The Henry and Marilyn Taub
Faculty of Computer Science

Module 3: Node and Tuple Embedding

StructML Course

Benny Kimelfeld
Technion – Israel Institute of Technology

Spring 2026

Module Goals

- Understand the concept of *embedding* (*representation learning*) and its role in (structured) ML
- Study the *encoder-decoder* paradigm to embedding
- Learn important *instantiations* of the encoder-decoder paradigm: *random walks* and *matrix decomposition*
- Learn how embedding techniques are adapted to *heterogeneous* graphs and *relational databases*

Outline

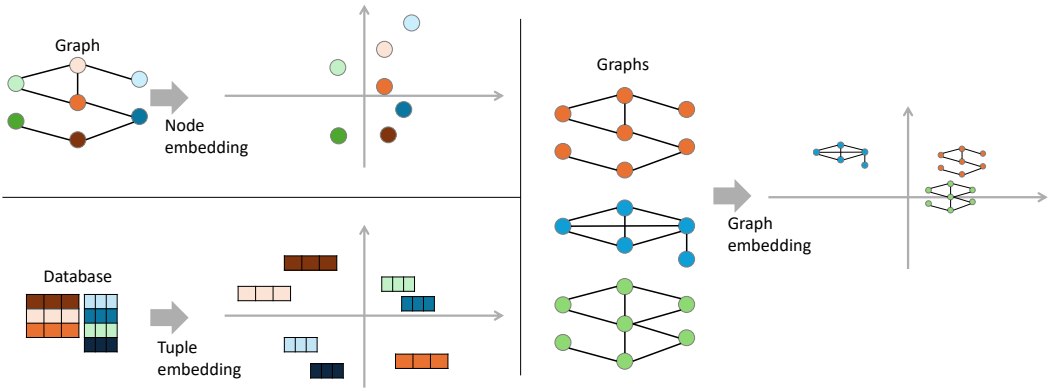
Representation Learning via Embedding

- ML models often operate over numerical feature vectors of their input entities
- When we learn over non-numerical data (e.g., text, graph, databases), we need to translate our data into a *numerical representation*
- The process of coming up with a suitable numerical representation is called *representation learning*
- Equivalently, we can think of representation learning as *embedding* our data in a specific vector space $\mathbb{R}^d$
- Conventionally, "embedding" refers to learning a **general-purpose representation** that does not depend on any downstream task
  - But not necessarily (discussed later)

What to Embed?

- In the context of *graphs*, we can think of several types of entities to embed:
  - Nodes (most common)
  - Edges (rare)
  - Whole graphs (for cases of many small graphs, e.g., molecules)
- In the context of *relational databases*:
  - Tuples (most common)
  - Cells (less common)
  - Schema items: relation names, attributes (used in *data integration* for aligning different schemas)
- Analogy in text: *token*, *word*, *sentence*, *document* embedding
- We will focus mainly on **node embedding** and **tuple embedding**, and discuss more briefly other entities like whole graphs, table cells, and attribute names

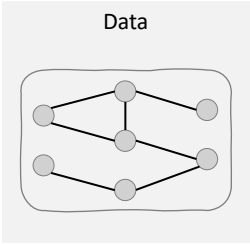
Node Embedding, Tuple Embedding, Graph Embedding (2d)



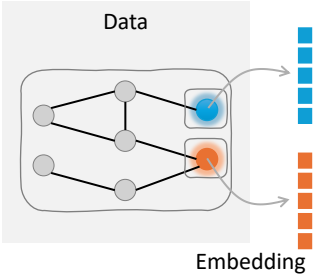
Task-Specific vs. Task-Agnostic

- ML uses representation of structured data in different ways:
  - Task-specific representation: Optimized directly for a downstream task at hand
    - Usually, *supervised* learning by task-provided training examples
    - **Evaluation**: compute both the representation and the upper-layer model
  - Task-agnostic embedding: Learned independently of any specific prediction task
    - Typically *unsupervised* or *self-supervised* learning
    - Should be good for all future tasks
    - **Evaluation**: use fixed embedding, apply learned model
    - **Quality measurement**: test on *a variety of downstream tasks* ; also, intuitive visualization in 2d (e.g., *t-SNE*)
  - Fine-tuning: Adapting a *task-agnostic* embedding to a *task-specific* one
    - **Evaluation**: start with the precomputed embedding, adapt using the model
- A good question about every model on structured data is whether the representations are *fixed*, *learned*, or *fine-tuned*
- We will focus on *task-agnostic embedding*

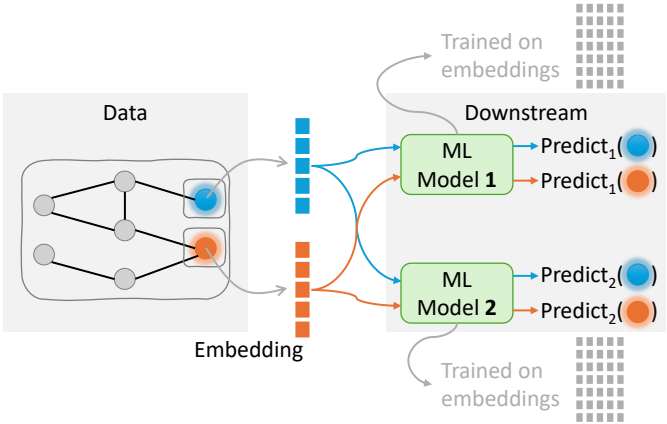
Example: Task-Agnostic Node Embedding



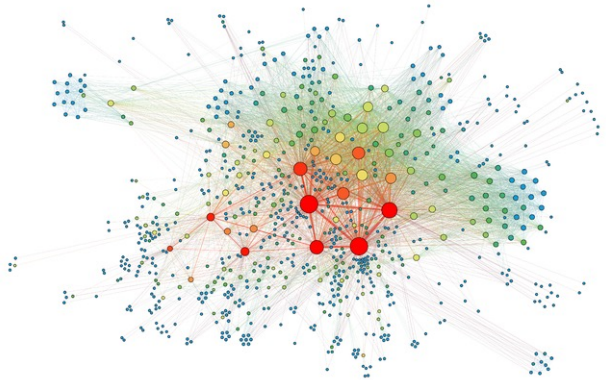
Example: Task-Agnostic Node Embedding



Example: Task-Agnostic Node Embedding



Embedding Visualization

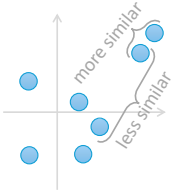


(Image from https://towardsdatascience.com/)

Graph $\rightarrow$ embedding $\rightarrow$ dimensionality reduction to 2d (typically t-SNE)

What Makes an Embedding Good?

- To be useful, an embedding should encode the information relevant to all downstream tasks
- Of course, with a sophisticated encoding, we can literally encode the entire structure in a single number; but is it useful?
- We seek an embedding that is:
  - Informative
  - Of a dimension of choice, *low-dimension* if needed
  - Useful to practical ML models, robust to nuances
- Guiding principle: geometric proximity reflects semantic proximity



If two things mean alike, they should lie alike
— and if they lie alike, they should mean alike

Example: CS Conferences

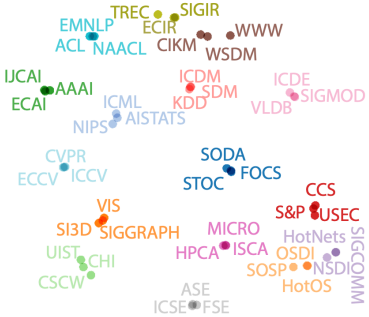


Figure 5: 2D t-SNE projections of the 128D embeddings of 48 CS venues, three each from 16 sub-fields.

[Dong et al., 2017]

How to Obtain Good Embeddings?

