Graph Representation Learning for Web-Scale Recommender Systems

Ahmed El-Kishky, Michael Bronstein, Ying Xiao, Aria Haghighi

August 14, 2022

Lecturers

Ahmed El-Kishky
Staff ML Researcher
Twitter Inc.
aelkishky@twitter.com

Michael Bronstein
Professor of Computer Science
Oxford University
michael.bronstein@cs.ox.ac.uk

Ying Xiao
Sr. Staff Researcher
Twitter Inc.
yxiao@twitter.com

Aria Haghighi
Senior Eng. Manager
Twitter Inc.
ahaghighi@twitter.com

Tutorial Outline

1. **Introduction & Motivations**
2. **Homogenous Graph Representation Learning**
3. **Heterogeneous Graph Representation Learning**
4. **Break**
5. **Graph Neural Networks**
6. **Graph-based Representations for Recommender Systems**

Intro and Motivation

Aria Haghighi

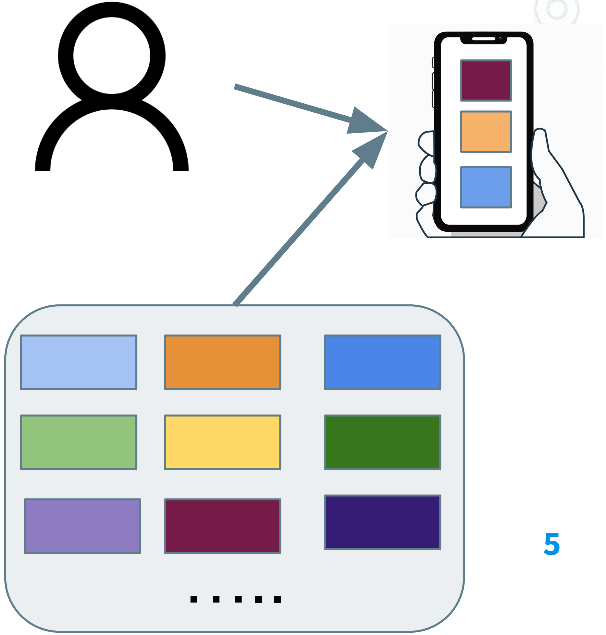
Recommender Systems

Technical Definition

- Given candidate *items* (**i**), rank items by relevance for a given user *u's* preferences
- **CTR model**: Relevance is probability of "engagement" (click, watch, follow, like, etc.)

Caveats

- Other formulations and variations exist (e.g, LTV, non-personalized, etc.)
- Production systems have many more components and rules



Recommender Systems

Many Applications For Different “items”

- Ads ranking [Ads]
- Account recommendations for social networks [Suggested User]
- Content recommendation for streaming services (e.g, Netflix, Disney+, etc.) [Videos]

Importance

- Recommender systems are typically the ML models closest to business objectives (e.g, Ads revenue, growing social graph, watch time)

Approaches To Recommender Systems

Content-Based
Item-item similarity. Useful when few engagements

- Vector space document model
- Transformer-based representations of items (E.g, BERT or CLIP)

	🟩	🟪
👤	3	2
👤	1	0

Collaborative Filtering (CF)
Leverage (user, item) engagement behavior

- Matrix factorization
- Predictive models (i.e, DLRM)

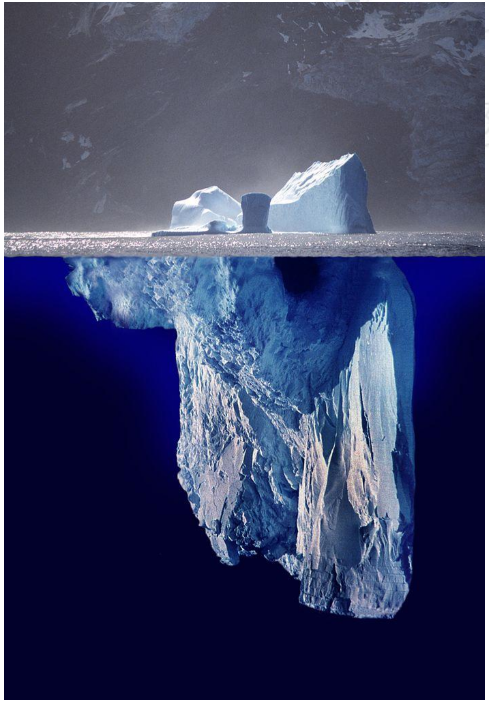
◎ Production systems are usually mixture of both approaches

◎ This tutorial focused on collaborative filtering, but some content-based extensions

Recommender System Challenges

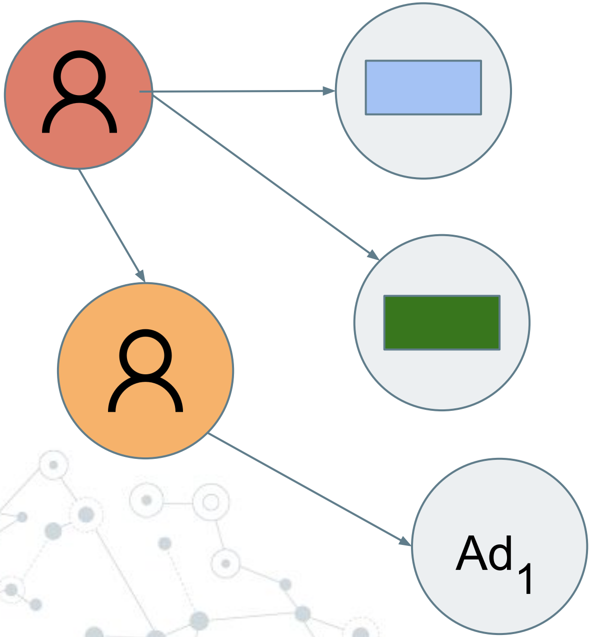
Sparsity and Cold-Start

- CF works reasonably well when there is (user, item) *density*
- **Cold-start**: When user or item has little-to-no past engagements to power CF.
  a. Prevalent for sparse engagement targets (e.g, performance ad actions like e-commerce purchases)
- **This tutorial**: Pre-trained graph embeddings can address cold-start and sparse recommendation problems



Tutorial In A Nutshell

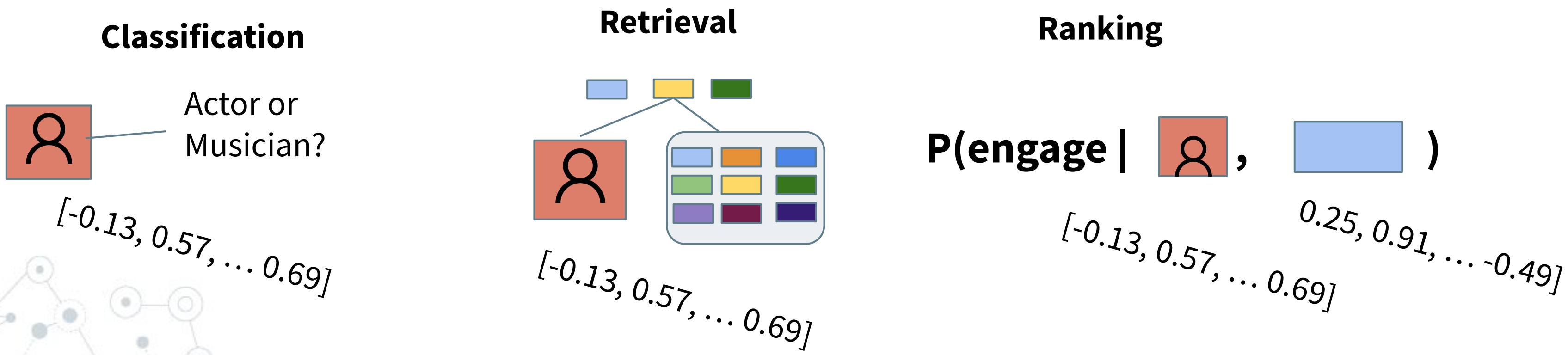
- Build graph of interactions between users, items, and other domain entities (e.g, ads, advertisers, content tags, etc.)
- Embed all graph entities



🟥	[-0.13, 0.57, … 0.69]
🟧	[-0.44, 0.29, … -0.53]
🟦	[0.92, -0.21, … -0.65]
🟩	[0.29, -0.11, … -0.41]

Tutorial In A Nutshell

- These pre-trained entity embeddings can be used for many different tasks involving business entities
  - Entity classification (e.g, account classification)
  - Recommendation candidate retrieval
  - Inputs to recommendations ranking models



Classification

Actor or Musician?

[-0.13, 0.57, … 0.69]

Retrieval

[-0.13, 0.57, … 0.69]

Ranking

P(engage | ,)

[-0.13, 0.57, … 0.69]

0.25, 0.91, … -0.49]

Homogenous Graph Representation

Aria Haghighi

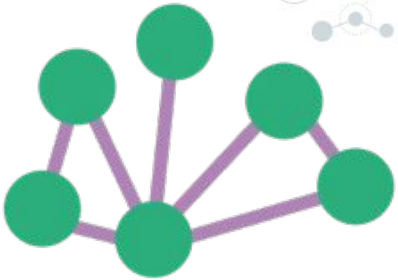
Homogeneous Graph Representations

Homogeneous Graphs

- Single node type and single edge type
- Twitter
  a. users *follow* other users

Running Application Example

- Nodes represent users and (single) edge type for user following relation
- Account recommendation: What account should a user follow?



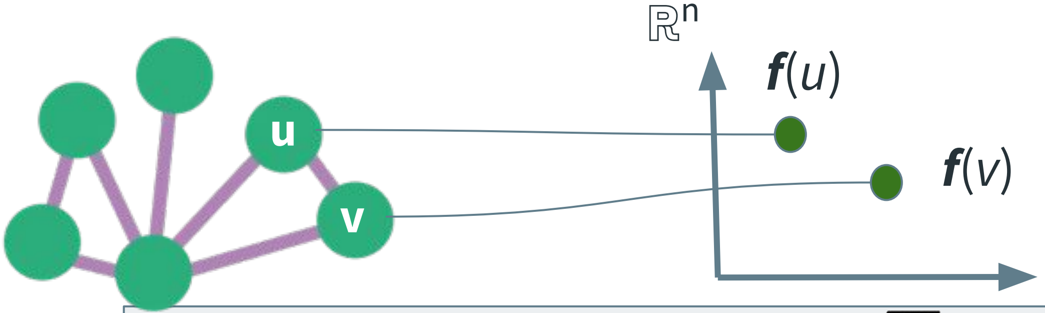
homogeneous

Future sections will generalize to heterogeneous graphs (multiple edge types)

Homogeneous Graph Representations

Node Embeddings

- Represent each graph node *u* by a vector, or embedding, ***f***(*u*) in $\mathbb{R}^n$
- Learn ***f*** so that "similar" nodes (*u*, *v*) map to vectors ***f***(*u*) and ***f***(*v*) close together



$$\mathtt{Sim}(u,v) \approx \mathbf{f}(u)^T\mathbf{f}(v)$$

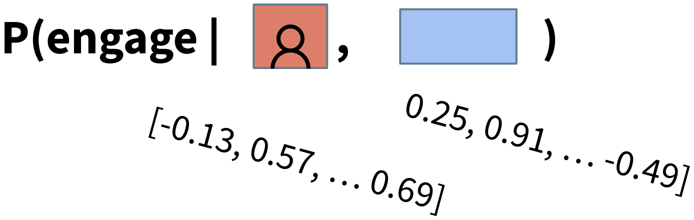
Homogeneous Graph Representations

Why bother with embeddings?

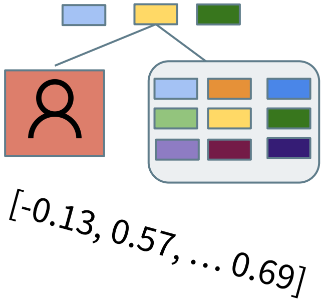
- Translate complex relational data into representation more amenable for Deep ML models

- Querying for “similarity” is more efficient leveraging approximate nearest neighbor (ANN) algorithms

Ranking



Retrieval



Random Walk Approaches

Defining Objective

- Very similar to word2vec
- Given nodes “similar” to node *u*, denoted **S**(*u*), assign node embedding to maximize probability of this “observed” data

$$P(\mathbf{S}(u)|u) = \prod_{v \in \mathbf{S}(u)} P(v|u)$$

$$= \prod_{v \in \mathbf{S}(u)} \left[\frac{\exp\left(\mathbf{f}(u)^T \mathbf{f}(v)\right)}{\sum_{v' \in V} \exp\left(\mathbf{f}(u)^T \mathbf{f}(v')\right)} \right]$$

Random Walk Approaches

Two Modeling Choices

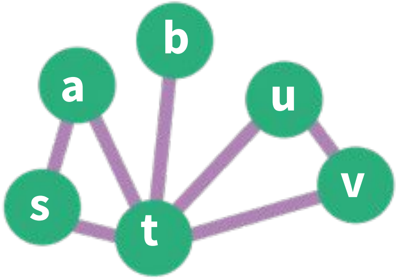
- How do we choose "similar" nodes **S**(*u*)?
  a. Determines kind of similarity captured by embeddings
- How to avoid computing denominator of P(*v* | *u*)?

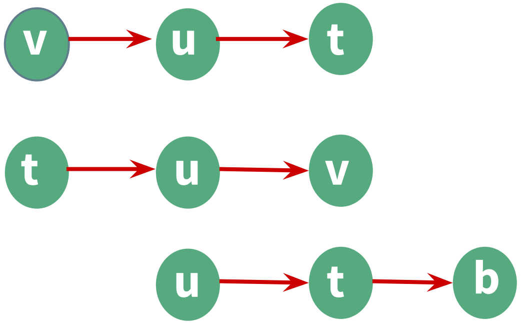
$$\left(\sum_{v' \in V} \exp \mathbf{f}(u)^T \mathbf{f}(v') \right)$$

Random Walk Approaches

DeepWalk (KDD ‘14, Perrozi et. al.)

- **Similar Nodes S**(*u*): Sample fixed-length random walks from each node. **S**(*u*) are nodes in a window around *u* weighed by window co-occurrence in sampled walks





S(*u*)= {

v (2), t (3), b (1)

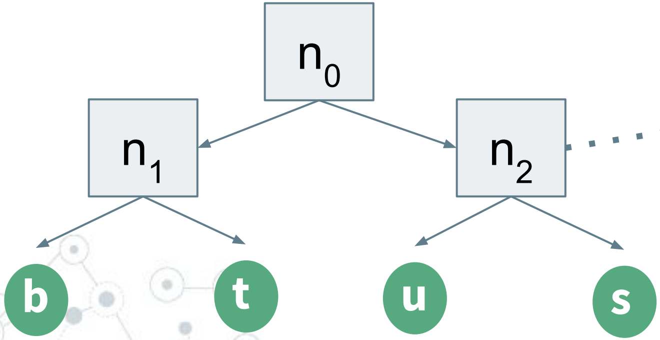
}

[Citation]

Random Walk Approaches

DeepWalk (KDD ‘14, Perrozi et. al.)

- Model P(*v* | *u)* using *hierarchical softmax*
- Create binary tree, where leaves are nodes **V**.
   a. Each binary branch has a probability of going left (or right) given input embedding, **f**(*u*).
   b. P(*v* | *u)* is product of binary choices in path to *v*



Sigmoid

$$P(\texttt{left}|\mathbf{f}(u)) = \sigma\left(W_{n_2}^T \mathbf{f}(u)\right)$$

$$P(\texttt{right}|\mathbf{f}(u)) = 1 - P(\texttt{left}|\mathbf{f}(u))$$

Random Walk Approaches

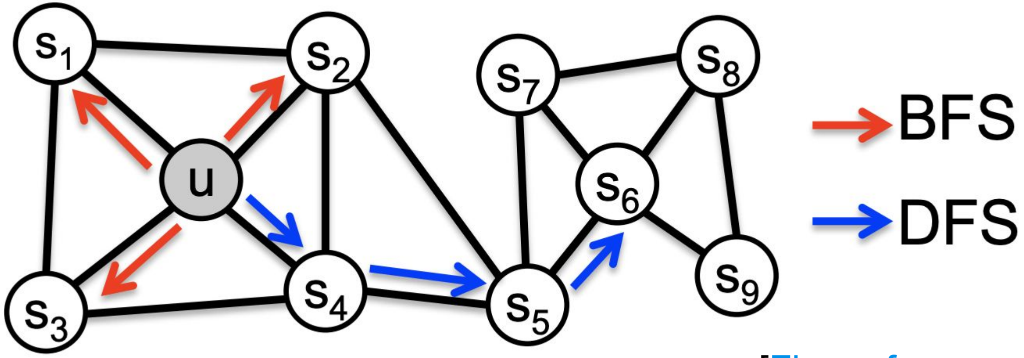
Recap of **DeepWalk** (KDD ‘14, Perrozi et. al.)

