Embeddings part II

Random-Walk Based Embeddings

Outline

- Embedding - What & Why
- Graph Embedding
  - Node2Vec
- Relational Embeddings
  - EmbDI
  - FoRWaRD

Outline

- Embedding - What & Why
- Graph Embedding
  - Node2Vec
- Relational Embeddings
  - EmbDI
  - FoRWaRD

Motivation

***Aim**: Translate DB data into numerical data*

ID	Name
1	Bob
2	Alice

Relational DB

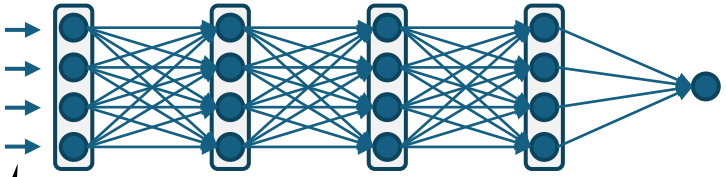
ID	Age	BMI	Smoking	Diabetes
1	23	25	current	0
2	67	31	former	1
3	50	29	never	0

Meaningless

Textual

?

Neural Network

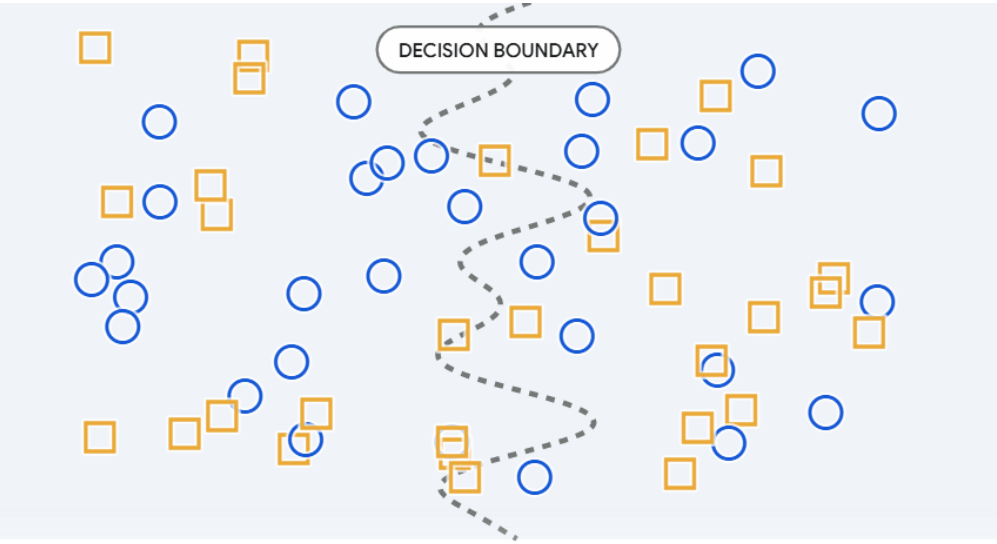


$\mathbb{R}^k$

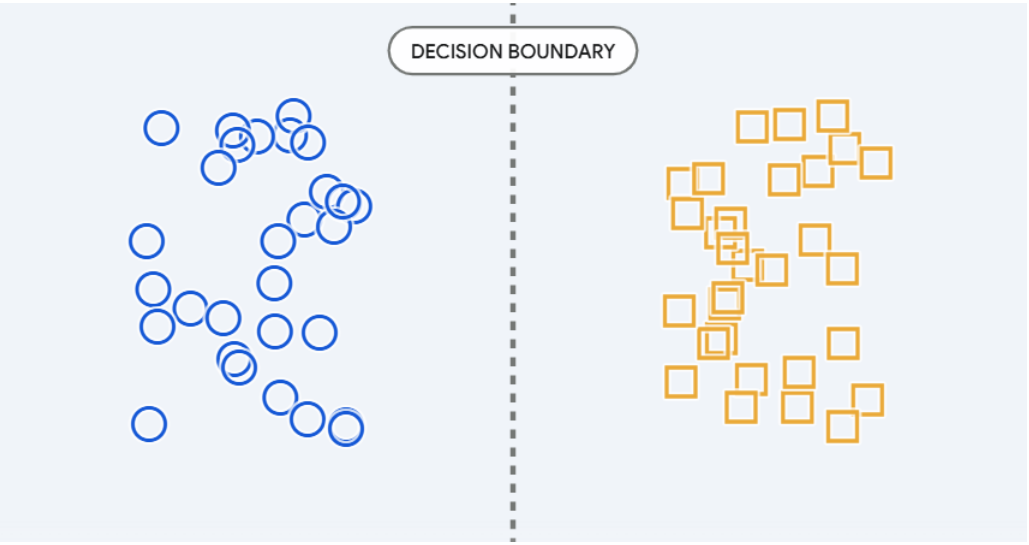
This is a **Geometric Euclidian Space**.
Does it matter how samples are distributed in it?

