optimizer 1**

\[ \log \sigma(\tilde{v}_1 \cdot \tilde{v}_1) \]

Log Likelihood

**Optimizer 2**

\[ \log \sigma(\tilde{v}_1 \cdot \tilde{v}_2) \]

Log Likelihood

**Optimizer 3**

\[ \log \sigma(\tilde{v}_1 \cdot \tilde{v}_3) \]

Log Likelihood

---

**M₁** user-item preference

**M₂**
cluster users

**M₃**
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

1. clone it
2. enter it
3. compile it
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 & nerrank 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)

```plaintext
userA itemA 3
userA itemC 5
userB itemA 1
userB itemB 5
userC itemA 4
```

Embeddings as output

```plaintext
6 5
userA 0.0815412 0.0205459 0.288714 0.296497 0.394043
itemA -0.287083 -0.258583 0.233185 0.0959801 0.258183
itemC 0.0185886 0.138003 0.213669 0.276383 0.45732
userB -0.0137994 -0.227462 0.103224 -0.456051 0.389858
itemB -0.317921 -0.163652 0.103891 -0.449869 0.318225
userC -0.156576 -0.3505 0.213454 0.10476 0.259673
```
• On-Going Work
SMORe (another modularized version)

find the branch smore

it's under refactoring ...
SMORe for Developers

Graph

Sampler() → entity

Mapper() → embedding

Optimizer() → loss

Embeddings for Recommendations

for Recommendations
SMORe Example Codes (in smore branch)

```c
46 // main
47 // 0. [Graph] read from file-based graph
48 FileGraph *file_graph = new FileGraph(path, 0);
49
50 // 1. [Sampler] determine what sampler to be used
51 VCSampler sampler(file_graph);
52
53 // 2. [Mapper] define what embedding mapper to be used
54 LookupMapper mapper(sampler.vertex_size, dimension);
55
56 // 3. [Optimizer] claim the optimizer
57 PairwiseOptimizer optimizer;
```

Graph()
Sampler()
Mapper()
Optimizer()
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);
```
**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);
```
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);
}
```
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);
    }
    mapper.update_with_l2(user, user_batch_loss, alpha, 0.01);
}
```
**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);
```

Sample positive (user, item) pair

Map user/item to its embedding

Estimate the loss from log likelihood

Run it until conditions hold
**SMORe** Factorization (in smore branch)

```
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);
    }
    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)

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();
    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);
    }
    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

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

Multi-threading

Practice

HOGWILD!

https://www.reddit.com/r/aww/comments/2oaqj8/multithreaded_programming_theory_and_practice/


Embeddings

[0.08 0.02 0.28]

[-0.31 -0.1 0.1]

[-0.15 -0.3 0.2]
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 Re:

SMORE Members at CFDA & CLIP Labs: [https://cfda.csie.org/](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
mftsa@nccu.edu.tw

Special Thanks to:

Link to Codes, Slides, Abstract:
[https://github.com/cnclabs/smore/](https://github.com/cnclabs/smore/)