- Learn embeddings of dimension *d* for each node in **V**
  a. This entails d |**V**| parameters to learn (e.g, embedding table)
- Sample short random walks for each node, use context window frequency for similarity multiset **S**(*u*)
- Hierarchical-softmax to model P(*v*|*u*) as sequence of binary decisions conditioned on embedding of *u*
  a. Can use arbitrary coding mechanism, but Huffman encoding used originally (what benefit?)
  b. This adds d (|**V**|-1) parameters (why?)

Random Walk Approaches

node2vec (KDD ‘16, Grover & Leskovec)

- **Similar Nodes S**(*u*): Similar to DeepWalk, but richer parametrization of random walks to allow flexibility
- Breadth-first search (BFS) and Depth-First search (DFS) yield a *microscopic* (local) and macroscopic (global) view of the graph respectively



[Figure from node2vec paper]

Random Walk Approaches

node2vec (KDD ‘16, Grover & Leskovec)

- **Biased Random Walk:** Introduce hyper-parameters *p* and *q* which will allow you to interpolate between a more BFS vs DFS-like random walk
- Imagine we just traversed (*s*, *u*) edge in our random walk. Compute 2nd order transition probabilities P(*t* | *s*, *u*)

$$P(t|s,u) = \frac{\alpha(t,u)}{\sum_{t' \in N(u)} \alpha(t',u)}$$

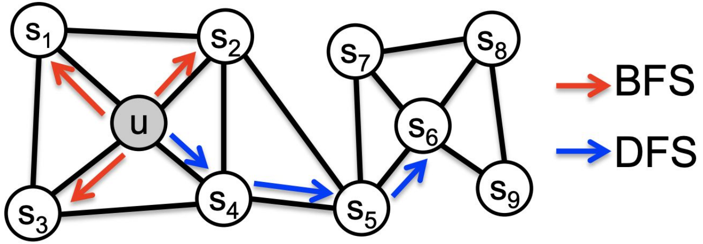
$$\alpha(t,u) = \begin{cases} p^{-1} & t = u \text{ [Return]} \\ 1 & d(t,u) = 1 \text{ [Adjacent]} \\ q^{-1} & d(t,u) = 2 \text{ [Wander]} \end{cases}$$

Random Walk Approaches

node2vec (KDD ‘16, Grover & Leskovec)

- Small *p* (large p^{-1}) is more BFS-like since encourage walk to stay close to start
- Small *q* (large q^{-1}) is more DFS-like since encourage walk to wander further away
- Recover DeepWalk sampling for *p*=*q*=1

$$\alpha(t,u) = \begin{cases} p^{-1} & t = u \text{ [Return]} \\ 1 & d(t,u) = 1 \text{ [Adjacent]} \\ q^{-1} & d(t,u) = 2 \text{ [Wander]} \end{cases}$$



Random Walk Approaches

node2vec (KDD ‘16, Grover & Leskovec)

- SkipGram Objective
  a. **Negative Sampling**: Approximate denominator by sampling from distribution, **D**(u) , over “negative” contexts for node u
  b. **Noise Contrastive Estimation (NCE)**: optimize probability of true vs false “negative samples”

$$\sum_{v \in \mathbf{S}(u)} \lg \sigma(\mathbf{f}(u)^T \mathbf{f}(v)) + \sum_{v' \in \mathbf{D}(u)} \lg \sigma(-\mathbf{f}(u)^T \mathbf{f}(v))$$

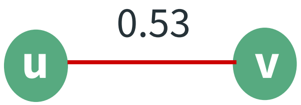
Random Walk Approaches

Recap

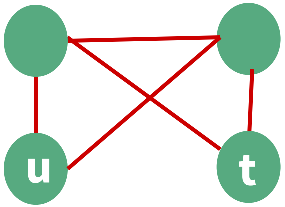
- Embed graph nodes by preserving pairwise node similarity, where node similarity is defined by co-occurence of nodes in a random walk
- DeepWalk samples short random walks uniformly, but node2vec has hyper-parameters to encourage walks to interpolate between DFS and BFS (to capture macro- and micro- concepts of similarity)
- For the user following graph, this yields user embeddings capturing similar follow behavior
  a. **Similar Accounts**: Retrieve nearest neighbors of a given user’s embedding
  b. **Account Classification**: Build a model with user embeddings as input

Higher-Order Methods

- Instead of obtaining "similar" nodes via random walk sampling, can we directly model graph properties?
- Graph Proximity
  a. **First-Order (L1)**: pairwise proximity between two nodes that are connected (typically an edge weight)
  b. **Second-Order (L2)**: pairwise proximity between two nodes, not connected but sharing neighbors



L1



L2

Higher-Order Methods

Large-Scale **I**nformation **N**etwork **E**mbedding (LINE) [WWW ‘15, Tang et. al.]

- Define an empirical measure of First-Order proximity and a model-based prediction. We want to tune embedding table to bring empirical close to model. **Note**: Only applies to undirected graphs.

Empirical

$$\hat{P}(u,v) \propto w_{u,v}$$

Proportional to edge-weight (or 0 otherwise)

Model

$$P(u,v) = \sigma(\mathbf{f}(u)^T \mathbf{f}(v))$$

Sigmoid of embedding dot product

Higher-Order Methods

Large-Scale **I**nformation **N**etwork **E**mbedding (LINE) [WWW ‘15, Tang et. al.]

- Objective function to minimize KL-divergence from empirical distribution to model-base prediction

$$O_1 \propto \sum_{(u,v)\in\mathbf{E}} w_{u,v} \lg P(u,v)$$

Higher-Order Methods

Large-Scale **I**nformation **N**etwork **E**mbedding (LINE) [WWW ‘15, Tang et. al.]

- **Second-Order proximity**: Define a directed graph over **V** where edge weights represent neighborhood similarity of nodes (e.g, jaccard between two nodes neighbors)
- Use a secondary embedding, **f’**, for embedding a “context” node (similar to word2vec)

Empirical

$$\hat{P}(v|u) = \frac{w_{u,v}}{\sum_{v'} w_{u,v'}}$$

Model

$$P(v|u) = \frac{\exp(\mathbf{f}(u)^T \mathbf{f}'(v))}{\sum_{v'} \exp(\mathbf{f}(u)^T \mathbf{f}'(v'))}$$

Higher-Order Methods

Large-Scale **I**nformation **N**etwork **E**mbedding (LINE) [WWW ‘15, Tang et. al.]

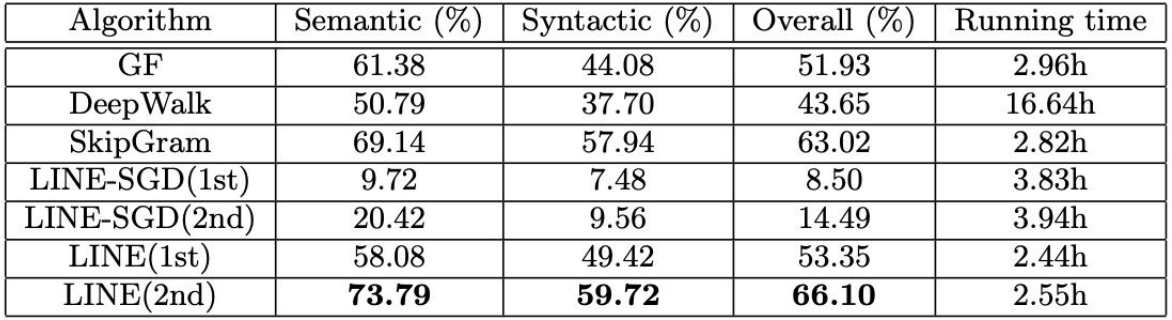
- Define a KL-divergence loss from the empirical second-order proximity distribution to the model-based one
- **NOTE**: Denominator of model-based term involves intractable summation

$$O_2 \propto \sum_{u,v} w_{u,v} \lg P(v|u)$$

Higher-Order Methods

Large-Scale **I**nformation **N**etwork **E**mbedding (LINE) [WWW ‘15, Tang et. al.]

- *Negative sampling* (like node2vec) to sample “negative” edges for model-based term denominator.
- Learn embeddings for O_1 and O_2 independently and concatenate
- Rather than SGD with raw edge weights, sample edges w/ Walker Alias method
- Experiments on text networks (co-occurring terms) in Wikipedia analogy
  a. 2nd order helps



Algorithm	Semantic (%)	Syntactic (%)	Overall (%)	Running time
GF	61.38	44.08	51.93	2.96h
DeepWalk	50.79	37.70	43.65	16.64h
SkipGram	69.14	57.94	63.02	2.82h
LINE-SGD(1st)	9.72	7.48	8.50	3.83h
LINE-SGD(2nd)	20.42	9.56	14.49	3.94h
LINE(1st)	58.08	49.42	53.35	2.44h
LINE(2nd)	**73.79**	**59.72**	**66.10**	2.55h

Higher-Order Methods

GraRep [WWW ‘15, Cao et. al.]

- Can represent a single-step dynamics of a graph walk starting from *u* using matrix algebra:

Normalized transition probs

$$A\mathbf{1}_u$$

One-hot vector on node *u*

- Similarly, can represent probability *k*-step walk starting from *u* will end at node *v* by iterative matrix multiplication

$$P_k(v|u) = (A^k)_{u,v}$$

Higher-Order Methods

GraRep [WWW ‘15, Cao et. al.]

- Similar to LINE, formulate “empirical” and “model” quantities to represent transition probabilities for *u* → *v* for a *k*-step uniform random walk. Use a separate source-destination embedding table (**f** and **f’**):

Empirical

$$\hat{P}_k(v|u) \propto (A^k)_{u,v}$$

Model

$$P_k(v|u) = \frac{\exp(\mathbf{f}(u)^T \mathbf{f}'(v))}{\sum_{v'} \exp(\mathbf{f}(u)^T \mathbf{f}'(v'))}$$

Higher-Order Methods

GraRep [WWW ‘15, Cao et. al.]

- Define loss over KL-divergence between “empirical” *k*-step transition probability and model-defined. Using negative sampling to approximate model denominator (ala node2vec), and skipping some math

$$\ell_k(v|u) = A^k_{u,v} \lg \sigma(\mathbf{f}'(v)^T \mathbf{f}(u)) + \beta \sum_{v' \in D(u)} A^k_{u,v'} \lg \sigma(-\mathbf{f}'(v')^T \mathbf{f}(u))$$

Constant involving negative sampling and number vertices

Higher-Order Methods

GraRep [WWW ‘15, Cao et. al.]

- Differentiating wrt **f’**(v)^T***f***(u) and setting to 0, we obtain

$$\mathbf{f}'(v)^T\mathbf{f}(u) = \lg \frac{A^k_{u,v}}{\sum_{v'} A^k_{u,v'}} - \lg \beta$$

- Equivalent to the matrix factorization problem ***A**** **= (F’)^T F**
  a. **F** and **F’** are matrices where rows are node embeddings
  b. ***A**** represents matrix of right-hand-side expression

Higher-Order Methods

GraRep [WWW ‘15, Cao et. al.]

- Similar to GLOVE where word embeddings becomes matrix-factorization
  a. Similar pro/cons versus SkipGram word embeddings in terms of memory vs compute trade-offs
- Compute representations for different *k* lengths and concatenate

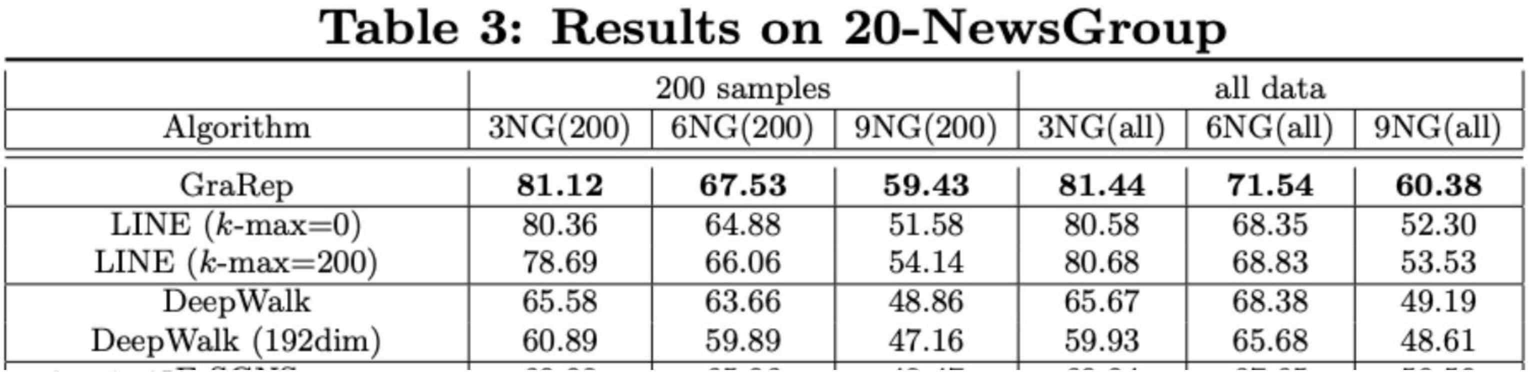


Table 3: Results on 20-NewsGroup

	200 samples			all data		
Algorithm	3NG(200)	6NG(200)	9NG(200)	3NG(all)	6NG(all)	9NG(all)
GraRep	**81.12**	**67.53**	**59.43**	**81.44**	**71.54**	**60.38**
LINE (k-max=0)	80.36	64.88	51.58	80.58	68.35	52.30
LINE (k-max=200)	78.69	66.06	54.14	80.68	68.83	53.53
DeepWalk	65.58	63.66	48.86	65.67	68.38	49.19
DeepWalk (192dim)	60.89	59.89	47.16	59.93	65.68	48.61

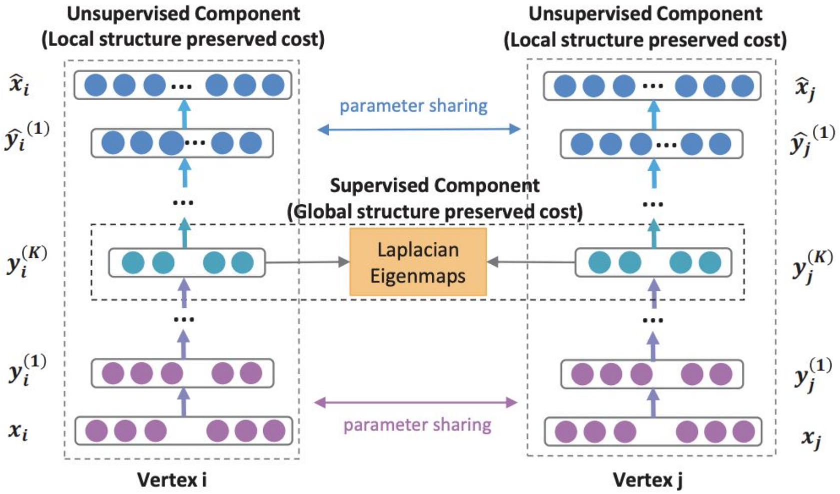
Higher-Order Methods

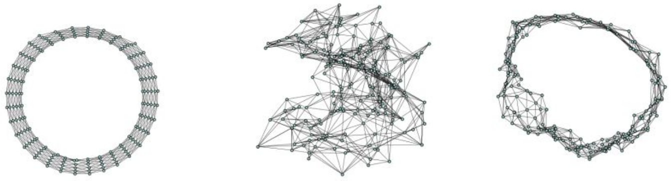
Recap

- Higher-order methods take “observed” graph properties (proximity structure or transition probabilities) and fit node embeddings as part of a model to match empirical properties
- Different methods encode different graph properties, but we see consistent value in encoding non-local structure.