- Rich research around this question
- One way is to extract many features, as we saw in the previous lecture (e.g., many graphlets), and:
  - Apply **feature selection** to reduce the number of features
    - E.g., *filter methods* that are based on analysis of feature statistics (correlation), *wrapper methods* find the features most useful to a task
  - Apply some generic form of **dimensionality reduction**, e.g., PCA
- However, these techniques are sensitive to the selected features, expensive to compute (due to the many features), and lack the rationale for semantic proximity
- The common way is to compute from scratch an embedding that *best captures* some simple and natural interpretation of proximity—**the focus of this module**

Outline

Node Embedding: Setting

- Given $G = (V, E)$ with the adjacency matrix $\mathbf{A}$, compute an embedding $\mathbf{Z} \in \mathbb{R}^{|V| \times k}$, having a row (representation) $\mathbf{z}_v$ for every node v
- Note: we consider graphs without node features
  - That is, we consider the problem of embedding a node only based on the graph structure, ignoring any predefined embedding

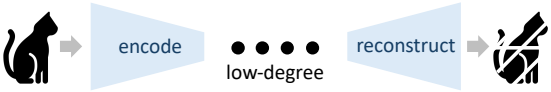
Usage Examples of Node Embedding

- Features for learning node-level models
  - Predict$(v) = F(\mathbf{z}_v)$
- Features for *link-prediction* tasks
  - PredictLink$(u, v) = F(\mathbf{z}_u, \mathbf{z}_v)$
- Node *clustering*
  - Via vector clustering techniques
- Graph *visualization*
  - E.g., via t-SNE
- *Whole-graph* embeddings
  - E.g., $\mathbf{z}_G = \sum_{v \in V} \mathbf{z}_v$ used for molecule embedding [Duvenaud et al., 2015]

Embedding via Self-Supervision

- Common techniques for node embedding use *self-supervision*
- **Self-supervised learning**: Learn from examples taken *from the data itself*
  - By defining a task whose targets are automatically derived from the inputs
    - Referred to as a "pretext task"
  - Learn a parameterized model by optimizing it on this pretext task
  - Examples of pretext tasks: predicting *masked* or *transformed* parts, or *reconstructing the input* (autoencoders)

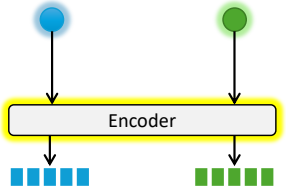
Idea of masking



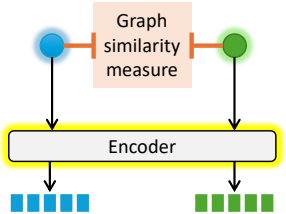
Idea of autoencoding

Outline

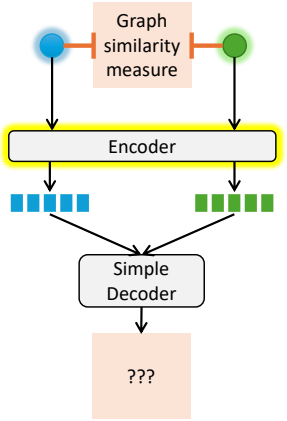
General Encoder-Decoder Scheme for Node Embedding



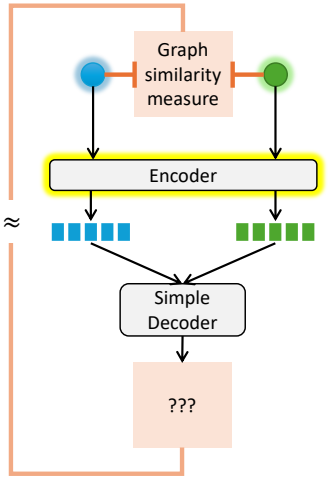
General Encoder-Decoder Scheme for Node Embedding



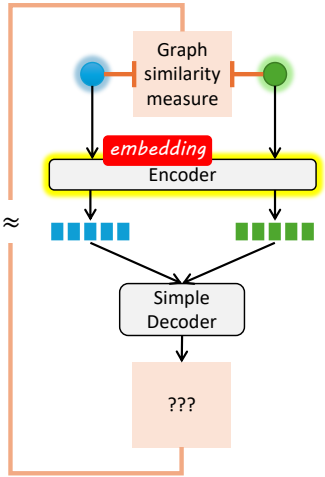
General Encoder-Decoder Scheme for Node Embedding



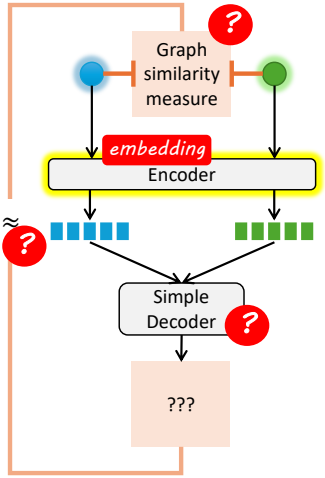
General Encoder-Decoder Scheme for Node Embedding



General Encoder-Decoder Scheme for Node Embedding



General Encoder-Decoder Scheme for Node Embedding



The Encoder-Decoder Paradigm

- Our goal is to learn an embedding $\mathbf{Z} : V \to \mathbb{R}^k$ of dimension k
- We define 3 components:
  - A **proximity function** $\mathcal{P} : V \times V \to \mathbb{R}$
  - A **decoder function** $\mathcal{D} : \mathbb{R}^k \times \mathbb{R}^k \to \mathbb{R}$ that aims to approximate $\mathcal{P}$
  - A **loss function** $\mathcal{L} : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ that assigns to $\mathbf{Z}$ a cost (error penalty)
- This gives a *cost function*:

$$\text{Cost}(\mathbf{Z}) := \sum_{u,v \in V} \mathcal{L}\left(\mathcal{P}(u,v), \mathcal{D}(\mathbf{z}_u, \mathbf{z}_v)\right)$$

- Goal: find a minimum-cost $\mathbf{Z}$

Popular Instantiations

From Hamilton's *Graph Representation Learning* book:

Decoder $\mathcal{D}$ | Proximity $\mathcal{P}$ | Loss $\mathcal{L}$

Method	Decoder	Similarity measure	Loss function
Lap. Eigenmaps	$\|\mathbf{z}_u - \mathbf{z}_v\|_2^2$	general	$\text{DEC}(\mathbf{z}_u, \mathbf{z}_v) \cdot \mathbf{S}[u, v]$
Graph Fact.	$\mathbf{z}_u^\top \mathbf{z}_v$	$\mathbf{A}[u, v]$	$\|\text{DEC}(\mathbf{z}_u, \mathbf{z}_v) - \mathbf{S}[u, v]\|_2^2$
GraRep	$\mathbf{z}_u^\top \mathbf{z}_v$	$\mathbf{A}[u, v], ..., \mathbf{A}^k[u, v]$	$\|\text{DEC}(\mathbf{z}_u, \mathbf{z}_v) - \mathbf{S}[u, v]\|_2^2$
HOPE	$\mathbf{z}_u^\top \mathbf{z}_v$	general	$\|\text{DEC}(\mathbf{z}_u, \mathbf{z}_v) - \mathbf{S}[u, v]\|_2^2$
DeepWalk	$\frac{e^{\mathbf{z}_u^\top \mathbf{z}_v}}{\sum_{k \in \mathcal{V}} e^{\mathbf{z}_u^\top \mathbf{z}_k}}$	$p_{\mathcal{G}}(v\|u)$	$-\mathbf{S}[u, v] \log(\text{DEC}(\mathbf{z}_u, \mathbf{z}_v))$
node2vec	$\frac{e^{\mathbf{z}_u^\top \mathbf{z}_v}}{\sum_{k \in \mathcal{V}} e^{\mathbf{z}_u^\top \mathbf{z}_k}}$	$p_{\mathcal{G}}(v\|u)$ (biased)	$-\mathbf{S}[u, v] \log(\text{DEC}(\mathbf{z}_u, \mathbf{z}_v))$

[Hamilton, 2020]

Severe Scale Issue

$$\text{Cost}(\mathbf{Z}) := \sum_{u,v \in V} \mathcal{L}\left(\mathcal{P}(u,v), \mathcal{D}(\mathbf{z}_u, \mathbf{z}_v)\right)$$

1. cost($\mathbf{Z}$) requires *every* pair of nodes
2. $\mathcal{D}(\mathbf{z}_u, \mathbf{z}_v)$ often normalizes w.r.t. *every* possible u', or every possible v'
   - For every instance/batch during SGD!