Motivation

Arbitrary ***Encoding***



Meaningful ***Embedding***



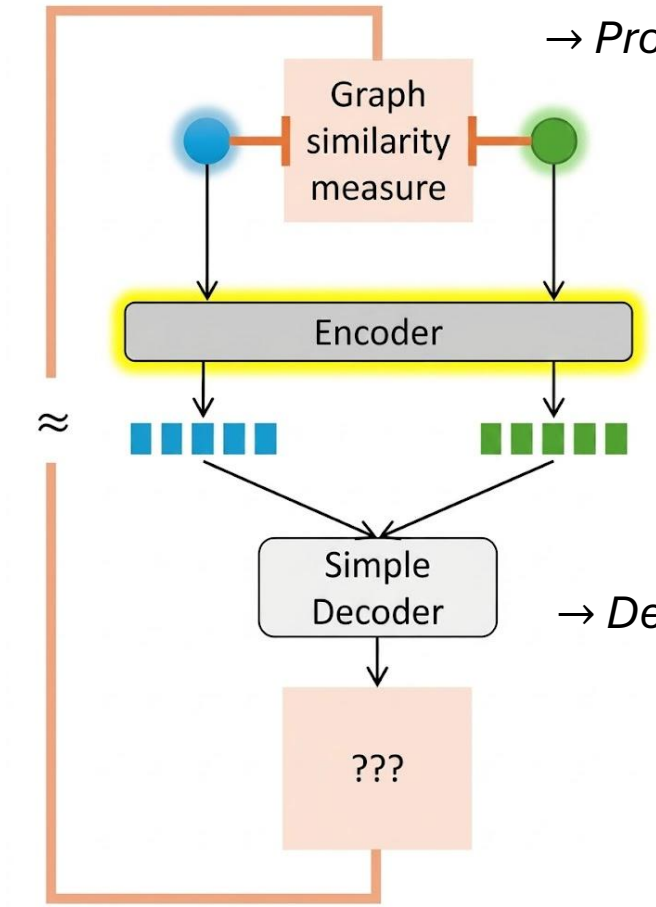
✅ Setting a meaningful embedding eases the model's ability to learn (approximate a function)
✅ Similar/connected tuples should have close embeddings
✅ Use unlabeled data to train embeddings – why?

Embedding Overview

Reminder: Encoder-Decoder Embedding Scheme

- *Fixed Embedding Dimension* → k
- *Tuple-level graph representation of the database*
- *Assumption: No features / Labels (homogenous)*

Input: graph → $G = (V, E) \quad n := |V|$



→ *Proximity Function*

$\mathcal{P}: V \times V \to \mathbb{R}$

→ a learnable map: $Z \in \mathbb{R}^{n \times \mathrm{k}}$

Loss Function ←

$\mathrm{Loss}_{\approx}: \mathbb{R} \times \mathbb{R} \to \mathbb{R}$

→ *Decoder Function*

$\mathcal{D}: \mathbb{R}^{\mathrm{k}} \times \mathbb{R}^{\mathrm{k}} \to \mathbb{R}$

$$\mathrm{cost}(\mathbf{Z}) := \sum_{u,v \in V} \mathrm{Loss}_{\approx}\left(\mathcal{P}(u,v), \mathcal{D}(\mathbf{z_u}, \mathbf{z_v})\right)$$

Outline

- Embedding - What & Why
- Graph Embedding
  - Node2Vec
- Relational Embeddings
  - EmbDI
  - FoRWaRD

Node2Vec

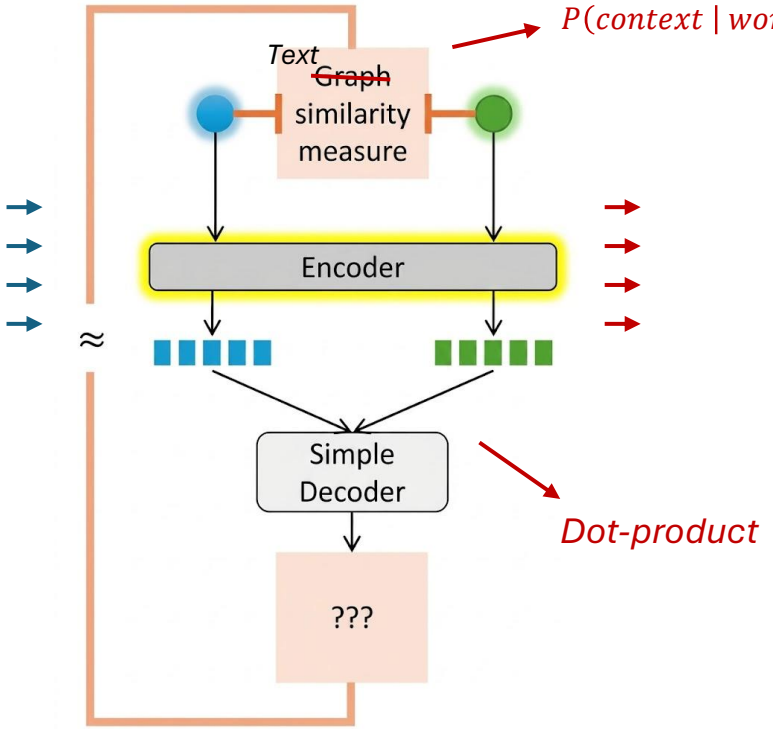
- Another encoder-decoder embedding for graphs
- Comes from the famous ***word2vec*** for text – learn from words context