Some Other Things To Check Out

- Structural Deep Network Embedding (**SDNE**)
  a. [KDD '16, Wang et. al.]
  b. Jointly learn first- and second-order proximity at different auto-encoder layers
- Hierarchical Representation Learning For Networks (**HARP**)
  a. [AAI '18, Chen et. al]
  b. Embed sequence of "coarser" graphs and "warm start" finer grained graph embedding





(a) Can_187 (b) LINE (c) HARP

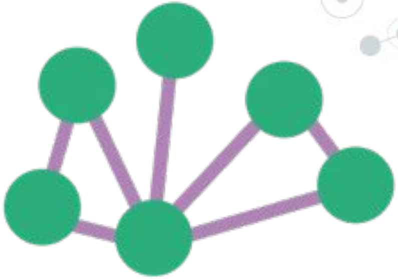
Heterogeneous Graph Representation

Ahmed El-Kishky

Homogeneous vs Heterogeneous Graphs

Homogeneous Graphs

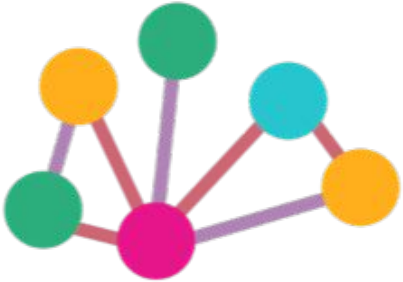
- Single node type and single edge type
- Twitter
  a. users *follow* other users



homogeneous

Heterogeneous Graphs

- Multiple node and/or edge types
- Twitter:
  - users *follow* other users
  - users *fave* *tweets*
  - users *reply_to* *tweets*



heterogeneous

Heterogeneous Graphs

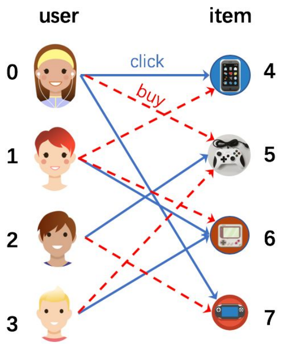
A heterogeneous graph is defined as:

$$\mathbf{G = (V, E, R, T)}$$

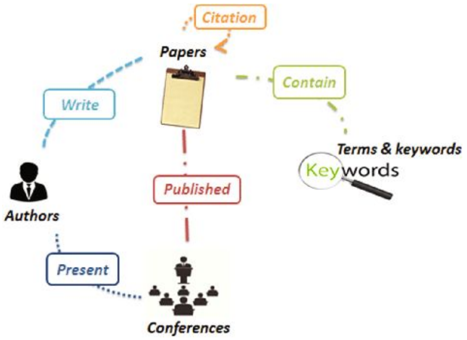
- Nodes with node types $v_i \in \mathbf{V}$
- Edges with relation types $(v_i , r, v_{\square}) \in \mathbf{E}$
- Node type $T(v_i)$
- Relation type $r \in \mathbf{R}$

Heterogeneous Graphs in the Wild

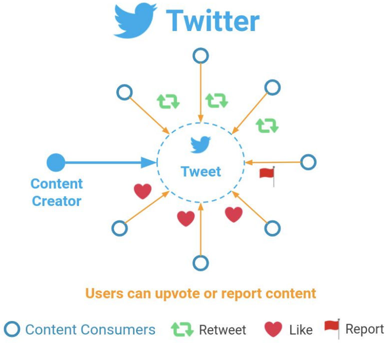
- Social Networks (e.g., Twitter, Facebook)
- Bibliographic networks (e.g., DBLP, ArXiv, Pubmed)
- User-Item Engagement (e.g., e-Commerce, search engines)
- World Wide Web
- Biological networks



e-Commerce



bibliographic

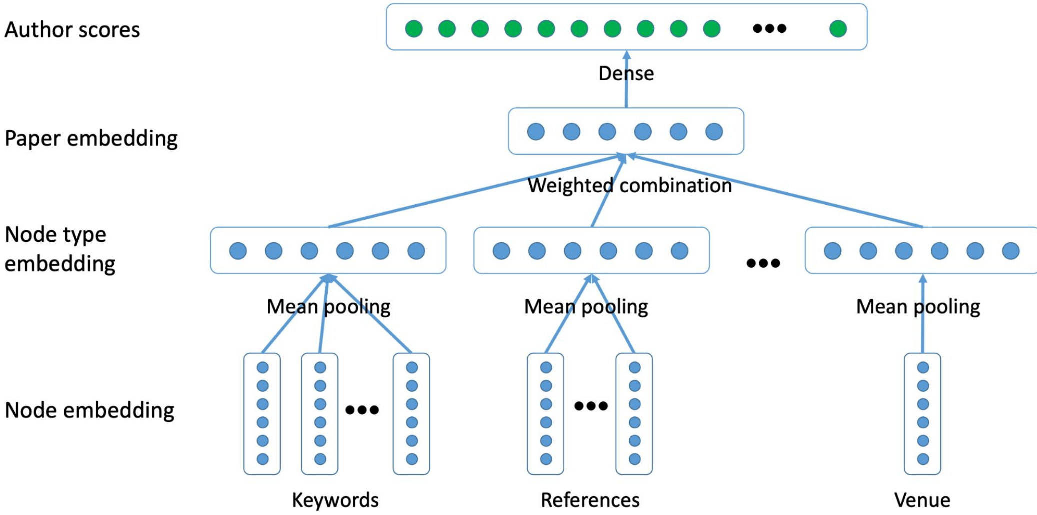


social

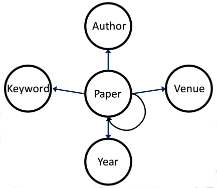
Heterogeneous Information Network Embeddings

Heterogeneous Star Network Embedding

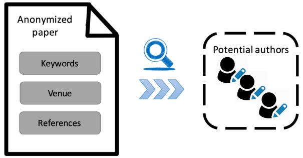
- Star-schema network
  - Papers, keywords, authors, venues
- Embed the center node type
  - Learn paper representation
- Predict authors for anonymized papers
  - Dot (author-emb, paper-emb)



Star-schema bibliographic network

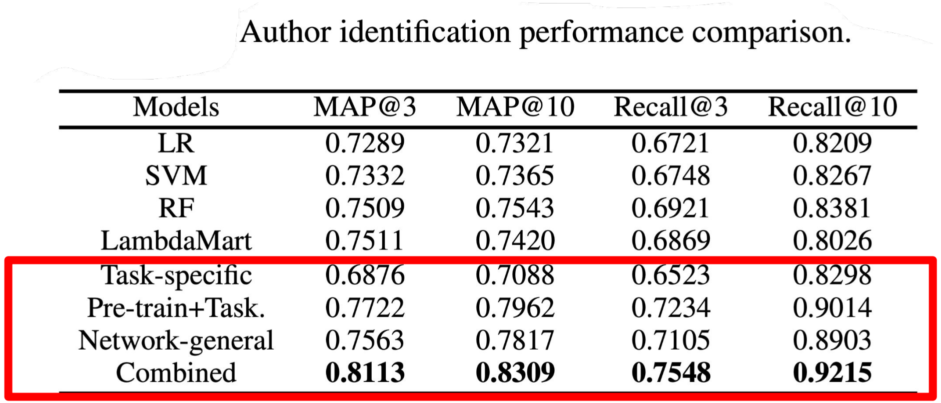


Author identification problem



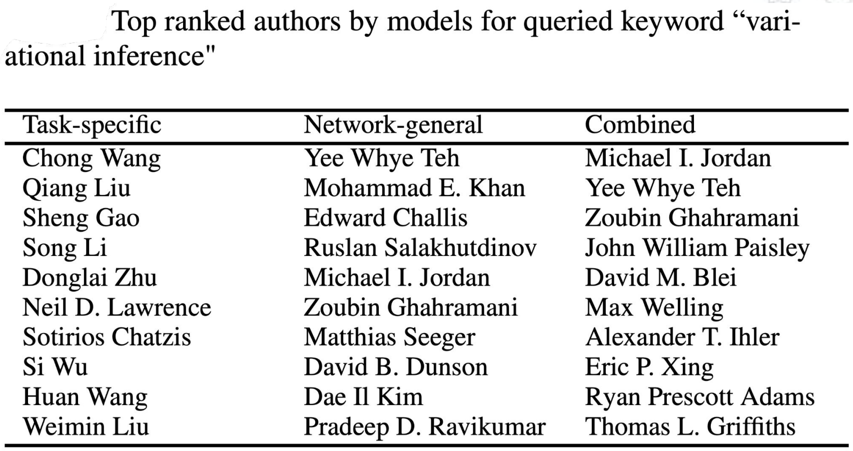
Paper: [Task-Guided and Path-Augmented Heterogeneous Network Embedding for Author Identification]()

Heterogeneous Star Network Embedding



Author identification performance comparison.

Models	MAP@3	MAP@10	Recall@3	Recall@10
LR	0.7289	0.7321	0.6721	0.8209
SVM	0.7332	0.7365	0.6748	0.8267
RF	0.7509	0.7543	0.6921	0.8381
LambdaMart	0.7511	0.7420	0.6869	0.8026
Task-specific	0.6876	0.7088	0.6523	0.8298
Pre-train+Task.	0.7722	0.7962	0.7234	0.9014
Network-general	0.7563	0.7817	0.7105	0.8903
Combined	**0.8113**	**0.8309**	**0.7548**	**0.9215**



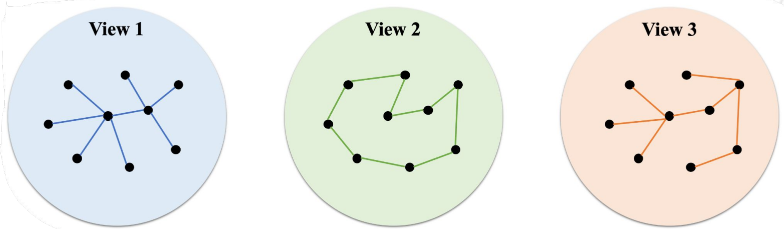
Top ranked authors by models for queried keyword "variational inference"

Task-specific	Network-general	Combined
Chong Wang	Yee Whye Teh	Michael I. Jordan
Qiang Liu	Mohammad E. Khan	Yee Whye Teh
Sheng Gao	Edward Challis	Zoubin Ghahramani
Song Li	Ruslan Salakhutdinov	John William Paisley
Donglai Zhu	Michael I. Jordan	David M. Blei
Neil D. Lawrence	Zoubin Ghahramani	Max Welling
Sotirios Chatzis	Matthias Seeger	Alexander T. Ihler
Si Wu	David B. Dunson	Eric P. Xing
Huan Wang	Dae Il Kim	Ryan Prescott Adams
Weimin Liu	Pradeep D. Ravikumar	Thomas L. Griffiths

Paper: Task-Guided and Path-Augmented Heterogeneous Network Embedding for Author Identification

Multi-view Network Embedding

- Real-world graphs have many edge types between nodes.
- Multiple relationships means multiple views
  - Each relationship type is a view
  - On Twitter:
    - Users *follow* other users
    - Users *retweet* other users
    - Users *favorite* tweets
    - Users *reply* to tweets

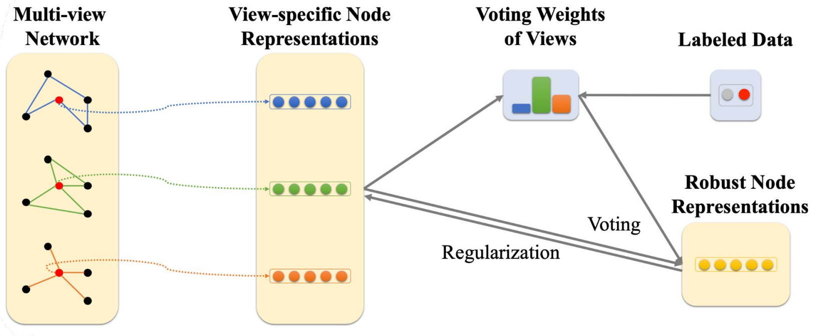


An example multi-view network with three views. Each view corresponds to a type of proximity between nodes, which is characterized by a set of edges. Different views are complementary to each other.

Paper: An Attention-based Collaboration Framework for Multi-View Network Representation Learning: https://dl.acm.org/doi/pdf/10.1145/3132847.3133021

Multi-view Network Embedding

- Nodes have view-specific embeddings
  - Regularization across views
- Robust embedding from attention across different views' embeddings

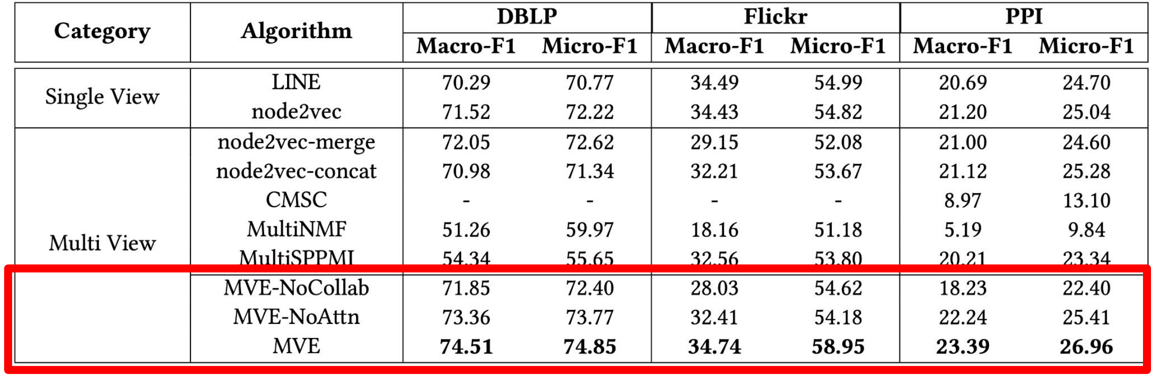


Overview of the proposed approach. The collaboration framework (yellow parts) preserves the node proximities of different views with a set of view-specific node representations, which further vote for the robust representations. During voting, we learn the weights of views through an attention based method (blue parts), which enables nodes to focus on the most informative views.