Becoming infeasible very quickly

Solution: Negative Sampling

$$\text{Cost}(\mathbf{Z}) := \sum_{u,v \in V} \mathcal{L}\left(\mathcal{P}(u,v), \mathcal{D}(\mathbf{z}_u, \mathbf{z}_v)\right)$$

- As we will see later, it is often the case that $\mathcal{P}$ is nonzero for just a small subset $\text{Pairs}_{\text{pos}}$ of $V \times V$ (e.g., 2-hop neighbors), called *positive pairs*
- *Negative sampling* (originated at Google's word2vec) samples a set $\text{Pairs}_{\text{neg}}$ of *negative pairs* and use only them instead of the entire complement of $\text{Pairs}_{\text{pos}}$
- The distribution may either be uniform or be biased, e.g., by node degrees

Using Negative Samples

$$\text{Cost}(\mathbf{Z}) := \sum_{u,v \in V} \mathcal{L}\left(\mathcal{P}(u,v), \mathcal{D}(\mathbf{z}_u, \mathbf{z}_v)\right)$$

- For $\text{Cost}(\mathbf{Z})$, this is typically an i.i.d. sampled set of pairs
- Hence, $\text{Cost}(\mathbf{Z})$ becomes:

$$\sum_{(u,v) \in \text{Pairs}_{\text{pos}}} \mathcal{L}\left(\mathcal{P}(u,v), \mathcal{D}(\mathbf{z}_u, \mathbf{z}_v)\right) + \lambda \cdot \sum_{(u,v) \in \text{Pairs}_{\text{neg}}} \mathcal{L}\left(0, \mathcal{D}(\mathbf{z}_u, \mathbf{z}_v)\right)$$

- For the normalization of $\mathcal{D}(\mathbf{z}_u, \mathbf{z}_v)$, this is typically an i.i.d. sampled set of:
  - Pairs (u, v')—*rightward* negative sampling; *or*
  - Pairs (u', v)—*leftward* negative sampling

Example 1: Graph Factorization

- [Ahmed et al., 2013]
- Work from Google, proposes a basic instantiation of the encoder-decoder paradigm, focuses on scaling via *distributed computing*
  - Demonstration on a graph with 200 million vertices and 10 billion edges
- Instantiation:
  - Proximity function: $\mathcal{P}(u, v) := \mathbf{A}_{i,j} = \begin{cases} 1 & \text{if } \{u, v\} \in E \\ 0 & \text{otherwise} \end{cases}$
  - Decoder function: $\mathcal{D}(\mathbf{z}_u, \mathbf{z}_v) := \mathbf{z}_u \cdot \mathbf{z}_v$
  - Loss function: $\mathcal{L}\big(\mathcal{P}(u, v), \mathcal{D}(\mathbf{z}_u, \mathbf{z}_v)\big) := \big(\mathcal{P}(u, v) - \mathcal{D}(\mathbf{z}_u, \mathbf{z}_v)\big)^2$
    - In addition, cost($\mathbf{Z}$) includes a regularization term $\sum_v \|\mathbf{z}_v\|_2^2$

Example 2: Large-scale Information Network Embeddings (LINE)

- [Tang et al., 2015]
- Idea: rather direct instantiation of the paradigm, considering both *first-order* and *second-order* node proximity
- Assumes every edge $\{u, v\}$ has a weight $w_{u,v} \geq 0$
  - Higher weight means stronger association (can be simply a fixed number)
  - We assume that it is normalized as a distribution $\sum w_{u,v} = 1$
- Proximity: $\mathcal{P}(u, v) = w_{u,v}$
- Loss function: *KL-divergence* between w and $\mathcal{D}$ (omitting a constant factor):

$$\mathcal{L}\left(\mathcal{P}(u, v), \mathcal{D}(\mathbf{z}_u, \mathbf{z}_v)\right) := -w_{u,v} \log \mathcal{D}(\mathbf{z}_u, \mathbf{z}_v)$$

- Decode: Two decoding options $\mathcal{D}$ proposed:

Decoding Options of LINE

- **1st order**: Sigmoid of the inner product:

$$\mathcal{D}_1(\mathbf{z}_u, \mathbf{z}_v) := \text{sigmoid}(\mathbf{z}_u \cdot \mathbf{z}_v) := \frac{1}{1 + \exp^{-\mathbf{z}_u \cdot \mathbf{z}_v}}$$

- **2nd order**: Conditional probability of u as source given the target v:

$$\mathcal{D}_2(\mathbf{z}_u, \mathbf{z}_v) := \frac{\exp(\mathbf{z}_u \cdot \mathbf{z}_v)}{\sum_{u'} \exp(\mathbf{z}_{u'} \cdot \mathbf{z}_v)}$$

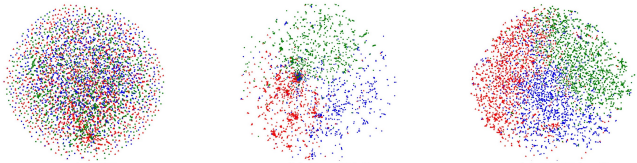
- With both, LINE uses a specialized improvement of SGD for loss optimization
  - Along with (leftward) negative sampling

Experiments from the LINE Paper

Graph factorization

Wikipedia page classification

Metric	Algorithm	10%	20%	30%	40%	50%	60%	70%	80%	90%
Micro-F1	GF	79.63	80.51	80.94	81.18	81.38	81.54	81.63	81.71	81.78
	DeepWalk	78.89	79.92	80.41	80.69	80.92	81.08	81.21	81.35	81.42
	SkipGram	79.84	80.82	81.28	81.57	81.71	81.87	81.98	82.05	82.09
	LINE-SGD(1st)	76.03	77.05	77.57	77.85	78.08	78.25	78.39	78.44	78.49
	LINE-SGD(2nd)	74.68	76.53	77.54	78.18	78.63	78.96	79.19	79.40	79.57
	LINE(1st)	79.67	80.55	80.94	81.24	81.40	81.52	81.61	81.69	81.67
	LINE(2nd)	79.93	80.90	81.31	81.63	81.80	81.91	82.00	82.11	82.17
	LINE(1st+2nd)	**81.04****	**82.08****	**82.58****	**82.93****	**83.16****	**83.37****	**83.52****	**83.63****	**83.74****



(a) GF (b) DeepWalk (c) LINE(2nd)

Figure 2: Visualization of the co-author network. The authors are mapped to the 2-D space using the t-SNE package with learned embeddings as input. Color of a node indicates the community of the author. Red: "data Mining," blue: "machine learning," green: "computer vision."

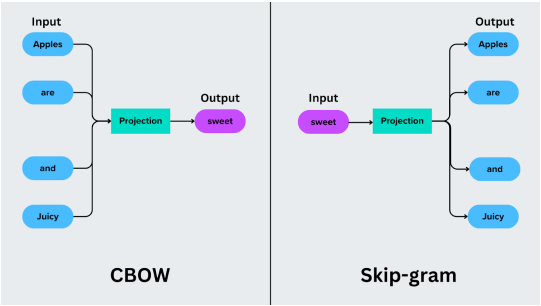
[Tang et al., 2015]

Outline

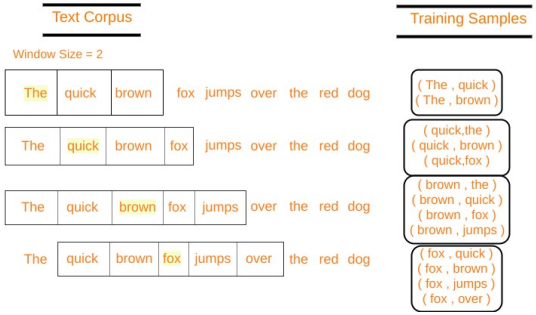
Background: word2vec

- Using random walks for node embedding has emerged at a time where *word embeddings* was a developed concept in Natural Language Processing (NLP) via Google's *word2vec*
- Such word embeddings were trained on a corpus of text documents:
  - Considering *sliding windows* of text as contexts
  - Optimizing the embeddings to allow for predicting masked words inside windows
  - Two main variants:
    - In **CBOW** (Continuous Bag of Words), we guess a masked word from the context
    - In **skip-gram**, we guess the context words from the center word
  - Introduced the usage of *negative sampling*

Illustration of Prediction Variants



https://zilliz.com/glossary/word2vec



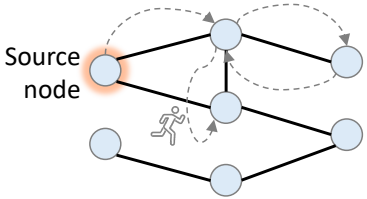
https://www.geeksforgeeks.org/python/implement-your-own-word2vecskip-gram-model-in-python/

The skip-gram model in a sliding window

Why does word2vec make sense? What is the intuition?