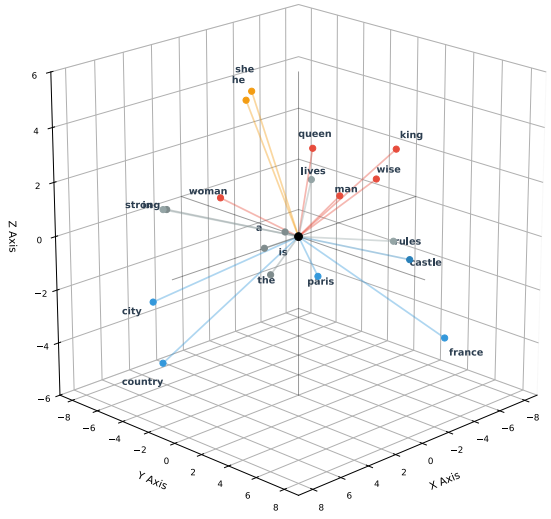
Sentences

The	quick	brown	fox	jumps

Context

(brown, the),
(brown, quick),
(brown, fox),
(brown, jumps)





⇒ So let's learn **node** embeddings from nodes context!

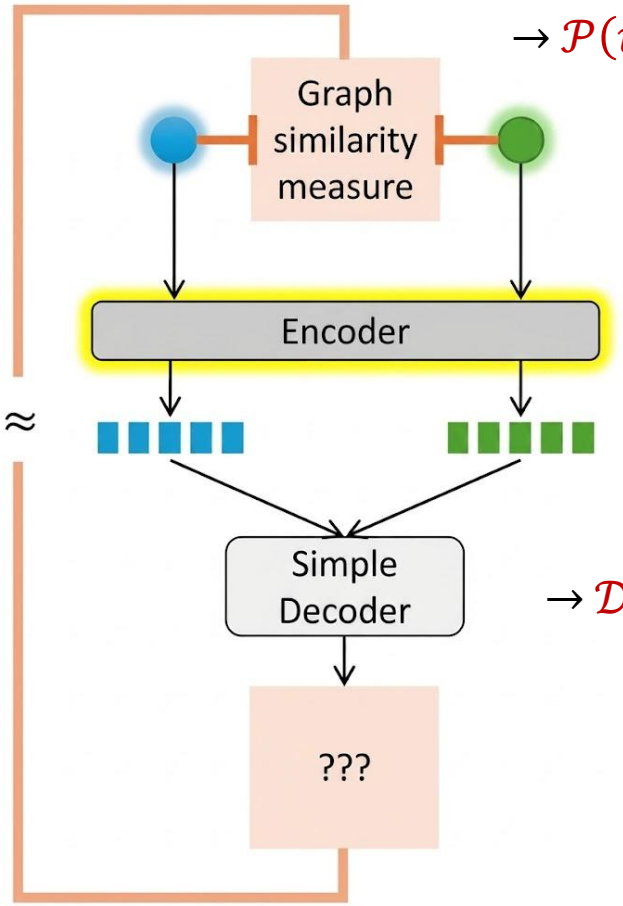
word2vec: Efficient Estimation of Word Representations in Vector Space
node2vec: node2vec: Scalable Feature Learning for Networks

Node2Vec

- Now we choose:

$\rightarrow \mathcal{P}(u, v) \in [0,1]$

Probability to occur in a **random walk**



Loss Function $\leftarrow \approx$

$$\text{Loss}_{\approx}(u, v) = -\log\left(\frac{\exp(\mathbf{z}_u \cdot \mathbf{z}_v)}{\sum_{w \in V} \exp(\mathbf{z}_w \cdot \mathbf{z}_v)}\right)$$

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

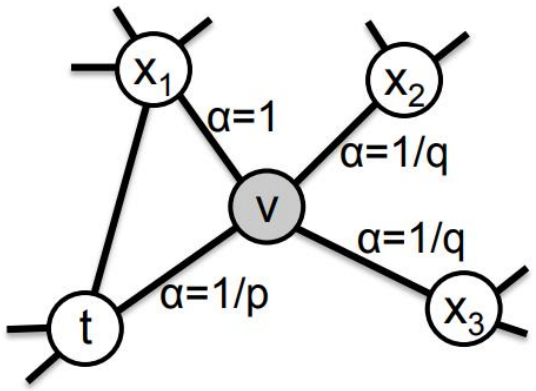
Node2Vec Proximity

$\mathcal{P}(u,v) \in [0,1]$ = probability of v to occur in a random walk starting from u

How do we randomize a walk?