Paper: An Attention-based Collaboration Framework for Multi-View Network Representation Learning: https://dl.acm.org/doi/pdf/10.1145/3132847.3133021

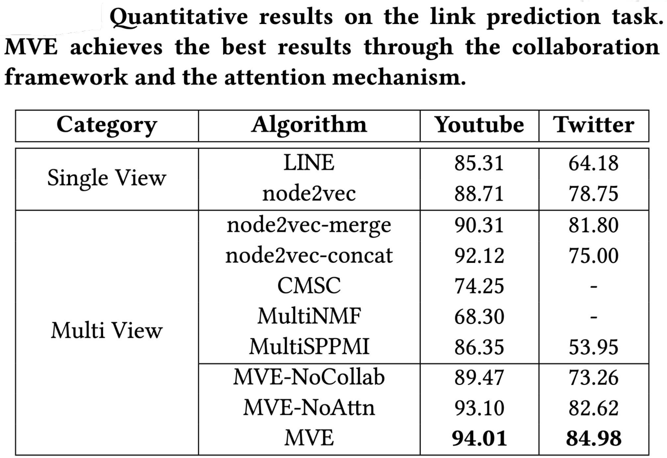
Multi-view Network Embedding

Node classification task



Category	Algorithm	DBLP		Flickr		PPI	
		Macro-F1	Micro-F1	Macro-F1	Micro-F1	Macro-F1	Micro-F1
Single View	LINE	70.29	70.77	34.49	54.99	20.69	24.70
	node2vec	71.52	72.22	34.43	54.82	21.20	25.04
Multi View	node2vec-merge	72.05	72.62	29.15	52.08	21.00	24.60
	node2vec-concat	70.98	71.34	32.21	53.67	21.12	25.28
	CMSC	-	-	-	-	8.97	13.10
	MultiNMF	51.26	59.97	18.16	51.18	5.19	9.84
	MultiSPPMI	54.34	55.65	32.56	53.80	20.21	23.34
	MVE-NoCollab	71.85	72.40	28.03	54.62	18.23	22.40
	MVE-NoAttn	73.36	73.77	32.41	54.18	22.24	25.41
	MVE	**74.51**	**74.85**	**34.74**	**58.95**	**23.39**	**26.96**

Link prediction classification task



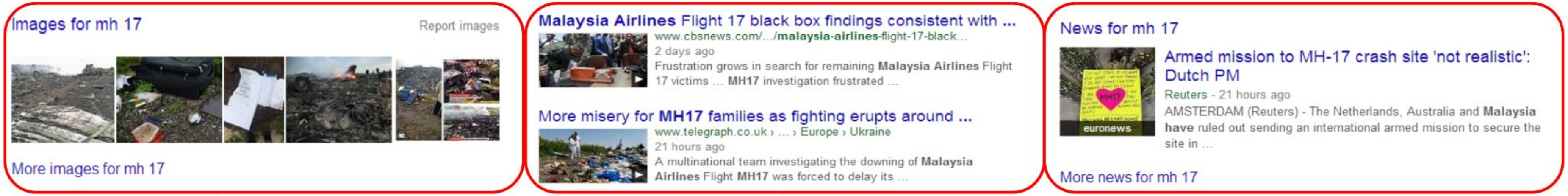
Quantitative results on the link prediction task. MVE achieves the best results through the collaboration framework and the attention mechanism.

Category	Algorithm	Youtube	Twitter
Single View	LINE	85.31	64.18
	node2vec	88.71	78.75
Multi View	node2vec-merge	90.31	81.80
	node2vec-concat	92.12	75.00
	CMSC	74.25	-
	MultiNMF	68.30	-
	MultiSPPMI	86.35	53.95
	MVE-NoCollab	89.47	73.26
	MVE-NoAttn	93.10	82.62
	MVE	**94.01**	**84.98**

Paper: An Attention-based Collaboration Framework for Multi-View Network Representation Learning: https://dl.acm.org/doi/pdf/10.1145/3132847.3133021

Heterogeneous Network Embeddings via Deep Architectures

- Heterogeneous information network consisting of linked text and images
- Objective: Makes the embeddings of linked nodes closer to each other
- Edge Types
  - Image-to-Image
  - Text-to-Image
  - Text-to-Text



Images for mh 17

Report images

More images for mh 17

Malaysia Airlines Flight 17 black box findings consistent with ...
www.cbsnews.com/.../malaysia-airlines-flight-17-black...
2 days ago
Frustration grows in search for remaining Malaysia Airlines Flight 17 victims ... MH17 investigation frustrated ...

More misery for MH17 families as fighting erupts around ...
www.telegraph.co.uk › ... › Europe › Ukraine
21 hours ago
A multinational team investigating the downing of Malaysia Airlines Flight MH17 was forced to delay its ...

News for mh 17

Armed mission to MH-17 crash site 'not realistic': Dutch PM
Reuters - 21 hours ago
AMSTERDAM (Reuters) - The Netherlands, Australia and Malaysia have ruled out sending an international armed mission to secure the site in ...

More news for mh 17

Images

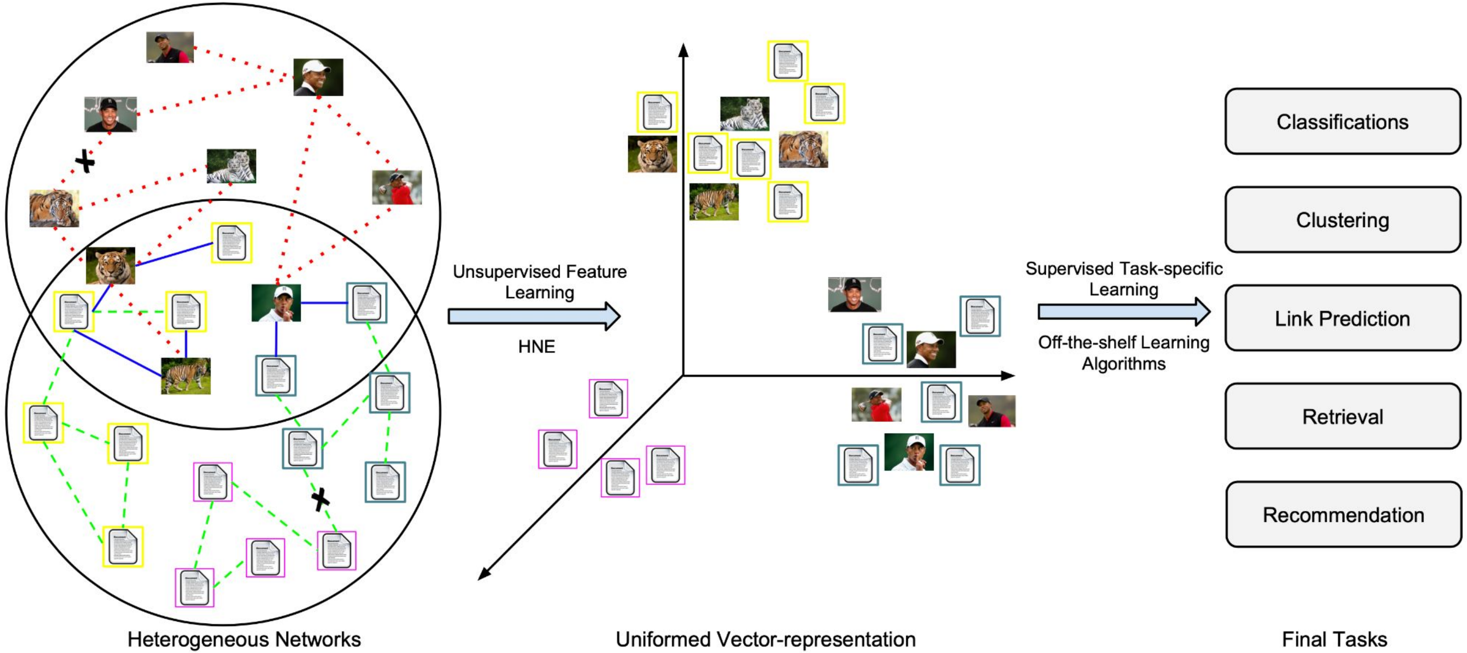
Videos

Text

Illustration of the heterogeneity of different data sources describing the same topic “MH 17”.

Paper: Heterogeneous Network Embeddings with Deep Architectures: https://dl.acm.org/doi/10.1145/2783258.2783296

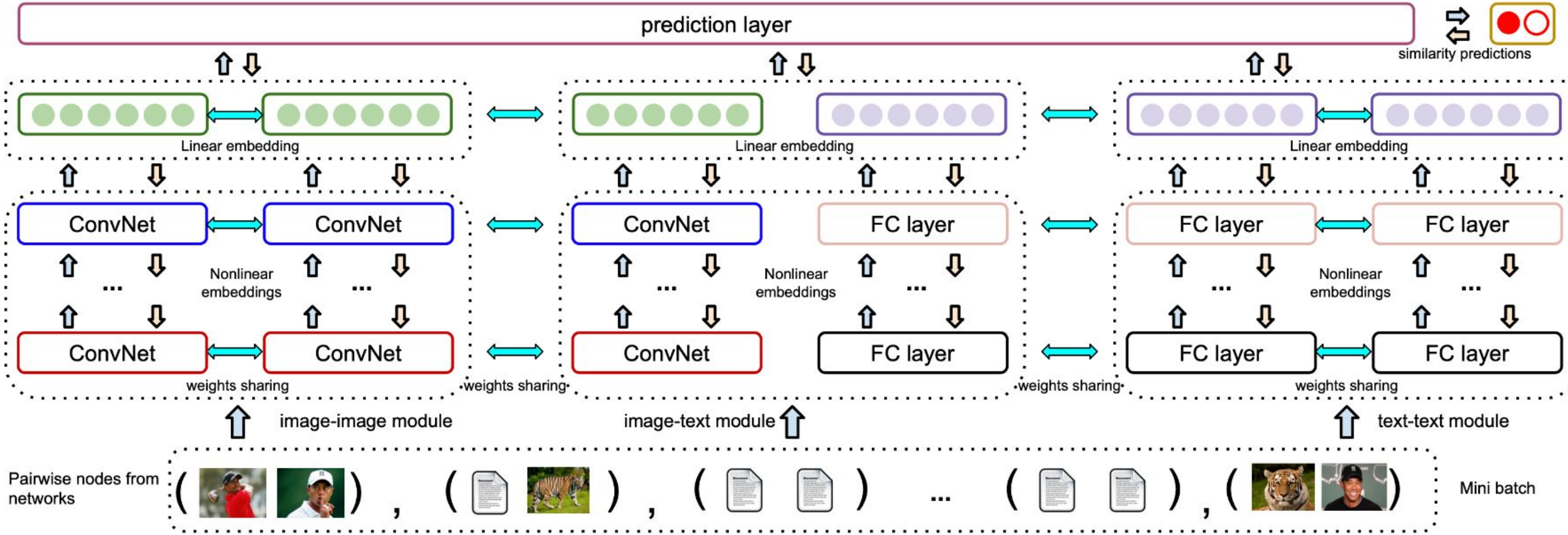
Heterogeneous Network Embeddings via Deep Architectures


Unsupervised Feature Learning
HNE
Supervised Task-specific Learning
Off-the-shelf Learning Algorithms
Classifications
Clustering
Link Prediction
Retrieval
Recommendation
Heterogeneous Networks
Uniformed Vector-representation
Final Tasks

The flowchart of the proposed Heterogeneous Network Embedding (HNE) framework.

Paper: Heterogeneous Network Embeddings with Deep Architectures

Heterogeneous Network Embeddings via Deep Architectures

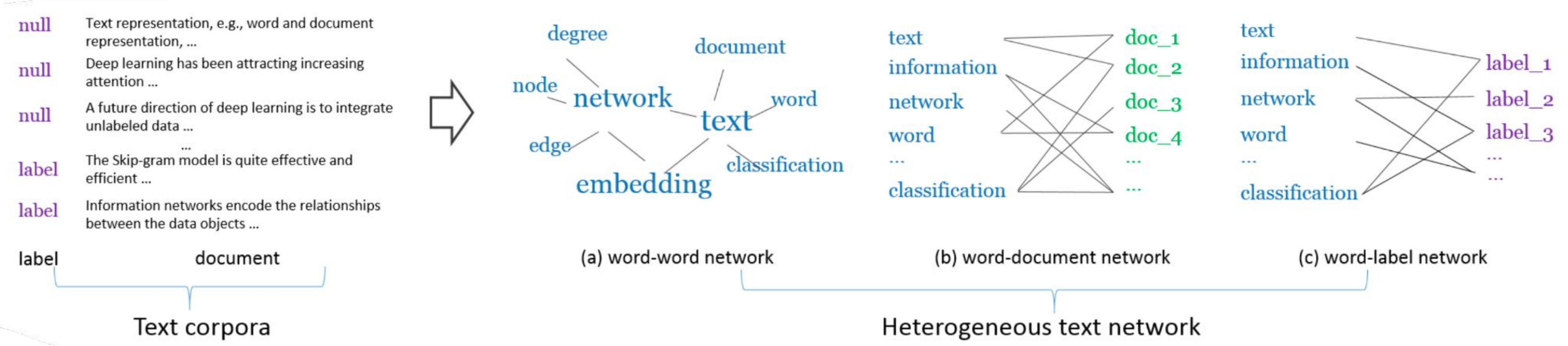


The overall architecture of HNE . The same color indicates the shared weights. The arrows are directions of forward feeding and back propagation.

Paper: Heterogeneous Network Embeddings with Deep Architectures

PTE: Predictive Text Embeddings via Large-scale Heterogeneous Text Networks

- Takes an unstructured text corpus and transforms into a heterogeneous text network
  - word-to-word, word-to-document, document-to-label edges
- Embed nodes of induced heterogeneous information network

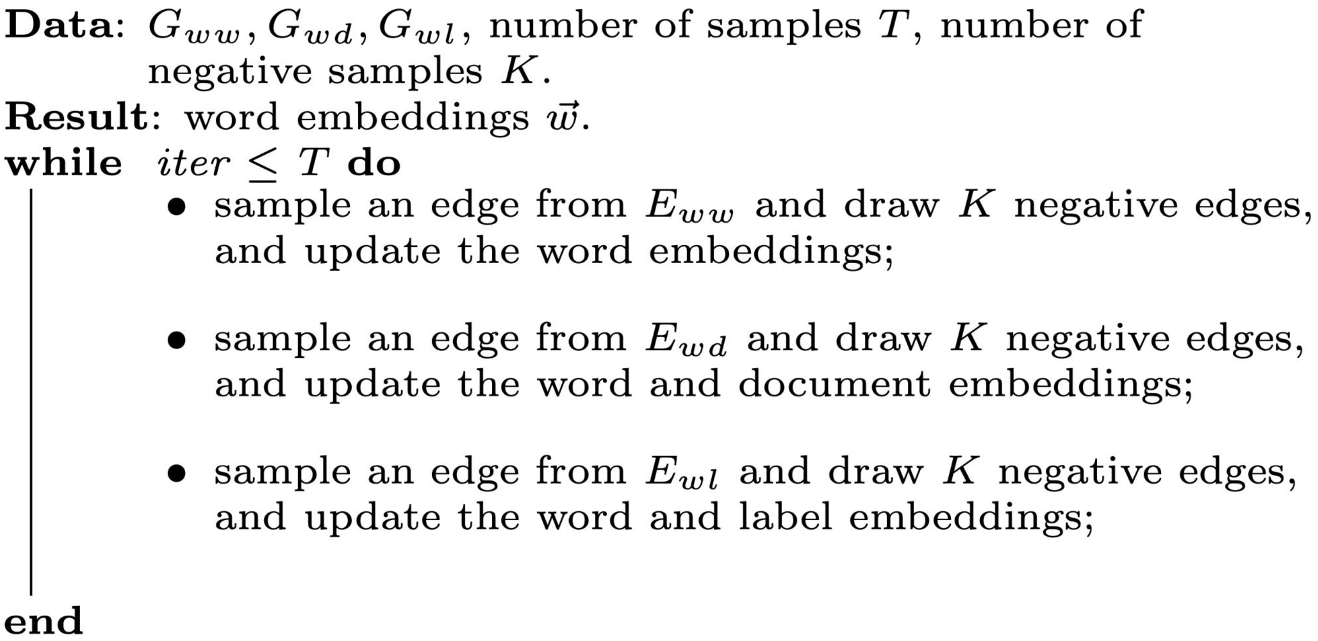


label	document
null	Text representation, e.g., word and document representation, ...
null	Deep learning has been attracting increasing attention ...
null	A future direction of deep learning is to integrate unlabeled data ...
	...
label	The Skip-gram model is quite effective and efficient ...
label	Information networks encode the relationships between the data objects ...

Illustration of converting a partially labeled text corpora to a heterogeneous text network. The word-word co-occurrence network and word-document network encode the unsupervised information, capturing the local context-level and document-level word co-occurrences respectively; the word-label network encodes the supervised information, capturing the class-level word co-occurrences.