Random Walk

A *random walk* is the stochastic process of visiting nodes by repeatedly moving from a node to a random neighbor

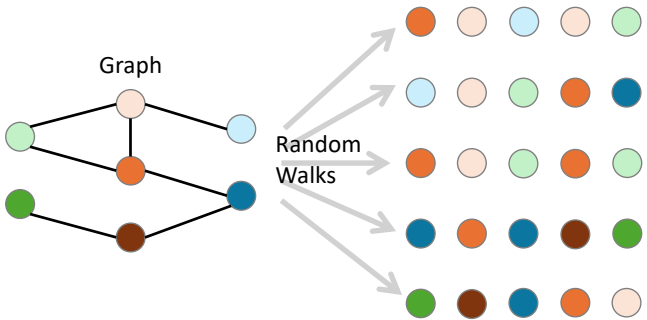


From Text to Graphs

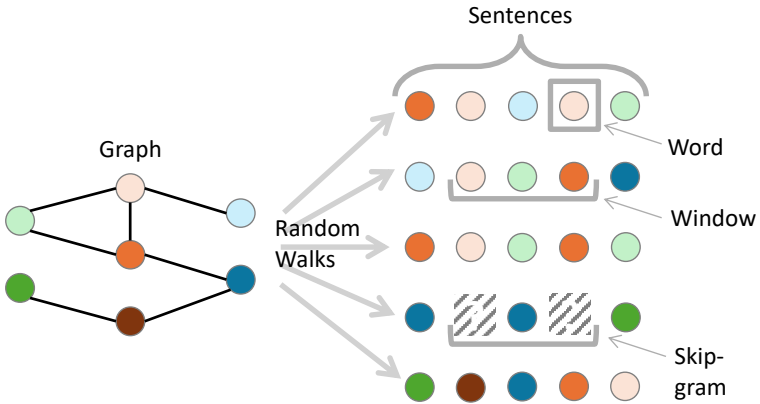
The adaptation from text (words) to graphs (nodes) is as follows:

- Treat the node set as the dictionary of *words*
- Sample from the graph a collection C of *random walks*; treat as sentences
- Learn word embeddings from C
- Skip-gram is the more popular alternative

Adaptation of word2vec Skip-Gram



Adaptation of word2vec Skip-Gram



In a random-walk approach, proximity $\mathcal{P}(u, v)$ is a *probability distribution* $\Pr(u \mid \mathbf{z}_v)$

- A common approach to learning such distributions is to learn a numerical function $\mathbb{R}^k \rightarrow \mathbb{R}^n$ and normalize
- What is the computational challenge here?
- How can we solve it?

DeepWalk [Perozzi et al., 2014]

- The proximity function is $\mathcal{P}(u, v) \in [0, 1]$ — the empirical probability of visiting u in a random path that includes v
- The decoding function $\mathcal{D}(\mathbf{z}_u, \mathbf{z}_v) \in [0, 1]$
  - Idea: use $\mathbf{Z}$ as the parameters of a learned regression model that targets $\Pr(u \mid \mathbf{z}_v)$
  - To this aim, they deploy *Hierarchical Softmax* [Mnih and Hinton, 2008]
    - Organize all nodes u on a binary tree, learn $\log(|V|)$ functions $\Pr(b_u^i \mid \mathbf{z}_v)$ — one for each step $b_u^i \in \{0, 1\}$ on the path to u, approximate $\Pr(u \mid \mathbf{z}_v)$ as $\prod_i \Pr(b_u^i \mid \mathbf{z}_v)$
  - As a result, it encourages nodes in the same neighborhood to have similar classifiers, hence similar coefficients, hence nearby embeddings
- Learning technique: go over the walks with v in some random order; whenever a node u is visited, apply SGD update to $\mathcal{D}(\mathbf{z}_u, \mathbf{z}_v)$

DeepWalk Algorithm

Algorithm 1 DeepWalk(G, w, d, γ, t)

Input: graph $G(V, E)$
window size w
embedding size d
walks per vertex γ
walk length t
Output: matrix of vertex representations $\Phi \in \mathbb{R}^{|V| \times d}$

Initialization: Sample Φ from $\mathcal{U}^{|V| \times d}$
Build a binary Tree T from V
for $i = 0$ to γ **do**
  $\mathcal{O} = \text{Shuffle}(V)$
  for each $v_i \in \mathcal{O}$ **do**
    $\mathcal{W}_{v_i} = RandomWalk(G, v_i, \text{t})$
    SkipGram(Φ, $\mathcal{W}_{v_i}$, w)
  end for
end for

Algorithm 2 SkipGram(Φ, $\mathcal{W}_{v_i}$, w)

for each $v_j \in \mathcal{W}_{v_i}$ **do**
  for each $u_k \in \mathcal{W}_{v_i}[j - w : j + w]$ **do**
    $J(\Phi) = -\log \Pr(u_k \mid \Phi(v_j))$
    $\Phi = \Phi - \alpha * \frac{\partial J}{\partial \Phi}$
  end for
end for

[Perozzi et al., 2014]

DeepWalk Algorithm

Algorithm 1 DeepWalk(G, w, d, γ, t)

Input: graph $G(V, E)$
window size w
embedding size d
walks per vertex γ
walk length t
Output: matrix of vertex representations $\Phi \in \mathbb{R}^{|V| \times d}$

1: Initialization: Sample Φ from $\mathcal{U}^{|V| \times d}$
2: Build a binary Tree T from V
3: **for** $i = 0$ to γ **do**
4: $\mathcal{O} = \text{Shuffle}(V)$
5: **for each** $v_i \in \mathcal{O}$ **do**
6: $\mathcal{W}_{v_i} = \mathit{RandomWalk}(G, v_i, t)$
7: SkipGram(Φ, $\mathcal{W}_{v_i}$, w)
8: **end for**
9: **end for**

Binary rep of nodes

Sample a random walk

Algorithm 2 SkipGram(Φ, $\mathcal{W}_{v_i}$, w)

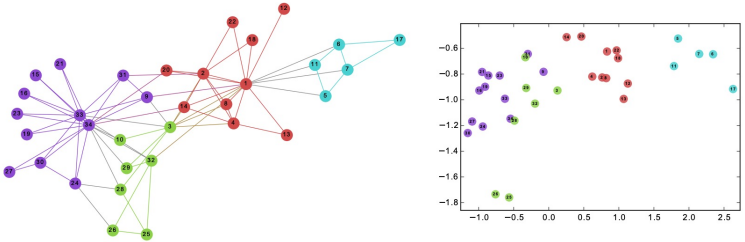
1: **for each** $v_j \in \mathcal{W}_{v_i}$ **do**
2: **for each** $u_k \in \mathcal{W}_{v_i}[j - w : j + w]$ **do**
3: $J(\Phi) = -\log \Pr(u_k \mid \Phi(v_j))$
4: $\Phi = \Phi - \alpha * \frac{\partial J}{\partial \Phi}$
5: **end for**
6: **end for**