- Hyperparameters: p,q
- 2nd order - previous: t, current: v, next: x
- Calculate at each step:

$$\alpha(t,x) = \begin{cases} \frac{1}{p} & t = x \\ 1 & dist(t,x) = 1 \\ \frac{1}{q} & dist(t,x) = 2 \end{cases}$$



💡 p Controls revisiting a node:
- higher p – less chance of revisiting
- lower p – more chance of revisiting

💡 q controls locality:
- lower q – less locality = more DFS-like
- higher q – more locality = more BFS-like

- Next step probability: $\Pr(x|v,t) = \frac{\alpha(t,x)\cdot w_{vx}}{NORM}$
  - Normalization factor $NORM$, edge weights w_{vx}
- Final proximity value $\mathcal{P}(u,v)$: $\frac{\#\text{Occurrences of v in u's walk}}{\#\text{Total walks originated from u}}$

💡 There are more random walk techniques (i.e uniform dist)

Node2Vec Decoder

Normalized dot product:

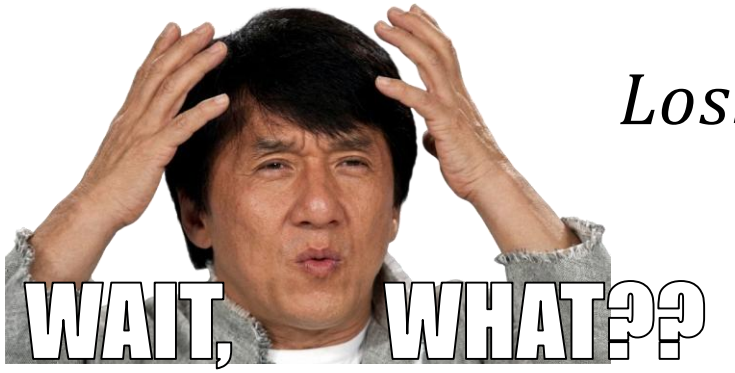
Forces positivity

Similarity - like before

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

Normalize approximated probability - sum to 1

Node2Vec Loss



$$Loss(u,v) = -\log\left(\frac{\exp(\mathbf{z}_u \cdot \mathbf{z}_v)}{\sum_{w\in V}\exp(\mathbf{z}_w \cdot \mathbf{z}_v)}\right)$$

Why not $Loss(u,v) = \left(\mathcal{P}(u,v) - D(u,v)\right)^2$ (=MSE)

Problem

- Hard to calculate real $\mathcal{P}$

Word2vec Solution

- skip-gram architecture which uses a different loss…

KL-Divergence

how close is a real probability distribution P to a predicted probability distribution Q?

$$D_{KL}(P \parallel Q) = \sum_{x \in \mathcal{X}} P(x) \log\left(\frac{P(x)}{Q(x)}\right)$$

For $P \approx Q \Rightarrow \frac{P(x)}{Q(x)} \approx 1$

Weigh by the significance of the sample ⇒
Rare cases are not important

Scale the score. “good” ratio = 1 yields 0

- The lower is better ⇒ useful as a cost function
- *Problem*: still don’t know P

Cross Entropy derived from KL-Divergence

Trick! We know this: $\log\left(\frac{a}{b}\right) = \log(a) - \log(b)$

Let's apply:

$$\sum_{x \in \mathcal{X}} P(x) \log\left(\frac{P(x)}{Q(x)}\right) = \underbrace{\sum_{x \in \mathcal{X}} P(x) \log(P(x))}_{\text{Const! Zeroed at derivative}} - \underbrace{\sum_{x \in \mathcal{X}} P(x) \log(Q(x))}_{\text{Let's just keep this}}$$

Approximated by sampling over our dataset, where sample frequency implicitly accounts for $P(x)$

Back to Node2Vec Loss

Applying all of this to our setting gives us the loss:

$$Loss(u, v) = -\log\big(\mathcal{D}(\mathbf{z}_u, \mathbf{z}_v)\big)$$

Instantiating $\mathcal{D}$:

$$Loss(u, v) = -\log\left(\frac{\exp(\mathbf{z}_u \cdot \mathbf{z}_v)}{\sum_{w \in V} \exp(\mathbf{z}_u \cdot \mathbf{z}_w)}\right)$$

New problem:

- Summing over all the vertices? For each vertex??

Negative sampling

$$Loss(u,v) = -\log\left(\frac{\exp(\mathbf{z}_u \cdot \mathbf{z}_v)}{\sum_{w \in V} \exp(\mathbf{z}_u \cdot \mathbf{z}_w)}\right)$$

💡 Fix a k and sum over k negative samples?