Paper: Predictive Text Embeddings via Large-scale Heterogeneous Text Networks

PTE: Predictive Text Embeddings via Large-scale Heterogeneous Text Networks

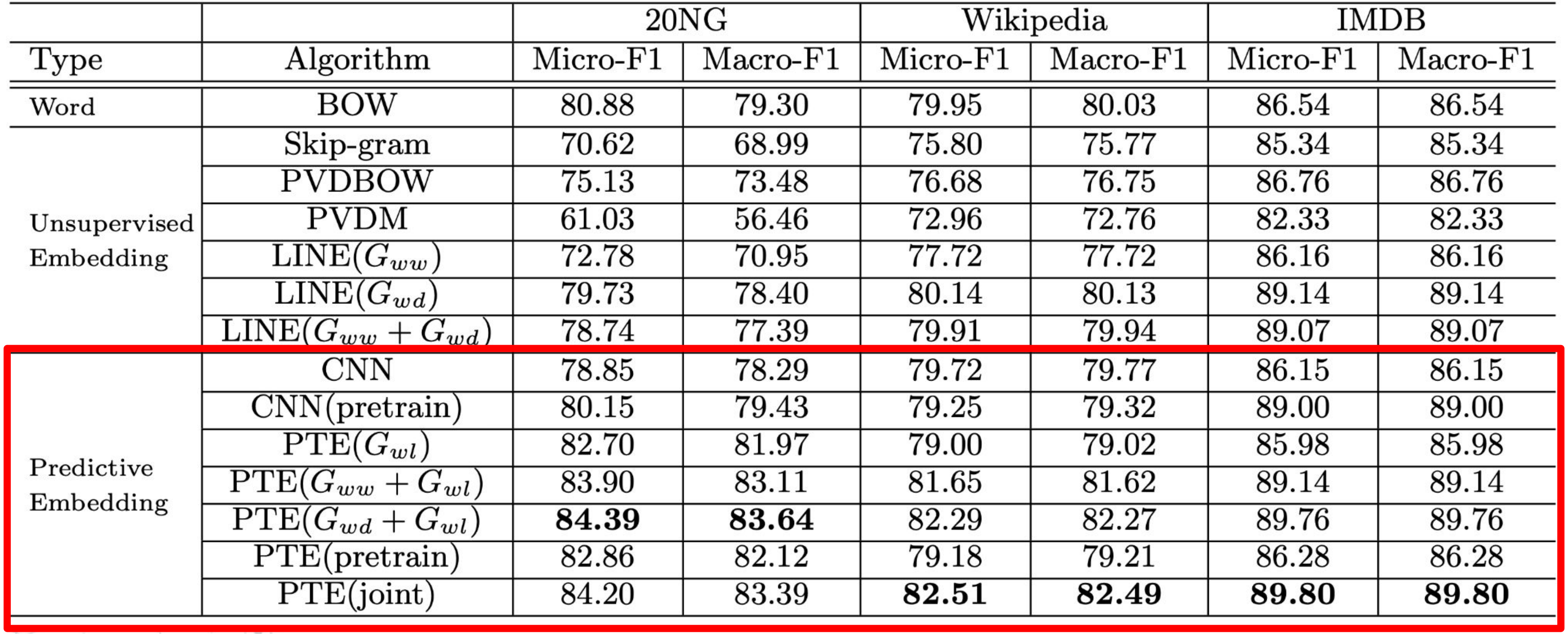


```
Data: $G_{ww}, G_{wd}, G_{wl}$, number of samples $T$, number of
      negative samples $K$.
Result: word embeddings $\vec{w}$.
while $iter \leq T$ do
    • sample an edge from $E_{ww}$ and draw $K$ negative edges,
      and update the word embeddings;

    • sample an edge from $E_{wd}$ and draw $K$ negative edges,
      and update the word and document embeddings;

    • sample an edge from $E_{wl}$ and draw $K$ negative edges,
      and update the word and label embeddings;

end
```

Paper: Predictive Text Embeddings via Large-scale Heterogeneous Text Networks

PTE: Predictive Text Embeddings via Large-scale Heterogeneous Text Networks

Long Document Text Classification

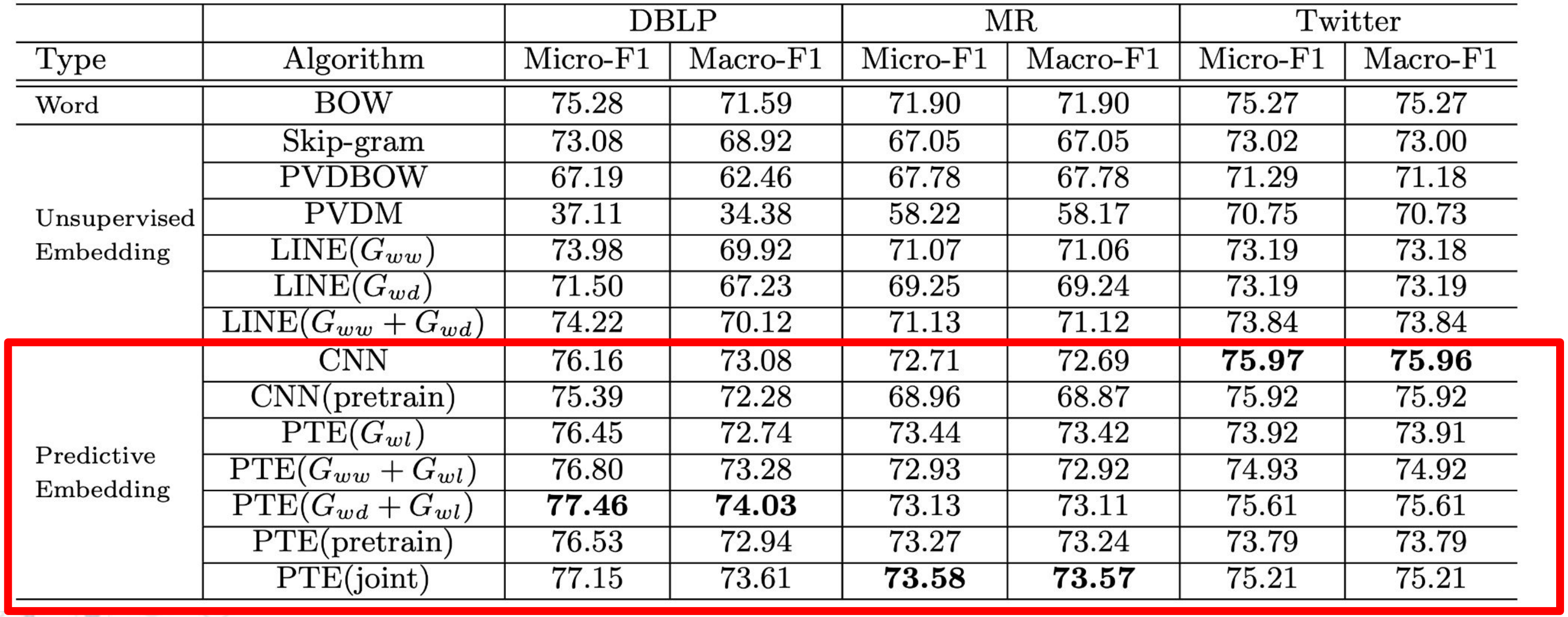


		20NG		Wikipedia		IMDB	
Type	Algorithm	Micro-F1	Macro-F1	Micro-F1	Macro-F1	Micro-F1	Macro-F1
Word	BOW	80.88	79.30	79.95	80.03	86.54	86.54
Unsupervised Embedding	Skip-gram	70.62	68.99	75.80	75.77	85.34	85.34
	PVDBOW	75.13	73.48	76.68	76.75	86.76	86.76
	PVDM	61.03	56.46	72.96	72.76	82.33	82.33
	LINE(G_{ww})	72.78	70.95	77.72	77.72	86.16	86.16
	LINE(G_{wd})	79.73	78.40	80.14	80.13	89.14	89.14
	LINE($G_{ww} + G_{wd}$)	78.74	77.39	79.91	79.94	89.07	89.07
Predictive Embedding	CNN	78.85	78.29	79.72	79.77	86.15	86.15
	CNN(pretrain)	80.15	79.43	79.25	79.32	89.00	89.00
	PTE(G_{wl})	82.70	81.97	79.00	79.02	85.98	85.98
	PTE($G_{ww} + G_{wl}$)	83.90	83.11	81.65	81.62	89.14	89.14
	PTE($G_{wd} + G_{wl}$)	**84.39**	**83.64**	82.29	82.27	89.76	89.76
	PTE(pretrain)	82.86	82.12	79.18	79.21	86.28	86.28
	PTE(joint)	84.20	83.39	**82.51**	**82.49**	**89.80**	**89.80**

Paper: Predictive Text Embeddings via Large-scale Heterogeneous Text Networks

PTE: Predictive Text Embeddings via Large-scale Heterogeneous Text Networks

Short Document Text Classification

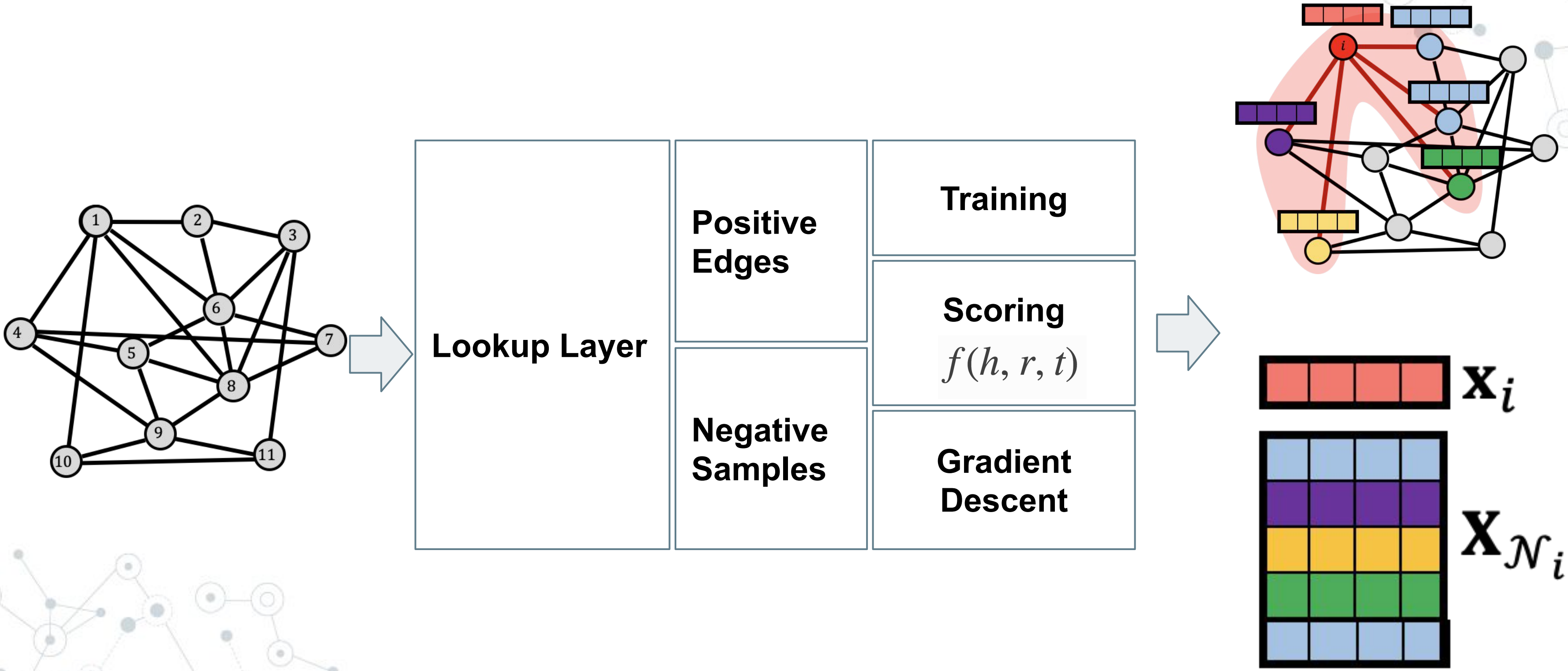


		DBLP		MR		Twitter	
Type	Algorithm	Micro-F1	Macro-F1	Micro-F1	Macro-F1	Micro-F1	Macro-F1
Word	BOW	75.28	71.59	71.90	71.90	75.27	75.27
Unsupervised Embedding	Skip-gram	73.08	68.92	67.05	67.05	73.02	73.00
	PVDBOW	67.19	62.46	67.78	67.78	71.29	71.18
	PVDM	37.11	34.38	58.22	58.17	70.75	70.73
	LINE(G_{ww})	73.98	69.92	71.07	71.06	73.19	73.18
	LINE(G_{wd})	71.50	67.23	69.25	69.24	73.19	73.19
	LINE($G_{ww} + G_{wd}$)	74.22	70.12	71.13	71.12	73.84	73.84
Predictive Embedding	CNN	76.16	73.08	72.71	72.69	**75.97**	**75.96**
	CNN(pretrain)	75.39	72.28	68.96	68.87	75.92	75.92
	PTE(G_{wl})	76.45	72.74	73.44	73.42	73.92	73.91
	PTE($G_{ww} + G_{wl}$)	76.80	73.28	72.93	72.92	74.93	74.92
	PTE($G_{wd} + G_{wl}$)	**77.46**	**74.03**	73.13	73.11	75.61	75.61
	PTE(pretrain)	76.53	72.94	73.27	73.24	73.79	73.79
	PTE(joint)	77.15	73.61	**73.58**	**73.57**	75.21	75.21

Paper: Predictive Text Embeddings via Large-scale Heterogeneous Text Networks

Knowledge Graph Embedding Techniques for Heterogeneous Graph Embeddings

Workflow of Shallow Heterogenous Graph Embedding



Lookup Layer

Positive Edges

Negative Samples

Training

Scoring $f(h, r, t)$

Gradient Descent

Shallow Heterogeneous Graph Embedding (Knowledge Graph Embedding Techniques)

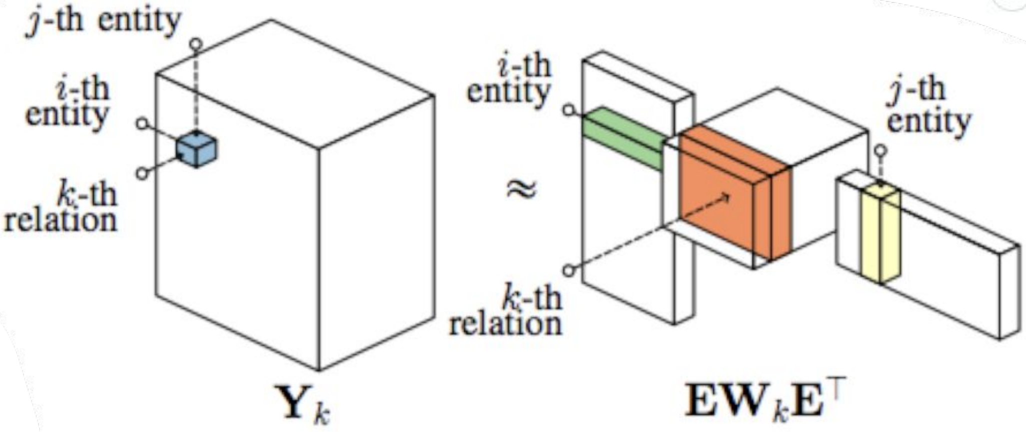
Many knowledge graph embedding (KGE) techniques have been proposed

1. RESCAL (Nickel et al, 2011)
2. TransE (Bordes et al, 2013)
3. Neural Tensor Network (Socher et al, 2013)
4. DistMult (Yang et al, 2015)
5. Complex Embeddings (Trouillon et al, 2016)
6. Quaternion Embeddings (Zhang et al, 2019)

RESCAL: A Three-way Model for Collective Learning on Multi-relational Data

Tensor factorization on the <*head-entity, tail-entity, relation*> tensor

- pairs of entities are represented via the tensor product of their embeddings
- difficult to scale quadratic runtime and memory complexity (embedding dimension)



RESCAL as a tensor factorization of the adjacency tensor $\mathbf{Y}$.