Window around v_i

SGD step

[Perozzi et al., 2014]

DeepWalk Case Study (2d on a Karate Network)



(a) Input: Karate Graph (b) Output: Representation

[Perozzi et al., 2014]

Different colors correspond to different predefined communities

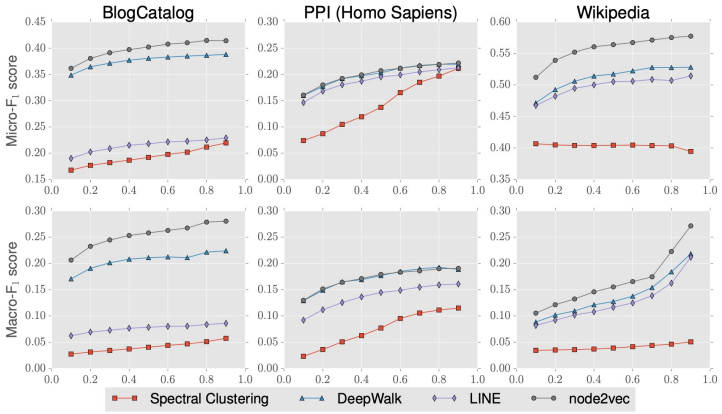
node2vec

- The *node2vec* algorithm [Grover and Leskovec, 2016] is similar to DeepWalk, with two main differences:
  1. The decoding $\mathcal{D}(\mathbf{z}_u, \mathbf{z}_v)$ is simpler—a normalized dot product:

$$\mathcal{D}(\mathbf{z}_u, \mathbf{z}_v) = \frac{\exp(\mathbf{z}_u \cdot \mathbf{z}_v)}{\sum_{w \in V} \exp(\mathbf{z}_u \cdot \mathbf{z}_w)}$$

     - Uses rightward negative sampling for the denominator
  2. It uses a flexible next-step strategy for the random walks, using probabilities:
     - q_1 of *going back* in the edge
     - q_2 of *staying* in place
     - q_3 for moving to a *new node*
  3. Importantly, these are hyperparameters optimized based on a portion (10%) of the training data (consider multiple combinations and take the best)

Experiments from the node2vec Paper



[Grover and Leskovec, 2016]

- Multi-class classification
- Logistic regression classifier over the embedding
- The x-axis corresponds to the fraction of labeled data, y-axis is the F1-score

Extension to Whole-Graph Embedding

- The analogy between *node embedding* and *word embedding* has also been extended to whole-graph embedding, similarly to the embedding of whole documents
  - For documents: *doc2vec*
  - For graphs: *graph2vec* [Narayanan et al., 2017]
- **Idea:**
  - Learn node embeddings $\mathbf{z}_v$ and graph embeddings $\mathbf{z}_G$ in parallel
  - For each node v, list nodes in the neighborhood of v
  - Optimize a (simplistic) probability model for a neighborhood, given $\mathbf{z}_G$:

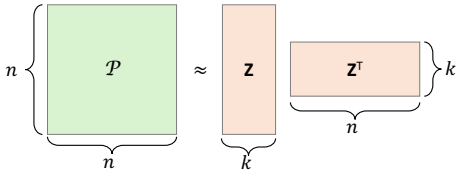
$$\Pr(N) := \frac{\prod_{v \in N} \mathbf{z}_G \cdot \mathbf{z}_v}{\sum_{N'} \prod_{v \in N'} \mathbf{z}_G \cdot \mathbf{z}_v}$$

  - As usual, use *negative sampling* for the denominator

Outline

Node Representation via Matrix Factorization

- Assume that our decoder $\mathcal{D}$ is the dot product
- The proximity $\mathcal{P}$ can be viewed as a matrix in $\mathbb{R}^{|V|\times|V|}$
- Our goal is then to find a matrix $\mathbf{Z} \in \mathbb{R}^{|V|\times k}$ such that $\mathcal{P} \approx \mathbf{Z} \cdot \mathbf{Z}^\top$
  - That is, $\mathcal{P}$ factors approximately as $\mathbf{Z}$ multiplied by itself (transformed)



- Traditional theory of *matrix factorization* gives an optimal solution, via **SVD**

Singular Value Decomposition (SVD)

Every matrix $\mathbf{A} \in \mathbb{R}^{n \times m}$ can be written as $\mathbf{A} = \mathbf{U}\,\mathbf{\Sigma}\,\mathbf{W}^\top$ where:

- $\mathbf{U} \in \mathbb{R}^{n \times n}$ and $\mathbf{W} \in \mathbb{R}^{m \times m}$: are orthonormal matrices
  - The columns of $\mathbf{U}$ and the columns of $\mathbf{W}$ are called the *left-singular vectors* and *right-singular vectors* of $\mathbf{A}$, respectively
- $\mathbf{\Sigma} \in \mathbb{R}^{n \times m}$: rectangular diagonal with the *singular values* $\sigma_1 \geq \cdots \geq \sigma_r \geq 0$ of $\mathbf{A}$
  - Recall: a *rectangular diagonal matrix* is an $(n \times m)$-matrix with all entries being zero except for the (i, i) entries
  - Recall: a *singular value* of $\mathbf{A}$ is the square root $\sqrt{\lambda}$ of an eigenvalue λ of the square matrix $\mathbf{A}^\top\mathbf{A}$
  - All such λs are real and nonnegative, since $\mathbf{A}^\top\mathbf{A}$ is symmetric, positive semidefinite

The decomposition $\mathbf{A} = \mathbf{U}\,\mathbf{\Sigma}\,\mathbf{W}^\top$ is called an *SVD of* $\mathbf{A}$

Truncated SVD

- Let $\mathbf{A} = \mathbf{U}\,\mathbf{\Sigma}\,\mathbf{W}^\top$ be an SVD of $\mathbf{A}$
- Recall: $\mathbf{\Sigma}$ has a diagonal with the order $\sigma_1 \geq \cdots \geq \sigma_r \geq 0$ of singular values
- For $k \leq m, n$, define the k-truncated SVD of $\mathbf{A}$ as the result of using the k *dominant singular directions*: $\mathbf{A}_k = \mathbf{U}_k\,\mathbf{\Sigma}_k\,\mathbf{W}_k^\top$
  - $\mathbf{U}_k \in \mathbb{R}^{n \times k}$ contains the first k left-singular vectors (columns of $\mathbf{U}$)
  - $\mathbf{\Sigma}_k \in \mathbb{R}^{k \times k}$ is diagonal with the top k (highest) singular values
  - $\mathbf{W}_k \in \mathbb{R}^{m \times k}$ contains the first k right-singular vectors (columns of $\mathbf{W}$)

Low-Rank Approximation via Truncated SVD

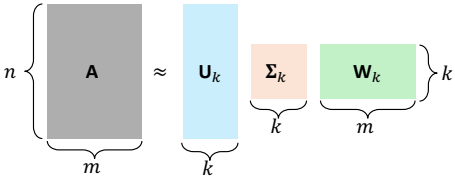
Theorem (Eckart–Young–Mirsky)

$\mathbf{A}_k$ is an optimal rank-k approximation; that is, $\|\mathbf{A} - \mathbf{A}_k\|$ is minimal among all $\|\mathbf{A} - \mathbf{B}\|$ where rank$(\mathbf{B}) \leq k$.