❌ will not sum up to 1

$\Rightarrow$

💡 Replace:
- $P(u,v)$ = "is v part of the random walk starting at u?"
  - Independently of other nodes
- Softmax with Sigmoid ($\boldsymbol{\sigma}$)

✅ No need to sum to 1

$$\Rightarrow Loss(u,v) \coloneqq \underbrace{-\log\big(\sigma(\mathbf{z}_u \cdot \mathbf{z}_v)\big)}_{\text{Positive sample } (u,v) \sim Random\ walks} - \underbrace{\sum_{i=1}^{k} \log\Big(\sigma\big(-\mathbf{z}_u \cdot \mathbf{z}_{n_i}\big)\Big)}_{\text{Negative samples } \{(u,n_i)\}_{i=1}^{k}}$$

This is a simplified variant of Noise Contrastive Estimation (NCE) optimized specifically for learning embeddings

- For reference: word2vec explained https://arxiv.org/pdf/1402.3722.pdf

Node2Vec Limitations

Transductive

New nodes entering the graph require recalculation of the whole embeddings

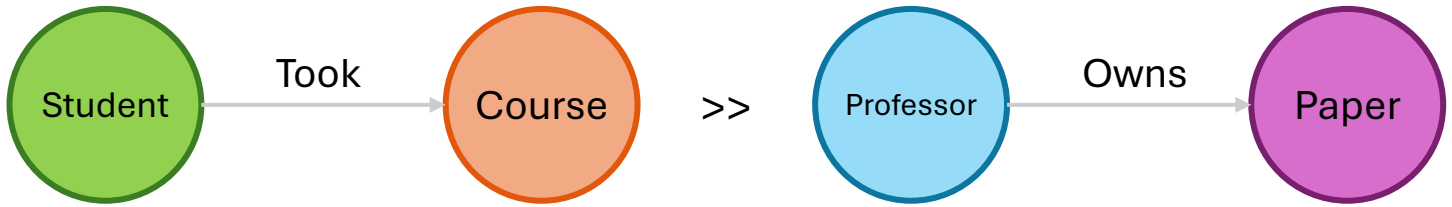
Node Feature Blindness

Ignores node features from heterogenous graphs

Label Blindness

Ignores node & edge labels from heterogenous graphs

***Predict**: Student's next course*



Outline

- Embedding - What & Why
- Graph Embedding
  - Node2Vec
- Relational Embeddings
  - EmbDI
  - FoRWaRD