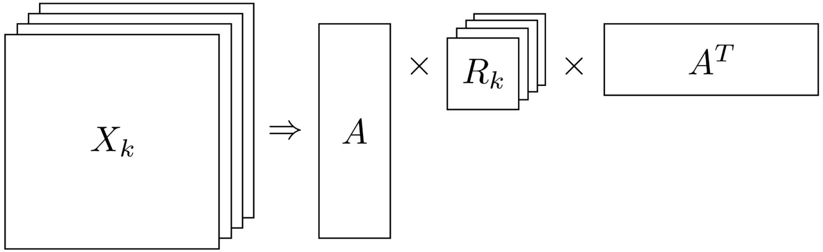
Paper: A Three-Way Model for Collective Learning on Multi-Relational Data

RESCAL: A Three-way Model for Collective Learning on Multi-relational Data

- Tensor factorization on the <*head-entity, tail-entity, relation*> tensor

$$X_k \approx A R_k A^T$$

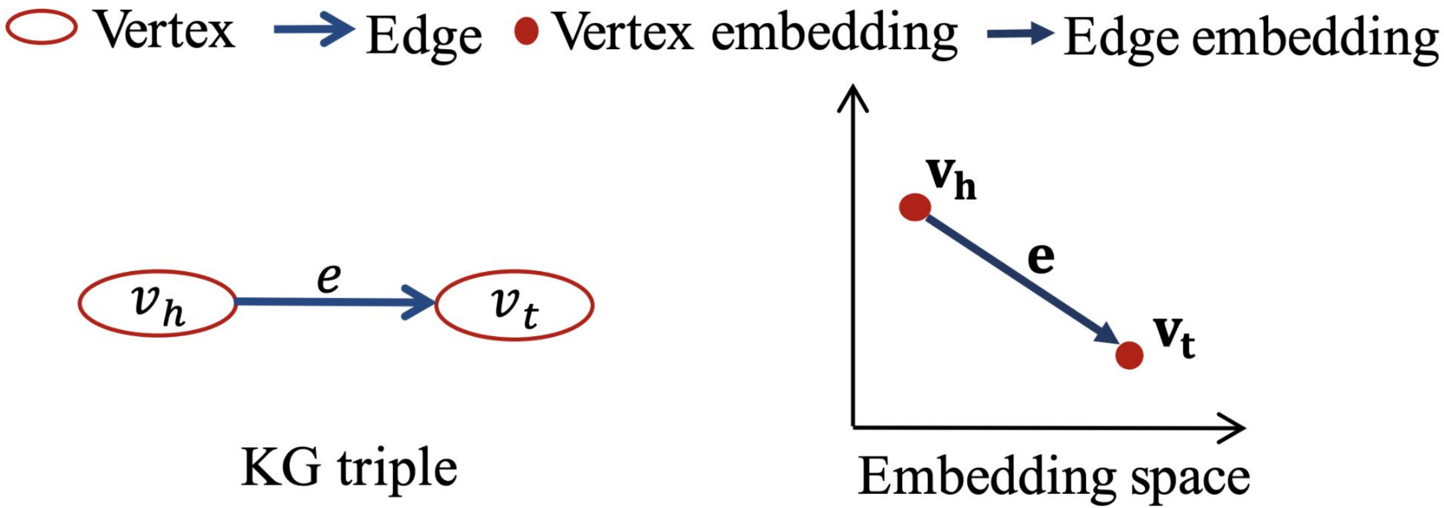
- **A** is a *n* × *r* matrix, representing the global entity-latent-component space
- **R**_k is an asymmetric *r* × *r* matrix that specifies the interaction of the latent components per predicate



Paper: A Three-Way Model for Collective Learning on Multi-Relational Data

TransE for Embedding Heterogeneous Graphs

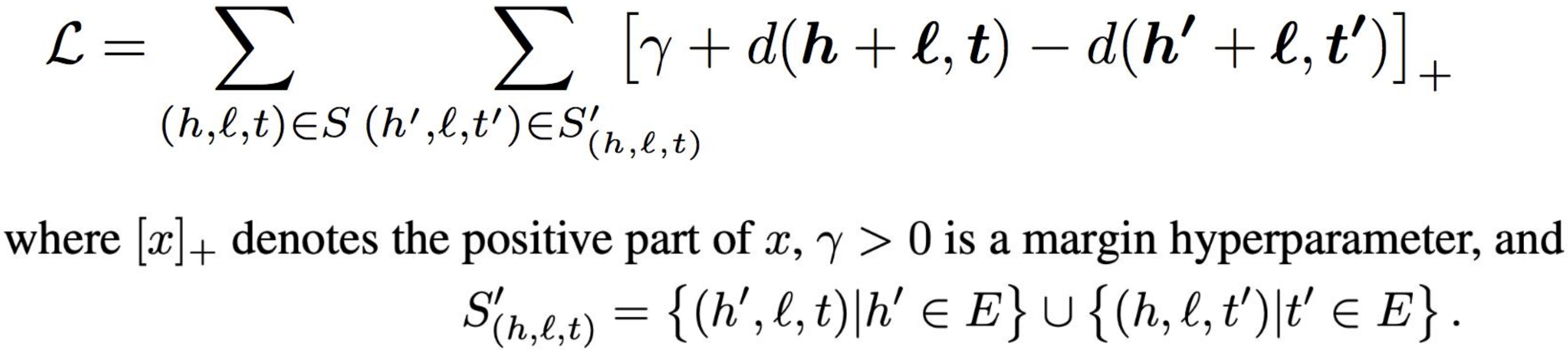
- Translation Embedding (TransE): when adding the relation to the head entity, we should get close to the target tail entity



Paper: [Translating Embeddings for Modeling Multi-relational Data]

TransE for Embedding Heterogeneous Graphs

- Margin based loss function:
  - Minimize the distance between (h+l) and t.
  - Maximize the distance between (h+l) to a randomly sampled tail t’ (negative example).



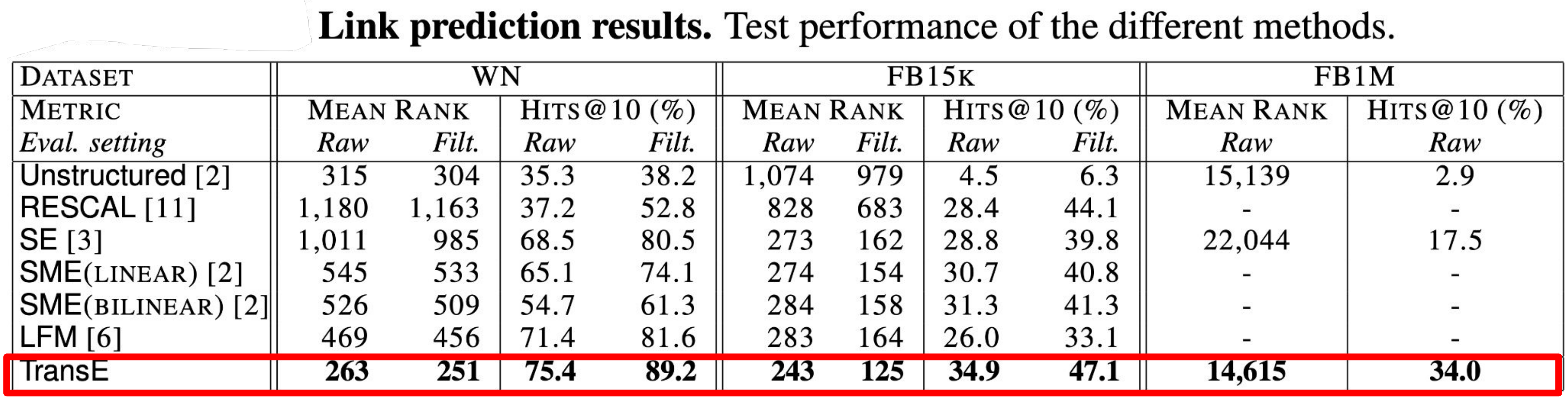
$$\mathcal{L} = \sum_{(h,\ell,t)\in S} \sum_{(h',\ell,t')\in S'_{(h,\ell,t)}} \left[\gamma + d(\boldsymbol{h}+\boldsymbol{\ell},\boldsymbol{t}) - d(\boldsymbol{h'}+\boldsymbol{\ell},\boldsymbol{t'})\right]_+$$

where $[x]_+$ denotes the positive part of x, $\gamma > 0$ is a margin hyperparameter, and

$$S'_{(h,\ell,t)} = \left\{(h',\ell,t)|h' \in E\right\} \cup \left\{(h,\ell,t')|t' \in E\right\}.$$

Paper: Translating Embeddings for Modeling Multi-relational Data

TransE for Embedding Heterogeneous Graphs

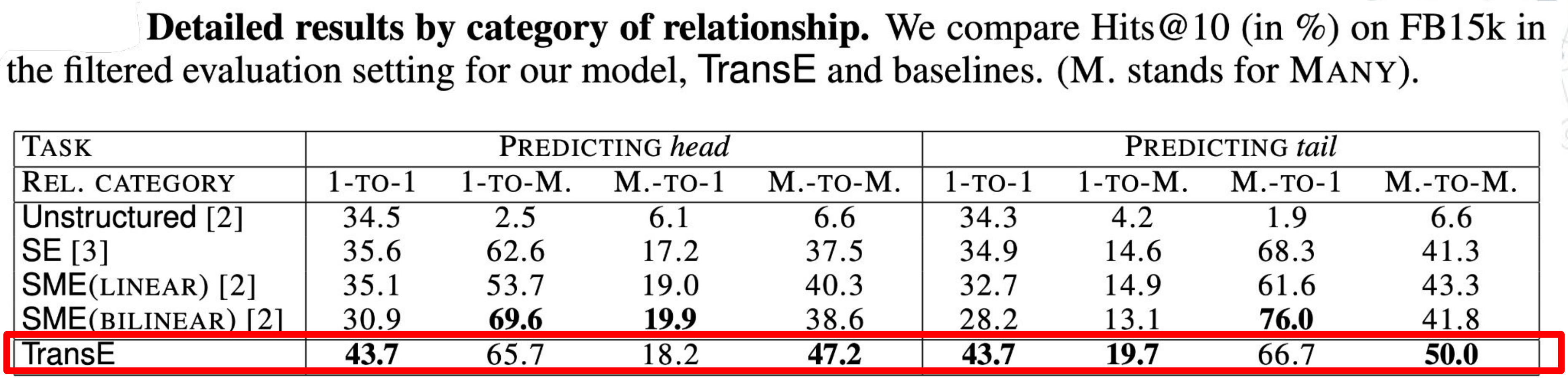


Link prediction results. Test performance of the different methods.

DATASET	WN				FB15K				FB1M	
METRIC	MEAN RANK		HITS@10 (%)		MEAN RANK		HITS@10 (%)		MEAN RANK	HITS@10 (%)
Eval. setting	*Raw*	*Filt.*	*Raw*	*Filt.*	*Raw*	*Filt.*	*Raw*	*Filt.*	*Raw*	*Raw*
Unstructured [2]	315	304	35.3	38.2	1,074	979	4.5	6.3	15,139	2.9
RESCAL [11]	1,180	1,163	37.2	52.8	828	683	28.4	44.1	-	-
SE [3]	1,011	985	68.5	80.5	273	162	28.8	39.8	22,044	17.5
SME(LINEAR) [2]	545	533	65.1	74.1	274	154	30.7	40.8	-	-
SME(BILINEAR) [2]	526	509	54.7	61.3	284	158	31.3	41.3	-	-
LFM [6]	469	456	71.4	81.6	283	164	26.0	33.1	-	-
TransE	**263**	**251**	**75.4**	**89.2**	**243**	**125**	**34.9**	**47.1**	**14,615**	**34.0**

Paper: Translating Embeddings for Modeling Multi-relational Data

TransE for Embedding Heterogeneous Graphs



Detailed results by category of relationship. We compare Hits@10 (in %) on FB15k in the filtered evaluation setting for our model, TransE and baselines. (M. stands for MANY).

TASK	PREDICTING *head*				PREDICTING *tail*			
REL. CATEGORY	1-TO-1	1-TO-M.	M.-TO-1	M.-TO-M.	1-TO-1	1-TO-M.	M.-TO-1	M.-TO-M.
Unstructured [2]	34.5	2.5	6.1	6.6	34.3	4.2	1.9	6.6
SE [3]	35.6	62.6	17.2	37.5	34.9	14.6	68.3	41.3
SME(LINEAR) [2]	35.1	53.7	19.0	40.3	32.7	14.9	61.6	43.3
SME(BILINEAR) [2]	30.9	**69.6**	**19.9**	38.6	28.2	13.1	**76.0**	41.8
TransE	**43.7**	65.7	18.2	**47.2**	**43.7**	**19.7**	66.7	**50.0**

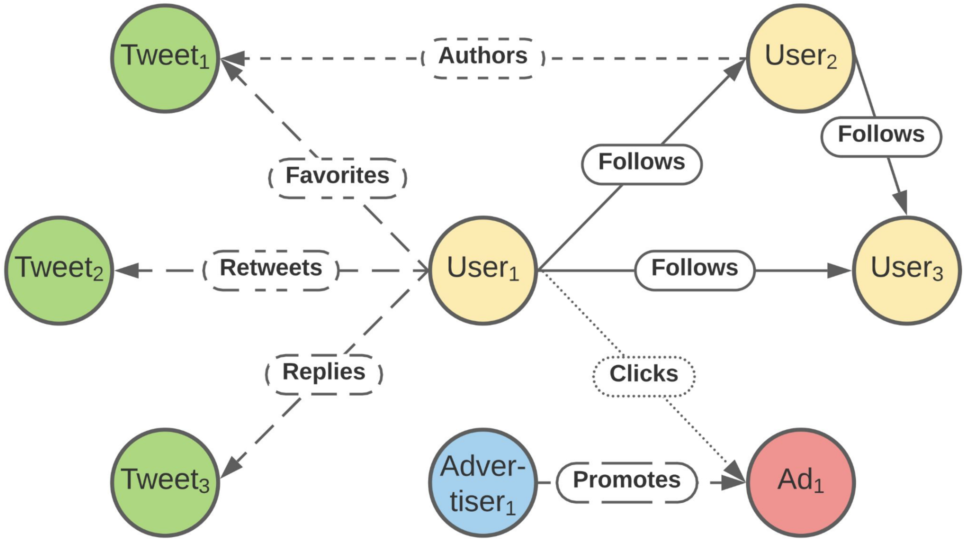
Paper: Translating Embeddings for Modeling Multi-relational Data

Embedding Twitter Heterogeneous Information Network (TwHIN) – TransE in the Wild

◎ As TransE is scalable, it can be used to embed graphs consisting of billions of nodes and hundreds of billions of edges.

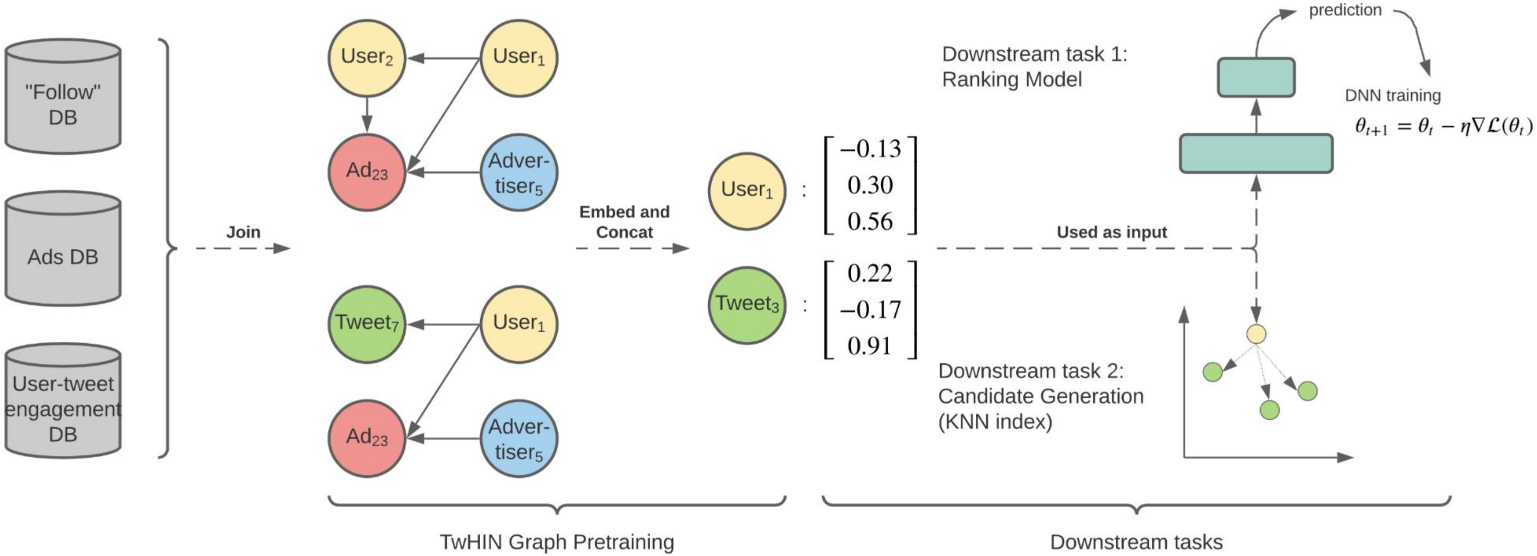
◎ Subsets of nodes, their embeddings, and associated edges are loaded into memory.