$\|\cdot\|$ refers to the *Frobenius* norm: ℓ_2 seeing $\mathbf{A}$ and $\mathbf{B}$ as vectors:

$$\|\mathbf{M}\|_F = \sqrt{\sum_{i,j} \mathbf{M}_{ij}^2}$$

Interpretation of SVD



- Latent topics: $\mathbf{A}$ represents a relationship between a set C (size n) and a set D (size m) — $\mathbf{A}_{i,j}$ states how c_i relates to d_j
  - SVD *learns* this relationship in terms of a small number of *topics*: What c_i seeks vs. what d_j offers: $\mathbf{A}_{i,j} \approx \mathbf{u}_i \cdot \mathbf{\Sigma}_k \cdot \mathbf{w}_j$
  - Proved highly successful in the *Netflix Prize* challenge (users rate movies)
- Low-rank compression:
  - Semantics-preserving compressed representation (similarly to embedding)
  - For symmetric matrices we factorize $\mathbf{A} \approx \mathbf{F}^\top \mathbf{F}$

SVD for Representation Learning

- When $\mathcal{P}$ is symmetric (viewed as a symmetric square matrix), the decomposition $\mathcal{P} = \mathbf{U}\,\mathbf{\Sigma}\,\mathbf{W}^\top$ has a special structure, namely $\mathbf{U} = \mathbf{W}$
- Hence, $\mathcal{P} = \mathbf{U}\,\mathbf{\Sigma}\,\mathbf{U}^\top = (\mathbf{U}\sqrt{\mathbf{\Sigma}})(\mathbf{U}\sqrt{\mathbf{\Sigma}})^\top$
  - Recall that $\mathbf{\Sigma}$ is diagonal, hence $\sqrt{\mathbf{\Sigma}}$ is also diagonal (hence, symmetric)
- Taking $\mathbf{Z} := \mathbf{U}_k\sqrt{\mathbf{\Sigma}_k}$ $(\in \mathbb{R}^{n\times k})$, we get that $\mathbf{Z}\cdot\mathbf{Z}^\top$ is an optimal approximation of $\mathcal{P}$ (under the Frobenius norm)
  - In fact, optimal among all matrices of degree k
- Exactly what we wanted!
- Several embedding techniques utilize SVD, in either the symmetric case or the general case

Example: GraRep

- In **GraRep** [Cao et al., 2015], the function $\mathcal{P}(u, v)$ estimates the probability of starting with u in a random walk, conditioned on seeing v after ℓ steps:

$$\mathcal{P}_\ell(v_i, v_j) := \log\left(\frac{(\mathbf{B}^\ell)_{i,j}}{\sum_t (\mathbf{B}^\ell)_{t,j}}\right) - \beta$$

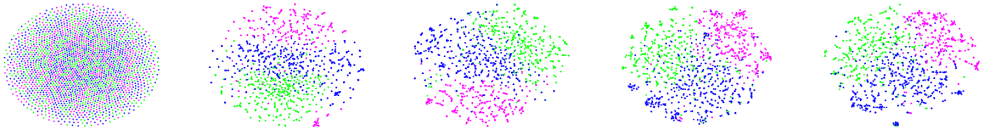
  - $\mathbf{B}$ is the adjutancy matrix $\mathbf{A}$, with each entry divided by the degree of the node of the corresponding column (to get a transition probability)
  - β is a constant bias term
- Define $\mathbf{Z}_\ell = \mathbf{U}_k \cdot \sqrt{\mathbf{\Sigma}_k}$ from the SVD of $\mathcal{P}_\ell$
- Repeat the process for $\ell = 1, \ldots, L$ and concatenate the results:

$$\mathbf{Z} := \mathbf{Z}_1 \oplus \cdots \oplus \mathbf{Z}_L$$

Experiments from the GraRep Paper

Table 4: Results on Blogcatalog

Metric	Algorithm	10%	20%	30%	40%	50%	60%	70%	80%	90%
Micro-F1	GraRep	**38.24**	**40.31**	**41.34**	**41.87**	**42.60**	**43.02**	**43.43**	**43.55**	**44.24**
	LINE	37.19	39.82	40.88	41.47	42.19	42.72	43.15	43.36	43.88
	DeepWalk	35.93	38.38	39.50	40.39	40.79	41.28	41.60	41.93	42.17
	E-SGNS	35.71	38.34	39.64	40.39	41.23	41.66	42.01	42.16	42.25
	Spectral Clustering	37.16	39.45	40.22	40.87	41.27	41.50	41.48	41.62	42.12
Macro-F1	GraRep	**23.20**	**25.55**	**26.69**	**27.53**	**28.35**	**28.78**	**29.67**	**29.96**	**30.93**
	LINE	19.63	23.04	24.52	25.70	26.65	27.26	27.94	28.68	29.38
	DeepWalk	21.02	23.81	25.39	26.27	26.85	27.36	27.67	27.96	28.41
	E-SGNS	21.01	24.09	25.61	26.59	27.64	28.08	28.33	28.34	29.26
	Spectral Clustering	19.26	22.24	23.51	24.33	24.83	25.19	25.36	25.52	26.21



(a) Spectral Clustering (b) DeepWalk (c) E-SGNS (d) LINE (e) GraRep

Figure 3: Visualization of author citation network. Each point indicates one author. Green: *Data Mining*, magenta: *computer vision* and blue: *Machine Learning*.

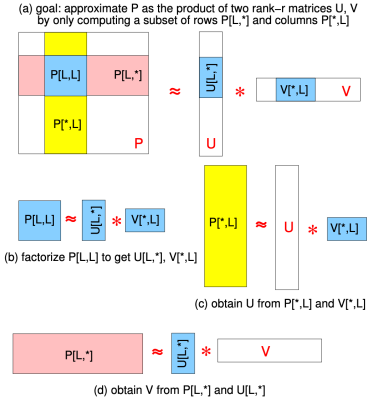
[Cao et al., 2015]

Other Usages of SVD for Node Embedding

Algorithm 3: NetMF for a Small Window Size T

1 Compute $\boldsymbol{P}^1, \cdots, \boldsymbol{P}^T$;
2 Compute $\boldsymbol{M} = \frac{\text{vol}(G)}{bT}\left(\sum_{r=1}^{T} \boldsymbol{P}^r\right)\boldsymbol{D}^{-1}$;
3 Compute $\boldsymbol{M}' = \max(\boldsymbol{M}, 1)$;
4 Rank-d approximation by SVD: $\log \boldsymbol{M}' = \boldsymbol{U}_d \boldsymbol{\Sigma}_d \boldsymbol{V}_d^\top$;
5 **return** $\boldsymbol{U}_d\sqrt{\boldsymbol{\Sigma}_d}$ as network embedding.

[Qiu et al., 2018]: SVD to unify various embedding techniques: DeepWalk, LINE, node2vec, etc.



[Song et al., 2009]: SVD on a sample for social-network-scale embedding

HOPE: SVD for Directed Graphs

- The **HOPE** algorithm [Ou et al., 2016] focuses on *directed graphs*
- Inspired by *recommendation systems*, produces two embeddings for each node v:
  - $\mathbf{z}_v^{\text{src}}$ represents v from a *source* (sender) perspective
  - $\mathbf{z}_v^{\text{tgt}}$ represents v from a *target* (receiver) perspective
- **Method**:
  - A directed variation of GraRep
  - Define an *asymmetric distance* metric $\mathcal{P} \in \mathbb{R}^{|V| \times |V|}$
    - $\mathcal{P}(u, v)$ measures the proximity from u to v
    - Configurable by many options: *Katz*, *number of 2-step paths*, *Adamic-Adar*
  - Apply truncated SVD: $\mathcal{P} \simeq \mathbf{U}_k \, \mathbf{\Sigma}_k \, \mathbf{W}_k^\top$
  - Then $\mathbf{Z}^{\text{src}} := \mathbf{U}_k \cdot \sqrt{\mathbf{\Sigma}_k}$ and $\mathbf{Z}^{\text{tgt}} := \sqrt{\mathbf{\Sigma}_k} \cdot \mathbf{W}_k^\top$

Outline

Embedding Database Tuples

- So far, we discussed the embedding of nodes in a graph using *only* the graph information
- What about database tuples?
- We discuss two approaches:
  - Extending node embeddings to hetero-graphs
    - Using the labels, i.e., the relation names
  - Database-specific embeddings
    - (Not through the standard graph representation)

Outline

How would you incorporate *labels* meaningfully in a node-embedding algorithm?