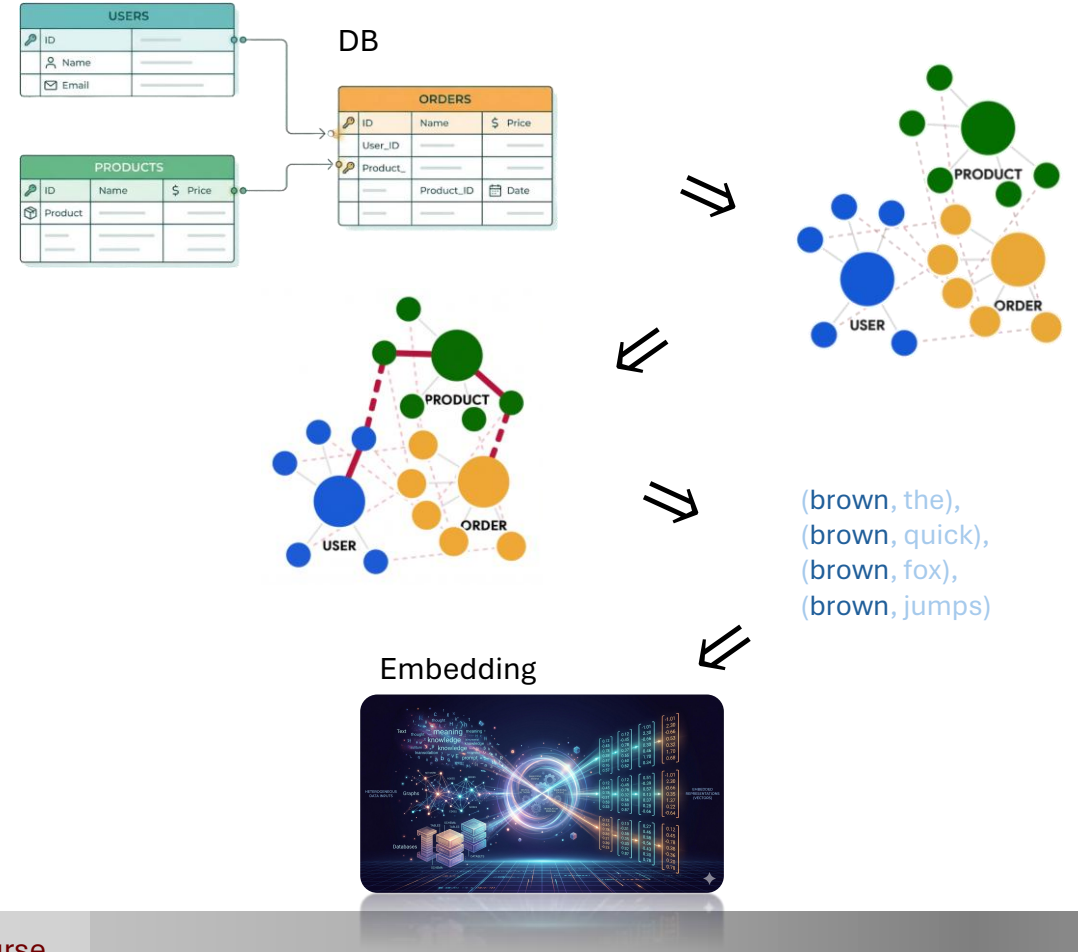
EmbDI

Focus: Embedding from DB, rather than from graph

EmbDI Algorithm

Input: **database**

- Convert DB to heterogenous graph
- Generate heterogenous random walks
- Apply text embeddings over the walks as sentences



Reference: Creating Embeddings of Heterogeneous Relational Datasets for Data Integration Tasks

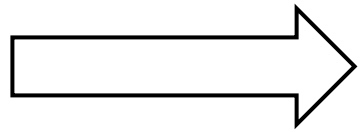
Phase 1 – Graph Construction

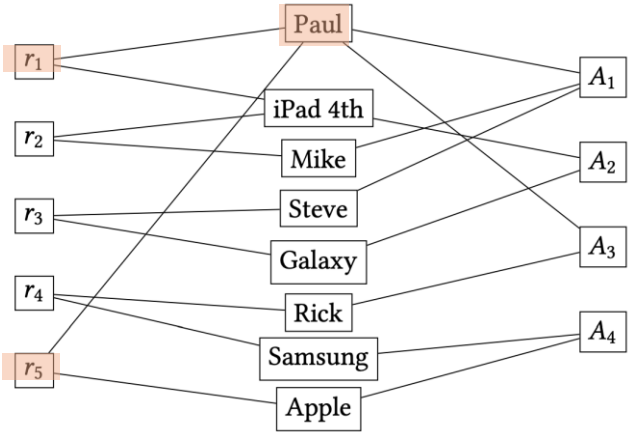
Conversion Method: Tripartite graph - ***Row*** ↔ ***Cell*** ↔ ***Attribute***

Datasets

	A_1	A_2
r_1	*Paul*	iPad 4th
r_2	*Mike*	iPad 4th
r_3	Steve	Galaxy

	A_3	A_4
r_4	**Rick**	Samsung
r_5	Paul	Apple





Idea: Incorporate Relational Context

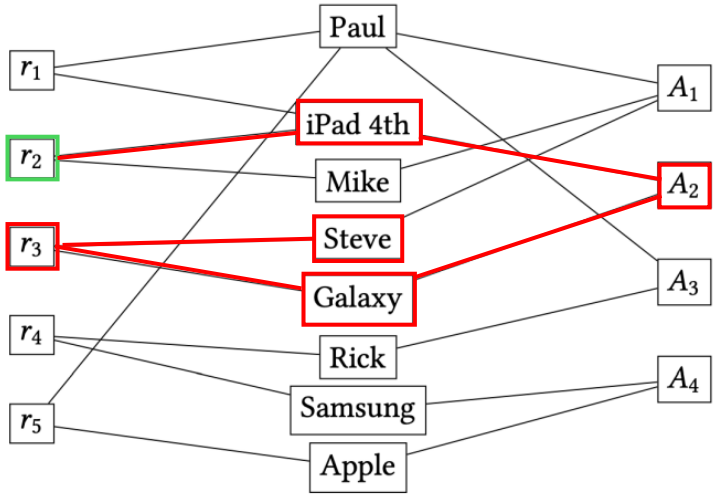
✅ Enables detecting identical entities (**Entity Resolution**)

⇒ “User123” = “User456” (same person)
⚠️ “Apple” as in iPhone ≠ “Apple” as the fruit
⇒ enough data will cluster them away

Phases 2-4 – Embedding Generation

Phase 2: Random Walk Generation:

- Start: at each node
- Next step strategy: Uniform



Phase 3: Convert walk ⇒ sentence

- Trivial: Node content ⇒ word

Phase 4: Apply word2vec

💡 What should be the window size?
⇒ 3: semantically strong enough

⇨ Sentence 1: r_2 'iPad 4th' A2 Galaxy r_3 Steve
Sentence 2: …
Sentence 3: …
…

⇨ word2vec ⇨ ***Embedding***

EmbDI Limitations

❌

Transductive

New nodes entering the graph requires recalculation of the whole embeddings

✅

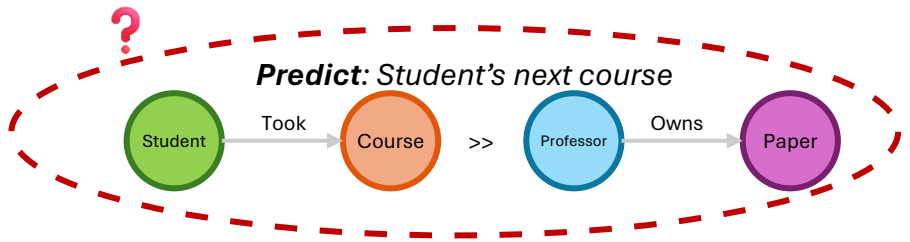
Treating Node Features

Incorporates node features from heterogenous graphs (via nodes)

❌

Labels Blindness

Ignores node & edge labels from heterogenous graphs



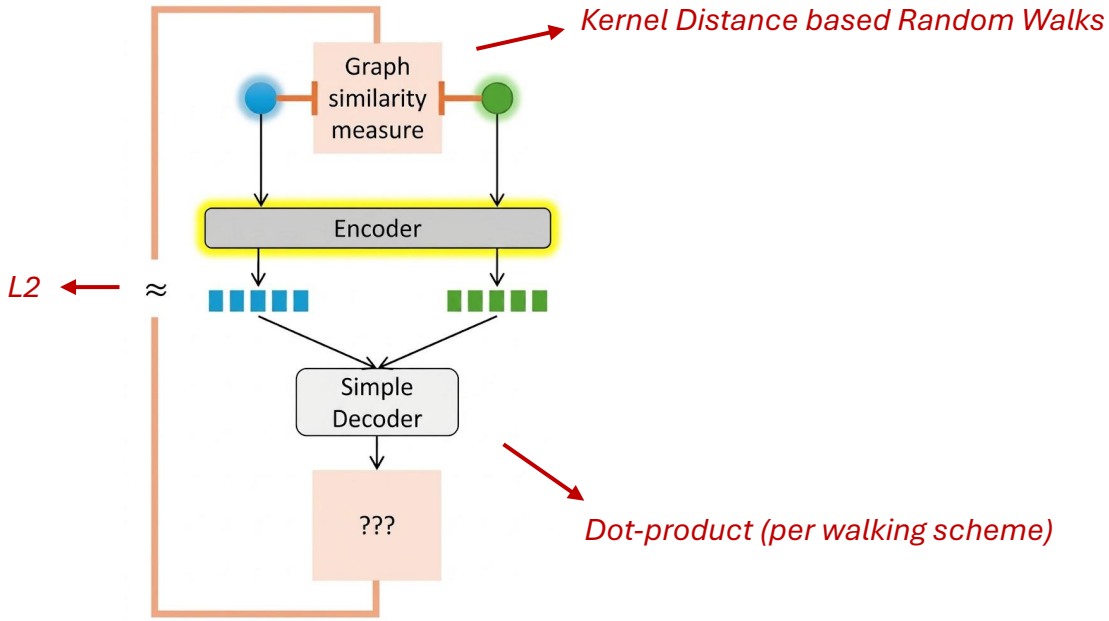
Outline

- Embedding - What & Why
- Graph Embedding
  - Node2Vec
- Relational Embeddings
  - EmbDI
  - FoRWaRD

FoRWaRD

FoRWaRD - **Fo**reign Key **R**andom **Wa**lk Embeddings for **R**elational **D**atabases

- Conversion: DB ⇒ tuple-level heterogeneous graph
- Encoding-Decoding scheme
- Random walks on labeled paths to calculate similarity



Reference: Stable Tuple Embeddings for Dynamic Databases

FoRWaRD – Random Walks

Meta-path: A labeled path scheme over the heterogeneous graph:

- $\sigma_i \in L_V, \tau_j \in L_E$ $(i \in \{0, \dots, k\}, j \in \{1, \dots, k\})$

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

A *Φ-walk* is a walk over the scheme

Example:

Users

user_id	age	country	Signup date
1	25	Israel	2023-10-01
2	40	USA	2023-09-15

Purchases

purchase id	User id (FK)	Item id (FK)	Purchase date	quantity
100	1	10	2024-01-01	1
101	2	10	2024-01-10	2
102	2	20	2024-01-05	1

Items

Item id	category	price
10	Books	50
20	Electronics	500

Meta-path Φ: Users — Purchases — Items

Φ-walk: (Users(1,25,Israel,2023-10-01), Purchases(100,1,10,2024-01-01,1), Items(10,Books,50)) → Fact

FoRWaRD – Proximity

Meta-path-A: (Φ, A) where a meta-path Φ is coupled with an attribute $A \in \sigma_k$