◎ TransE training to learn embeddings



Paper: TwHIN: Embedding the Twitter Heterogeneous Information Network for Personalized Recommendation

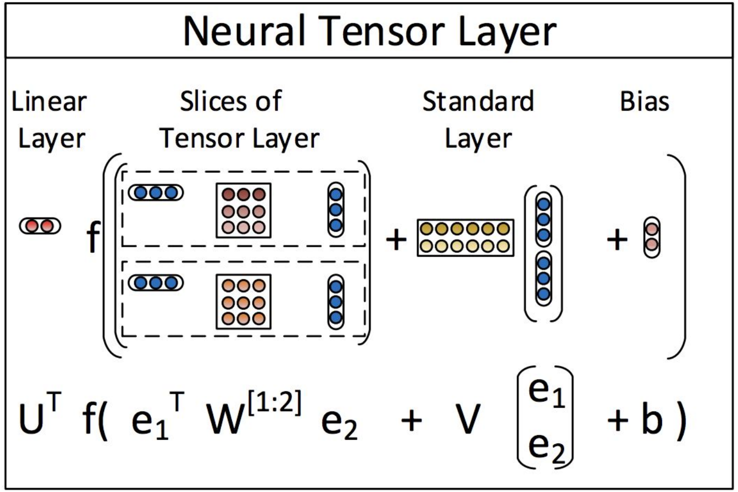
Embedding Twitter Heterogeneous Information Network (TwHIN) – TransE in the Wild

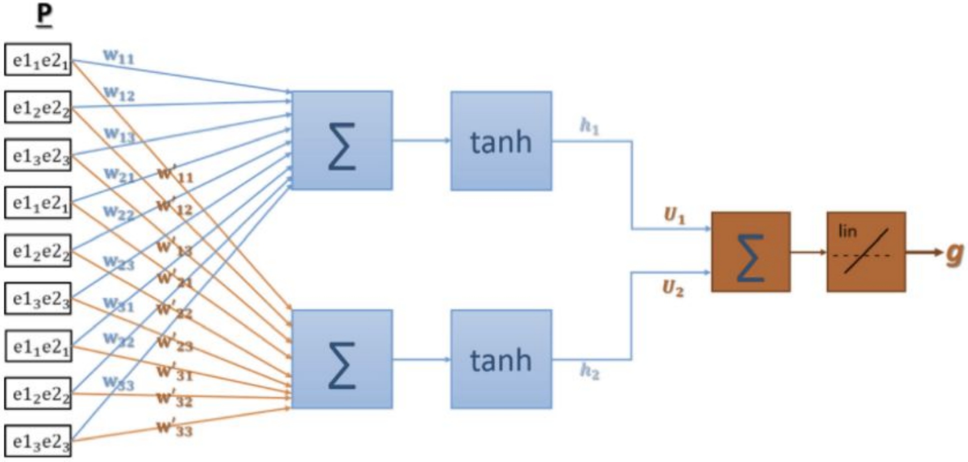

"Follow" DB
Ads DB
User-tweet engagement DB
Join
User$_2$
User$_1$
Ad$_{23}$
Adver-tiser$_5$
Tweet$_7$
User$_1$
Ad$_{23}$
Adver-tiser$_5$
Embed and Concat
User$_1$: $\begin{bmatrix} -0.13 \\ 0.30 \\ 0.56 \end{bmatrix}$
Tweet$_3$: $\begin{bmatrix} 0.22 \\ -0.17 \\ 0.91 \end{bmatrix}$
Used as input
Downstream task 1: Ranking Model
prediction
DNN training
$\theta_{t+1} = \theta_t - \eta \nabla \mathcal{L}(\theta_t)$
Downstream task 2: Candidate Generation (KNN index)
TwHIN Graph Pretraining
Downstream tasks

Paper: TwHIN: Embedding the Twitter Heterogeneous Information Network for Personalized Recommendation

Neural Tensor Networks for Embedding Heterogeneous Graphs

- Model the bilinear interaction between entity pairs using tensors
  - The model computes a score of how likely it is that two entities are in a certain relationship by the following NTN-based function g(e_1, R, e_2):


Neural Tensor Layer
Linear Layer
Slices of Tensor Layer
Standard Layer
Bias
f
+
+
$U^T \ f(\ e_1^T \ W^{[1:2]} \ e_2 \ + \ V \begin{pmatrix} e_1 \\ e_2 \end{pmatrix} + b \ )$



Paper: [Reasoning with Neural Tensor Networks for Knowledge Base Completion]()

Neural Tensor Networks for Embedding Heterogeneous Graphs

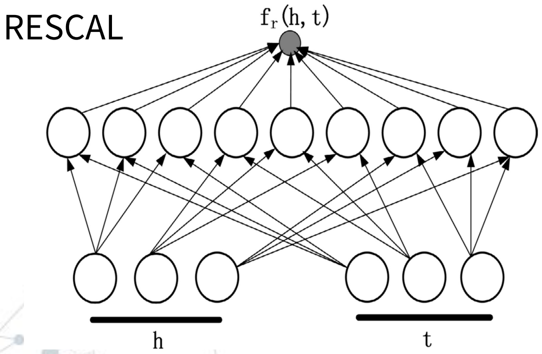
- Training objective: $T^{(i)}_{c} = (e^{(i)}_{1}, R^{(i)}, e_{c})$ is a triplet with a random entity corrupted from a correct triplet $T (i) = (e^{(i)}_{1}, R^{(i)}, e^{(i)}_{2})$
  - Score the correct relation triplet higher than its corrupted one up to a margin of 1.
  - For each correct triplet sample C random corrupted triplets.

$$J(\boldsymbol{\Omega}) = \sum_{i=1}^{N} \sum_{c=1}^{C} \max\left(0, 1 - g\left(T^{(i)}\right) + g\left(T_c^{(i)}\right)\right) + \lambda \|\boldsymbol{\Omega}\|_2^2$$

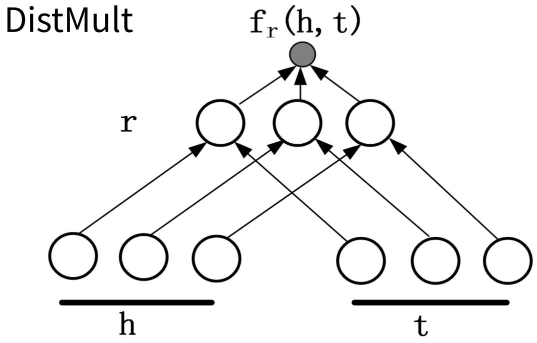
Paper: Reasoning with Neural Tensor Networks for Knowledge Base Completion

DistMult (bilinear-diag): Embedding Entities and Relations for Learning and Inference in Knowledge Bases

- Special case of neural tensor network
  - without nonlinear layer, linear operator, and uses 2-d matrix instead of tensor for the relation
- Bi-linear formulation with diagonal matrix relation
  - same number of parameters as TransE
  - element-wise product between relation embedding and entity embedding



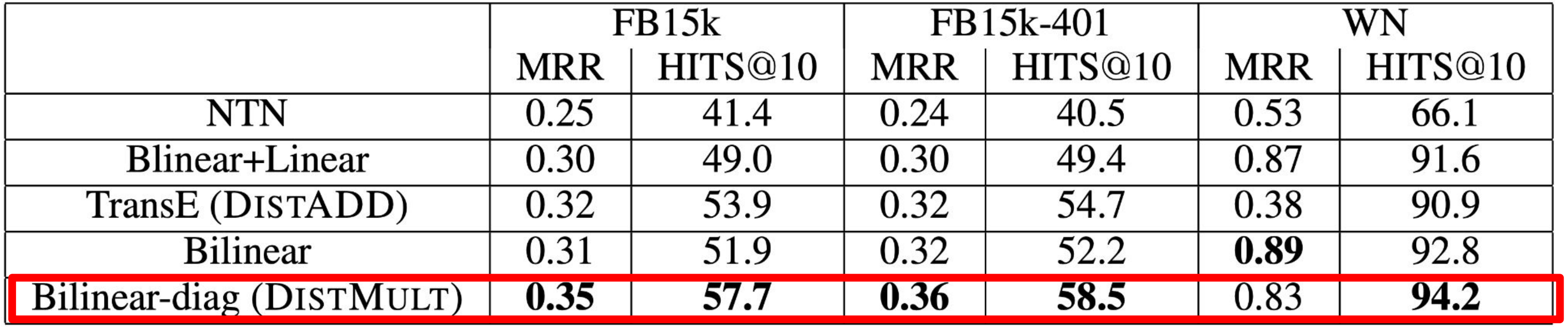
vs



Paper: Embedding Entities and Relations for Learning and Inference in Knowledge Bases

DistMult (bilinear-diag)

Link Prediction Task



	FB15k		FB15k-401		WN	
	MRR	HITS@10	MRR	HITS@10	MRR	HITS@10
NTN	0.25	41.4	0.24	40.5	0.53	66.1
Blinear+Linear	0.30	49.0	0.30	49.4	0.87	91.6
TransE (DISTADD)	0.32	53.9	0.32	54.7	0.38	90.9
Bilinear	0.31	51.9	0.32	52.2	**0.89**	92.8
Bilinear-diag (DISTMULT)	**0.35**	**57.7**	**0.36**	**58.5**	0.83	**94.2**

- Performance increases as complexity of model decreases
- Likely because these graphs are relatively small, so overfitting with complex models

Paper: Embedding Entities and Relations for Learning and Inference in Knowledge Bases

ComplEx Embeddings for Simple Link Prediction

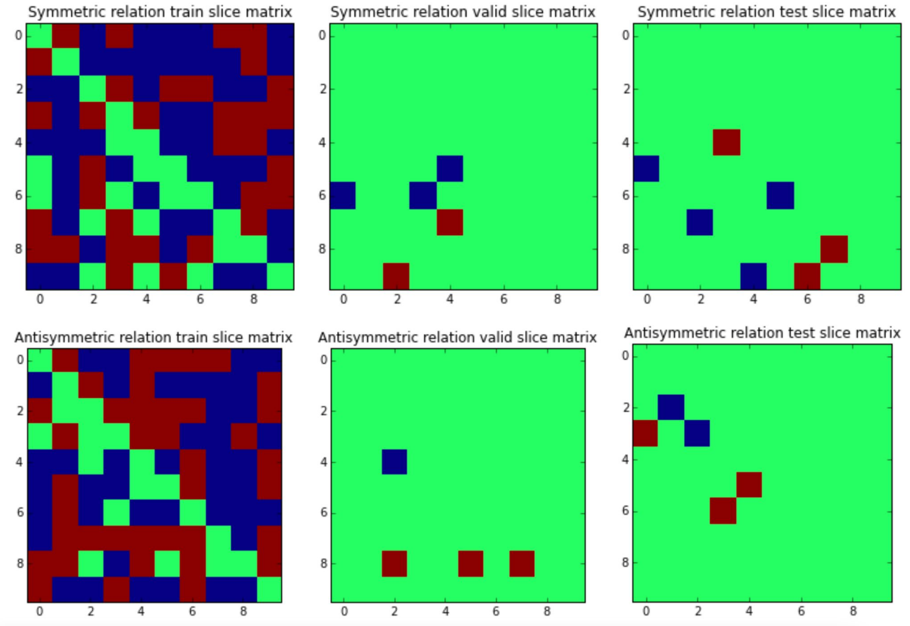
- DistMult Performs dot product in real-space
  - This can’t model anti-symmetric relationships
- ComplEx Embeddings
  - Extends DistMult by performing dot product in Complex space (Hermitian)
  - This can capture anti-symmetric relationships

$$f_{ComplEx} = Re(\langle \mathbf{r}_p, \mathbf{e}_s, \overline{\mathbf{e}_o} \rangle)$$

Paper: Complex Embeddings for Simple Link Prediction

ComplEx Embeddings for Simple Link Prediction

- Visualizing training, validation and test sets exps
  - one symmetric relation
  - one antisymmetric relation
  - Red pixels are positive triples
  - Blue pixels are negatives
  - Green missing ones
- Top: Plots of the symmetric slice (relation) for the 10 first entities
- Bottom: Plots of the antisymmetric slice for the 10 first entities.



Symmetric relation train slice matrix

Symmetric relation valid slice matrix

Symmetric relation test slice matrix

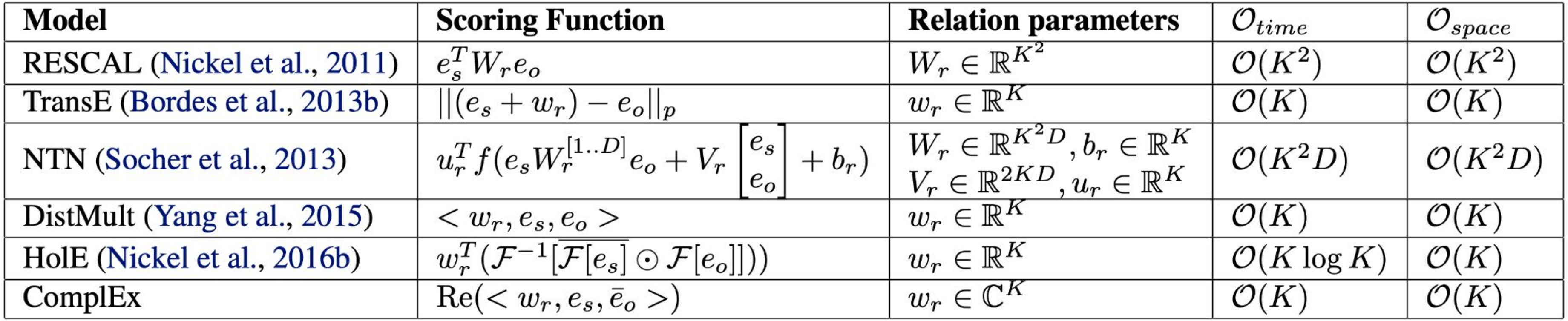
Antisymmetric relation train slice matrix