Meta-Paths

- Let $(L_\mathsf{v}, L_\mathsf{e}, M, d)$ be a hetero-graph schema
  - Recall: L_v and L_e are the sets of node and edges labels, respectively, M maps each edge label to a pair of node labels
- A *meta-path* Φ over $(L_\mathsf{v}, L_\mathsf{e}, M, d)$ is a sequence of the form

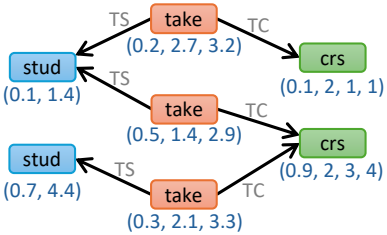
$$\sigma_0 \xrightarrow{\tau_1} \sigma_1 \xrightarrow{\tau_2} \sigma_2 \ldots \sigma_{k-1} \xrightarrow{\tau_k} \sigma_k$$

where:

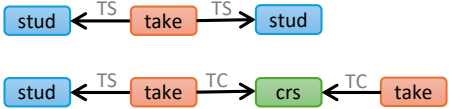
- Each σ_i is in L_v
- Each τ_i is in L_e
- $M(\tau_i) = (\sigma_{i-1}, \sigma_i)$ or $M(\tau_i) = (\sigma_i, \sigma_{i-1})$ for $i = 1, \ldots, k$

Illustration

Hetero-graph



Meta-paths



Random Walks over Meta-Paths

- Let $G = (L_V, L_E, V, E, \lambda, \mathbf{X})$ be a hetero-graph over $(L_\mathrm{v}, L_\mathrm{e}, d, M)$
- An undirected path $v_0, \ldots, v_k$ of G *conforms* to the meta-path Φ if for $i = 1, \ldots, k$ it holds that $\lambda(v_{i-1}) = \sigma_{i-1}$, and $\lambda(v_i) = \sigma_i$, and there is a τ_i-edge either from v_{i-1} to v_i or from v_i to v_{i-1}
- A *random Φ-walk* from a node v_0 is the process of building a random path $v_0, \ldots, v_k$ as follows:
  - For $i = 1, \ldots, k$, choose v_i uniformly at random from the σ_i-nodes connected to v_{i-1} via a τ_i-edge

Using Meta-Paths for Node Embedding

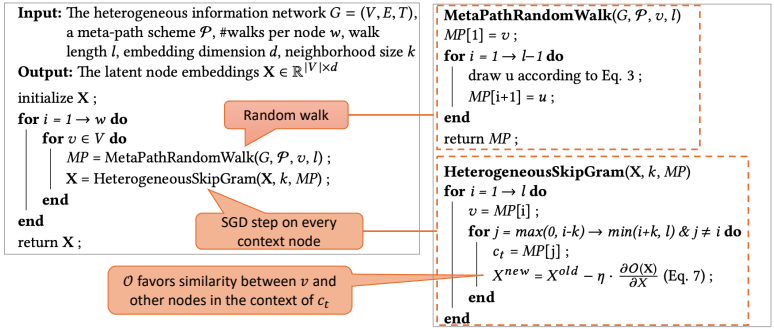
- Using meta-paths is the common way of extending embedding techniques from *homogeneous graphs* to *heterogeneous graphs*
- Examples:
  - **HIN2Vec** (HIN = Heterogeneous INformation) [Fu et al., 2017]
    - Build a simple predictor: *Given nodes u and v and a meta-path Φ, is there a path from u to v conforming to Φ?*
    - The probability estimator is a sigmoid over the product of 3 embeddings: $\mathbf{z}_u$, $\mathbf{z}_v$, $\mathbf{z}_\Phi$
  - **ESim** [Shang et al., 2016]
    - Estimate the random Φ-walks as a *Markov process* that transitions between nodes just based on their embeddings
    - Optimize the embeddings to bring the actual and Markov distributions close

$$\Pr(v \mid u, \Phi) := \frac{\exp\big(f(u, v, \Phi)\big)}{\sum_{v' \in V} \exp\big(f(u, v', \Phi)\big)} \qquad f(u, v, \Phi) := \mu_\Phi + \mathbf{p}_\Phi \cdot \mathbf{z}_u + \mathbf{q}_\Phi \cdot \mathbf{z}_v + \mathbf{z}_u \cdot \mathbf{z}_v$$

The metapath2vec Algorithm

The *metapath2vec* algorithm [Dong et al., 2017] is similar to *node2vec*, except that the paths are bound to a specific meta-path

- Negative sampling from nodes of the same type as the context node



[Dong et al., 2017]

Experiments from the metapath2vec Paper

Experiments over the AMiner dataset: 3,194,405 papers from 3,883 CS venues till 2016

Table 4: Node clustering results (NMI) in AMiner data.

methods	venue	author
DeepWalk/node2vec	0.1952	0.2941
LINE (1st+2nd)	0.8967	0.6423
PTE	0.9060	0.6483
metapath2vec	0.9274	0.7470
metapath2vec++	0.9261	0.7354

Improvement of clustering results compared to homogeneous

- NMI (Normalized Mutual Information) measures correlation between computed and true clusterings

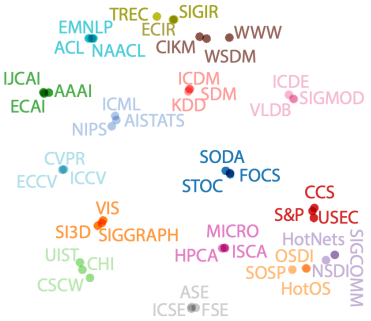


Figure 5: 2D t-SNE projections of the 128D embeddings of 48 CS venues, three each from 16 sub-fields.

Outline

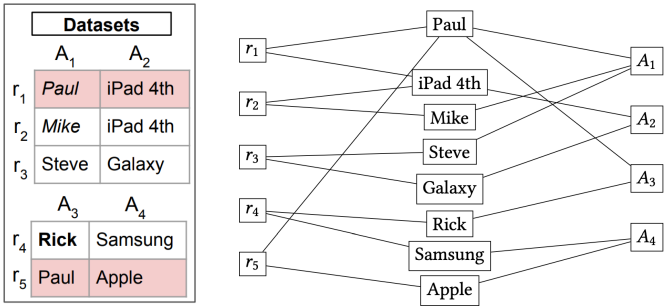
Tuple Embedding via Text Embedding

- One of the earliest ideas of embedding tuples is to *transform the database to a text document* and apply a word-embedding algorithm (e.g., word2vec) [Bordawekar and Shmueli, 2017]
  - Propose to use embeddings to enrich query languages with *similarity predicates*
- The EmbDI algorithm [Cappuzzo et al., 2020] focuses on an embedding for *data-integration* tasks:
  - Entity resolution: matching between entities of two tables
  - Schema matching: matching between attributes of two different tables

EmbDI Approach

- 3-step embedding:
  1. Build a graph G from the database D (as explained in the next slide)
  2. Sample random walks from G
  3. Treating the random walks as *sentences* in natural language, apply *word2vec*
- The graph G includes nodes for *tuples*, *cell values*, and *attribute names*
- Embeddings for records are used for entity resolution ; embeddings for attributes are used for schema matching

EmbDI Graph Illustration



[Cappuzzo et al., 2020]

Records (facts) r_i connect to their cell values that, in turn, connect to the attribute names A_j that include them

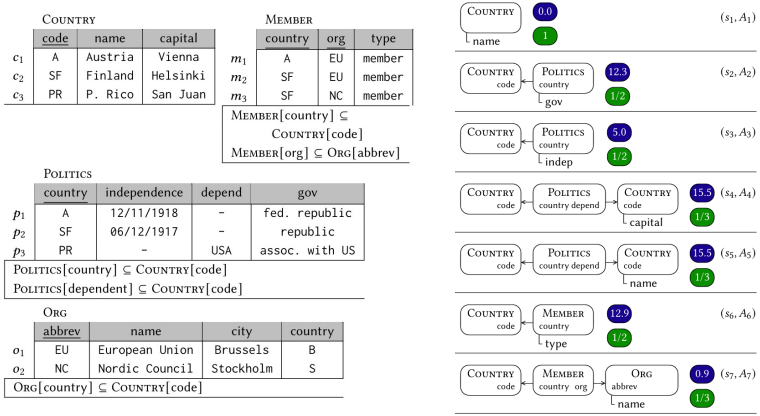
EmbDI Challenges

- Handling *numeric data* — *bucketize* them to obtain discrete values
- Handling *textual data* — apply a policy of when to keep textual data as a single value node, and when to break (tokenize) the text into multiple value nodes
- Handling *missing values* — use a unique NULL value to represent the cell