- Set a list $\mathcal{M}$ of predefined (Φ, A) pairs, and for each – sample walks, and for each walk:

$$\mathcal{P}_{\Phi,A}(f_1, f_2) = KD(d_{f_1}[A], d_{f_2}[A])$$

Similarity measure over A

Destination tuple of the walk originating at f_1/f_2

Example:

$\Phi = \text{Users} - \text{Purchases} - \text{Items}$, $A = \text{category}$, $KD(c_1, c_2) = \begin{cases} 1, & c_1 = c_2 \\ 0, & c_1 \neq c_2 \end{cases}$

Users

user_id	age	country	Signup date
1	25	Israel	2023-10-01
2	40	USA	2023-09-15

Purchases

purchase id	User id (FK)	Item id (FK)	Purchase date	quantity
100	1	10	2024-01-01	1
101	2	10	2024-01-10	2
102	2	20	2024-01-05	1

Items

Item id	category	price
10	Books	50
20	Electronics	500

$\neq \Rightarrow$ push away original tuples

$= \Rightarrow$ get closer original tuples

$\Phi = \text{Users} - \text{Purchases} - \text{Items}$, $A = \text{price}$, $KD(a, b) = e^{\frac{-(a-b)^2}{2v}}$

RBF Kernel

FoRWaRD – Decoding & Loss

Extend dot product to each walking scheme:

- 💡 *Idea*: Separate the embedding learning for different meta-paths.

$$D_{\Phi,A}(f_1, f_2) = z_{f_1}^{\top} M_{\Phi,A} z_{f_2}$$

Learned matrix for each Meta-Path-A

  - $z_{f_i} \in \mathbb{R}^d, M_{\Phi,A} \in \mathbb{R}^{d \times d}$

- *Final Loss* (L2): $Loss_{\Phi,A}(f_1, f_2) = \frac{1}{2}\left(D_{\Phi,A}(f_1, f_2) - \mathcal{P}_{\Phi,A}(f_1, f_2)\right)^2$
  - Jointly optimize across a predefined list of (Φ, A) pairs $\mathcal{M}$
  - $Loss\ (f_1, f_2) = \sum_{(\Phi,A)\in\mathcal{M}} Loss_{\Phi,A}(f_1, f_2)$

- New fact f_{new} ⇒ Sample walks only from it + solve: $z_{f_{new}} = Cb$
  - For constants $C \in \mathbb{R}^{d \times k}, b \in \mathbb{R}^k$
  - Inductive! ✅

FoRWaRD Capabilities

✅ **Inductive**

new nodes in the graph don't require recalculation of all the embeddings

✅ **Treating Node Features**

Incorporates node features from heterogenous graphs (via facts)

✅ **Treating Labels**

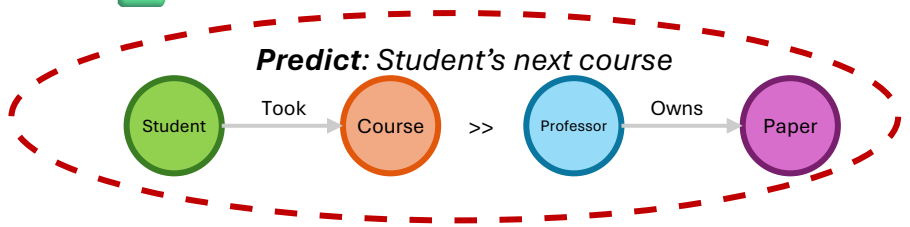
Incorporates node & edge labels from heterogenous graphs (via meta-paths)

Though now...

❌ **Staleness - Frozen Embeddings**

Trained tuples keep the same embeddings forever

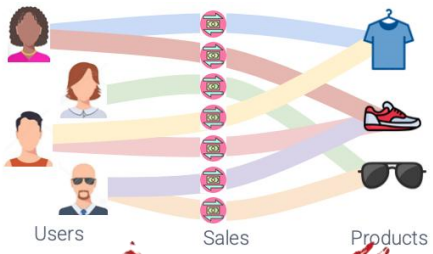
✅



Big Picture

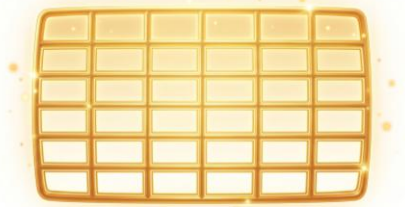
Input Database

Graph Representation



Feature Engineering

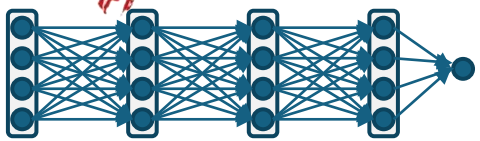
From Graphs



Big Table

Embedding

Graph Factorization
node2vec
EmbDI
FoRWaRD
...



Learning Algorithm

Prediction