Tuple Embedding via Encoder-Decoder

- The *FoRWaRD* algorithm focuses on embeddings that can be extended to new tuples without changing the embeddings of previous tuples
  - [Tönshoff et al., 2023]
- **Idea**: Each meta-path selects an attribute from the target relation — nearby nodes should induce a similar *distribution* over the values of this attribute
- More precisely:
  - Use the standard conversion of a database to a hetero-graph (tuples become nodes, edges via foreign keys)
  - An *attribute pointing meta-path*, or *meta-path-A* for short, is a pair (Φ, A) where:
    - $\Phi = \sigma_0 \xrightarrow{\tau_1} \sigma_1 \xrightarrow{\tau_2} \sigma_2 \ldots \sigma_{k-1} \xrightarrow{\tau_k} \sigma_k$ is a meta-path
    - A is an attribute of the target node (relation) σ_k of Φ

Illustration of Meta-Path-A



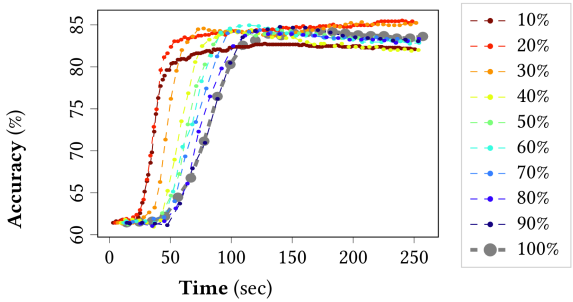
[Lubarsky et al., 2023] (boxed numbers will be clarified later)

Encoder-Decoder Scheme of FoRWaRD

- Using a large collection of predefined meta-path-As, the algorithm uses an encoder-decoder scheme, as follows
- Given a source fact $f = R(\mathbf{t})$, the meta-path-A (Φ, A) defines a distribution over the values in the A column:
  - Sample a random Φ-walk from f to a tuple $f' = R'(\mathbf{t}')$
  - Return $\mathbf{t}'[A]$
- Encoder-decoder scheme:
  - Encoder: The proximity $\mathcal{P}_{\Phi,A}(f_1, f_2)$ between two nodes (facts) f_1 and f_2 is a proximity measure between their (Φ, A) distributions
    - We do not define the specific one here, see the paper
  - Decoder: $\mathbf{z}_{f_1}^\top \mathbf{M}_{\Phi,A} \mathbf{z}_{f_2}$ for a (learned) matrix $\mathbf{M}_{\Phi,A}$

Meta-Path Selection

- Follow-up work proposed techniques for selecting the most effective meta-path-As [Lubarsky et al., 2023]
- Experimentation methodology:
  1. Select a subset of the meta-paths-As by some strategy
  2. Learn an embedding
  3. Train on a *downstream task* (simple model – sees only the tuple's embedding)
  4. After each epoch, record the time and accuracy



[Lubarsky et al., 2023]

Key Takeaways from Module 3

- Embedding in structured data aims to learn a vector representation of each item to reflect its semantics (given by the data)
- Principle: *Vector (geometric) proximity reflects semantic proximity*
- In node embedding, this approach is manifested in different instantiations of the *encoder-decoder* scheme
  - Idea: distance can be recovered via decoding over the embedding (encoding)
  - Random walks often used for incorporating the structure in semantic proximity
  - Techniques include learning a parameterized model for a probabilistic function, or matrix decomposition from linear algebra
- For relational databases, techniques involve *adapting graph embeddings to hetero-graphs* (enrichment with meta-paths), reductions to *text embedding*, and specialized encoder-decoder schemes

The End

Many thanks for helping with the slides:

- **Shunit Agmon** (Technion)
- **Boaz Berger** (Technion)
- **Ilias Fountalis** (RelationalAI)

References I


Ahmed, A., Shervashidze, N., Narayanamurthy, S. M., Josifovski, V., and Smola, A. J. (2013).
Distributed large-scale natural graph factorization.
In *WWW*, pages 37–48. International World Wide Web Conferences Steering Committee / ACM.


Bordawekar, R. and Shmueli, O. (2017).
Using word embedding to enable semantic queries in relational databases.
In *DEEM@SIGMOD*, pages 5:1–5:4. ACM.


Cao, S., Lu, W., and Xu, Q. (2015).
GraRep: Learning graph representations with global structural information.
In *CIKM*, pages 891–900. ACM.


Cappuzzo, R., Papotti, P., and Thirumuruganathan, S. (2020).
Creating embeddings of heterogeneous relational datasets for data integration tasks.
In *SIGMOD Conference*, pages 1335–1349. ACM.

References II


Dong, Y., Chawla, N. V., and Swami, A. (2017).
metapath2vec: Scalable representation learning for heterogeneous networks.
In *KDD*, pages 135–144. ACM.


Duvenaud, D., Maclaurin, D., Aguilera-Iparraguirre, J., Gómez-Bombarelli, R., Hirzel, T., Aspuru-Guzik, A., and Adams, R. P. (2015).
Convolutional networks on graphs for learning molecular fingerprints.
In *NIPS*, pages 2224–2232.


Fu, T., Lee, W., and Lei, Z. (2017).
HIN2Vec: Explore meta-paths in heterogeneous information networks for representation learning.
In *CIKM*, pages 1797–1806. ACM.


Grover, A. and Leskovec, J. (2016).
node2vec: Scalable feature learning for networks.
In *KDD*, pages 855–864. ACM.

References III


Hamilton, W. L. (2020).
Graph Representation Learning.
Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers.


Lubarsky, Y. L., Tönshoff, J., Grohe, M., and Kimelfeld, B. (2023).
Selecting walk schemes for database embedding.
In *CIKM*, pages 1677–1686. ACM.


Mnih, A. and Hinton, G. E. (2008).
A scalable hierarchical distributed language model.
In *NIPS*, pages 1081–1088. Curran Associates, Inc.


Narayanan, A., Chandramohan, M., Venkatesan, R., Chen, L., Liu, Y., and Jaiswal, S. (2017).
graph2vec: Learning distributed representations of graphs.
CoRR, abs/1707.05005.

References IV


Ou, M., Cui, P., Pei, J., Zhang, Z., and Zhu, W. (2016).
Asymmetric transitivity preserving graph embedding.
In *KDD*, pages 1105–1114. ACM.


Perozzi, B., Al-Rfou, R., and Skiena, S. (2014).
DeepWalk: online learning of social representations.
In *KDD*, pages 701–710. ACM.


Qiu, J., Dong, Y., Ma, H., Li, J., Wang, K., and Tang, J. (2018).
Network embedding as matrix factorization: Unifying DeepWalk, LINE, PTE, and node2vec.
In *WSDM*, pages 459–467. ACM.


Shang, J., Qu, M., Liu, J., Kaplan, L. M., Han, J., and Peng, J. (2016).
Meta-path guided embedding for similarity search in large-scale heterogeneous information networks.
CoRR, abs/1610.09769.

References V


Song, H. H., Cho, T. W., Dave, V., Zhang, Y., and Qiu, L. (2009).
Scalable proximity estimation and link prediction in online social networks.
In *Internet Measurement Conference*, pages 322–335. ACM.


Tang, J., Qu, M., Wang, M., Zhang, M., Yan, J., and Mei, Q. (2015).
LINE: large-scale information network embedding.
In *WWW*, pages 1067–1077. ACM.


Tönshoff, J., Friedman, N., Grohe, M., and Kimelfeld, B. (2023).
Stable tuple embeddings for dynamic databases.
In *ICDE*, pages 1286–1299. IEEE.