Antisymmetric relation valid slice matrix

Antisymmetric relation test slice matrix

Paper: Complex Embeddings for Simple Link Prediction

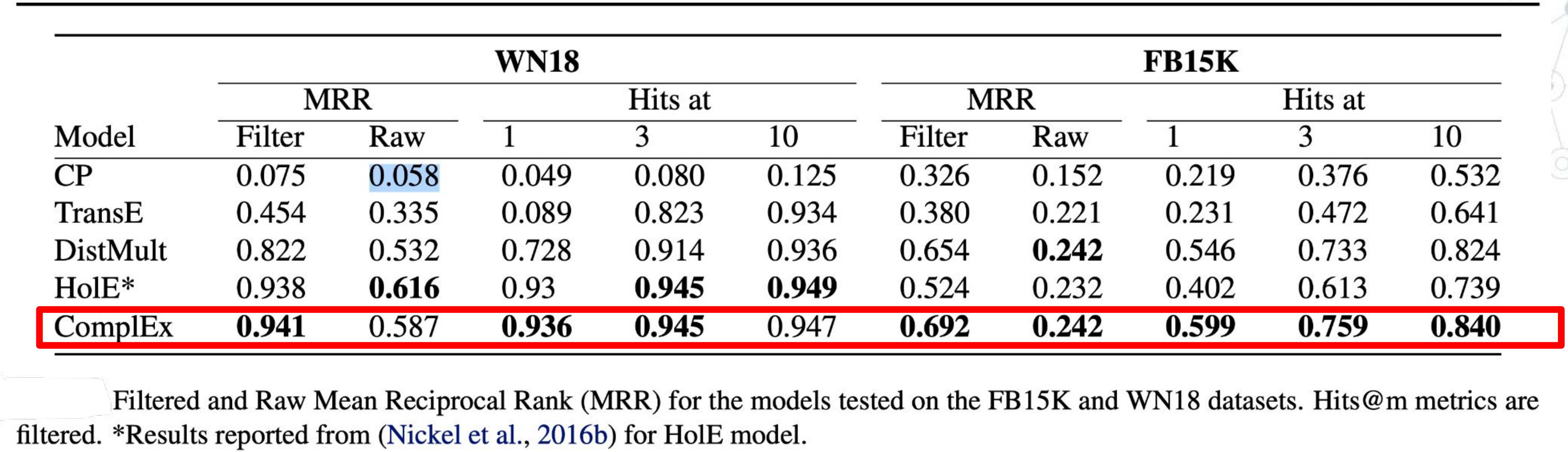
ComplEx Embeddings for Simple Link Prediction



Model	Scoring Function	Relation parameters	$\mathcal{O}_{time}$	$\mathcal{O}_{space}$
RESCAL (Nickel et al., 2011)	$e_s^T W_r e_o$	$W_r \in \mathbb{R}^{K^2}$	$\mathcal{O}(K^2)$	$\mathcal{O}(K^2)$
TransE (Bordes et al., 2013b)	$\|\|(e_s + w_r) - e_o\|\|_p$	$w_r \in \mathbb{R}^K$	$\mathcal{O}(K)$	$\mathcal{O}(K)$
NTN (Socher et al., 2013)	$u_r^T f(e_s W_r^{[1..D]} e_o + V_r \begin{bmatrix} e_s \\ e_o \end{bmatrix} + b_r)$	$W_r \in \mathbb{R}^{K^2 D}, b_r \in \mathbb{R}^K$ $V_r \in \mathbb{R}^{2KD}, u_r \in \mathbb{R}^K$	$\mathcal{O}(K^2 D)$	$\mathcal{O}(K^2 D)$
DistMult (Yang et al., 2015)	$< w_r, e_s, e_o >$	$w_r \in \mathbb{R}^K$	$\mathcal{O}(K)$	$\mathcal{O}(K)$
HolE (Nickel et al., 2016b)	$w_r^T(\mathcal{F}^{-1}[\overline{\mathcal{F}[e_s]} \odot \mathcal{F}[e_o]]))$	$w_r \in \mathbb{R}^K$	$\mathcal{O}(K \log K)$	$\mathcal{O}(K)$
ComplEx	$\mathrm{Re}(< w_r, e_s, \bar{e}_o >)$	$w_r \in \mathbb{C}^K$	$\mathcal{O}(K)$	$\mathcal{O}(K)$

Paper: Complex Embeddings for Simple Link Prediction

ComplEx Embeddings for Simple Link Prediction

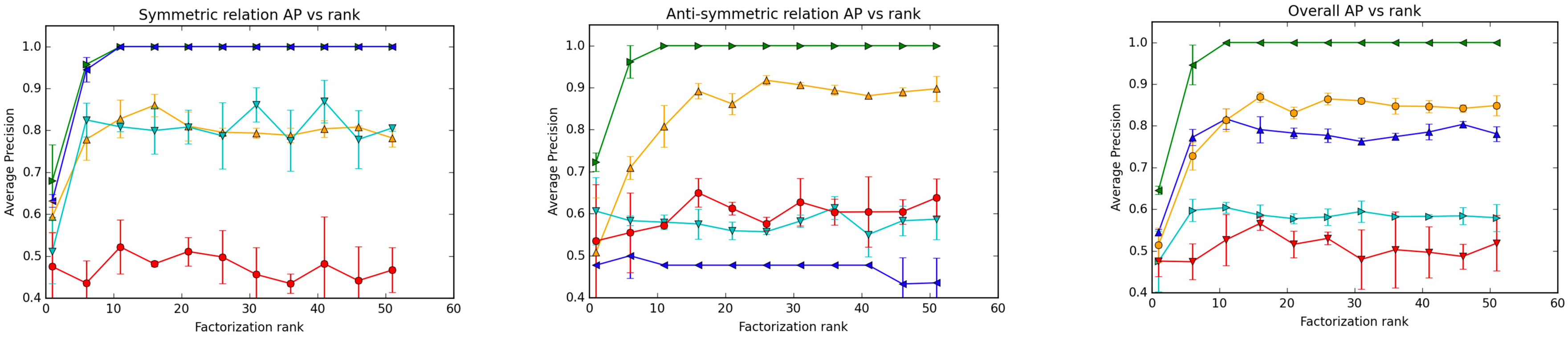


	WN18					FB15K				
	MRR		Hits at			MRR		Hits at		
Model	Filter	Raw	1	3	10	Filter	Raw	1	3	10
CP	0.075	0.058	0.049	0.080	0.125	0.326	0.152	0.219	0.376	0.532
TransE	0.454	0.335	0.089	0.823	0.934	0.380	0.221	0.231	0.472	0.641
DistMult	0.822	0.532	0.728	0.914	0.936	0.654	**0.242**	0.546	0.733	0.824
HolE*	0.938	**0.616**	0.93	**0.945**	**0.949**	0.524	0.232	0.402	0.613	0.739
ComplEx	**0.941**	0.587	**0.936**	**0.945**	0.947	**0.692**	**0.242**	**0.599**	**0.759**	**0.840**

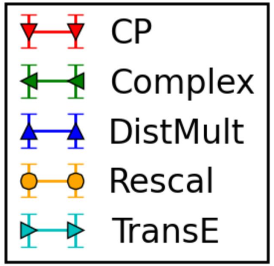
Filtered and Raw Mean Reciprocal Rank (MRR) for the models tested on the FB15K and WN18 datasets. Hits@m metrics are filtered. *Results reported from (Nickel et al., 2016b) for HolE model.

Paper: Complex Embeddings for Simple Link Prediction

ComplEx Embeddings for Simple Link Prediction



Average Precision (AP) for each factorization rank from 1-50 for different KGE models on asymmetry and symmetry experiments. Top-left: AP for symmetric relation only, middle: AP for anti-symmetric relation, right: overall AP.



Paper: Complex Embeddings for Simple Link Prediction

QuatE: Quaternion Knowledge Graph Embeddings

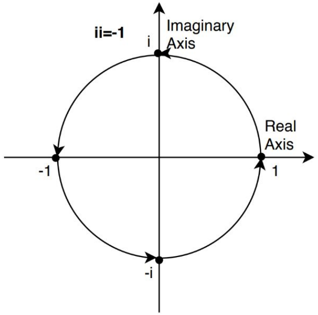
- QuatE: Hypercomplex representations to model entities and relations

  (1) rotate the head quaternion using the unit relation quaternion

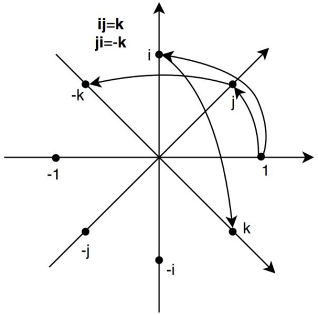
  (2) take the quaternion inner product between the rotated head quaternion and the tail quaternion to score each triplet

  Edge exists: rotated head entity has smaller angle between head/tail so the product is maximized

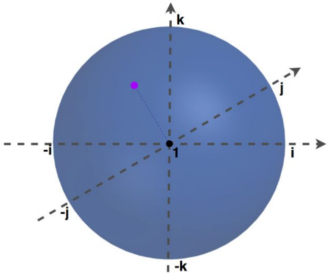
  - **Edge does not exist:** Head and tail entity are orthogonal so that their product becomes zero.



Complex Plane



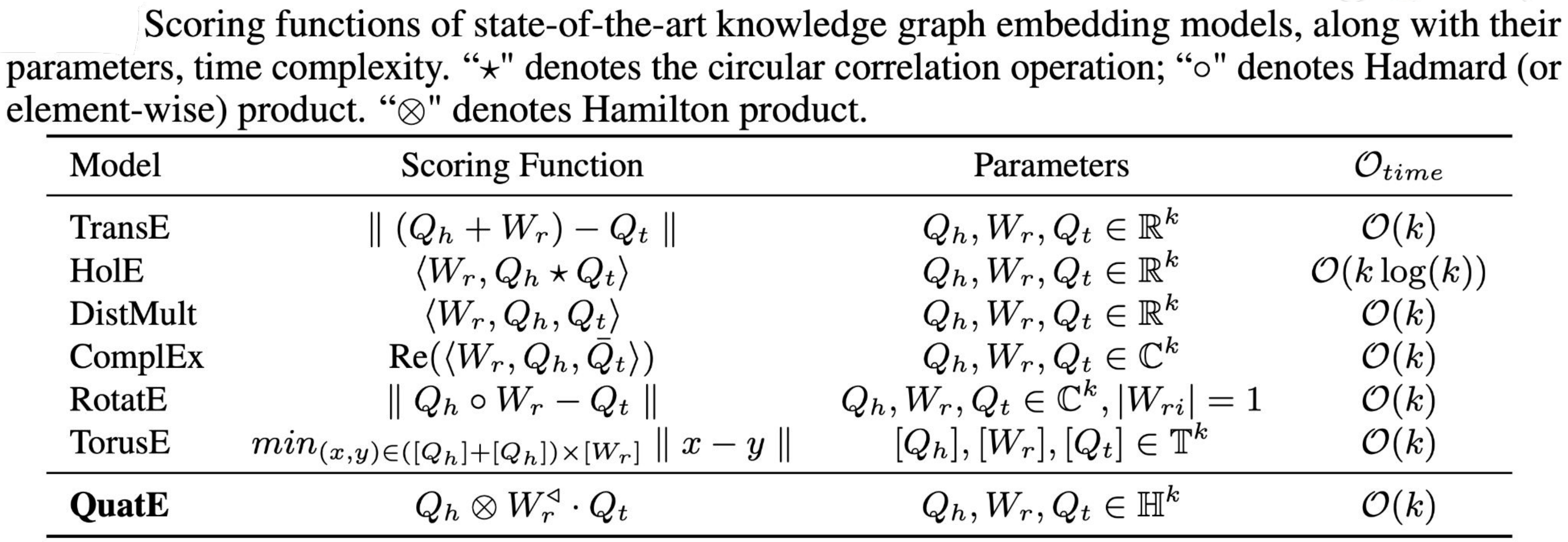
Quaternion units product



sterographically projected hypersphere in 3D space. The purple dot indicates the position of the unit quaternion.

Paper: [Quaternion Knowledge Graph Embeddings]

QuatE: Quaternion Knowledge Graph Embeddings

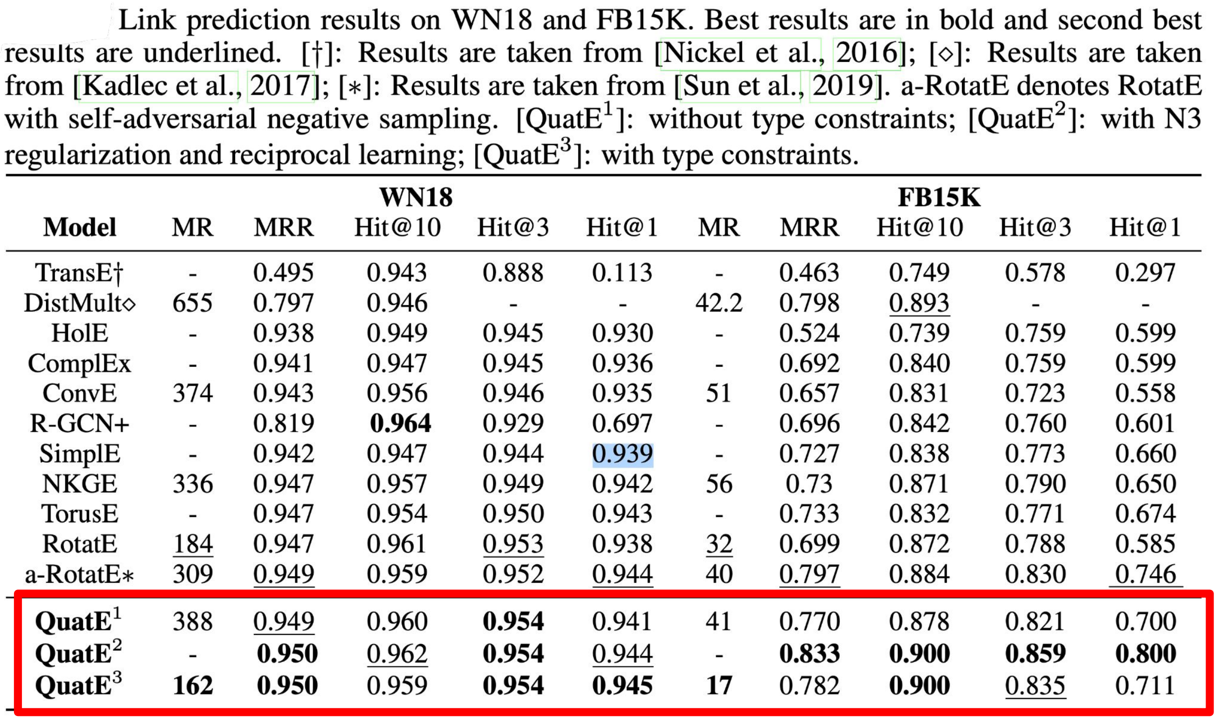


Scoring functions of state-of-the-art knowledge graph embedding models, along with their parameters, time complexity. "$\star$" denotes the circular correlation operation; "$\circ$" denotes Hadmard (or element-wise) product. "$\otimes$" denotes Hamilton product.

Model	Scoring Function	Parameters	$\mathcal{O}_{time}$
TransE	$\parallel (Q_h + W_r) - Q_t \parallel$	$Q_h, W_r, Q_t \in \mathbb{R}^k$	$\mathcal{O}(k)$
HolE	$\langle W_r, Q_h \star Q_t \rangle$	$Q_h, W_r, Q_t \in \mathbb{R}^k$	$\mathcal{O}(k \log(k))$
DistMult	$\langle W_r, Q_h, Q_t \rangle$	$Q_h, W_r, Q_t \in \mathbb{R}^k$	$\mathcal{O}(k)$
ComplEx	$\mathrm{Re}(\langle W_r, Q_h, \bar{Q}_t \rangle)$	$Q_h, W_r, Q_t \in \mathbb{C}^k$	$\mathcal{O}(k)$
RotatE	$\parallel Q_h \circ W_r - Q_t \parallel$	$Q_h, W_r, Q_t \in \mathbb{C}^k, \lvert W_{ri} \rvert = 1$	$\mathcal{O}(k)$
TorusE	$min_{(x,y)\in([Q_h]+[Q_h])\times[W_r]} \parallel x - y \parallel$	$[Q_h], [W_r], [Q_t] \in \mathbb{T}^k$	$\mathcal{O}(k)$
QuatE	$Q_h \otimes W_r^{\triangleleft} \cdot Q_t$	$Q_h, W_r, Q_t \in \mathbb{H}^k$	$\mathcal{O}(k)$

Paper: Quaternion Knowledge Graph Embeddings

QuatE: Quaternion Knowledge Graph Embeddings

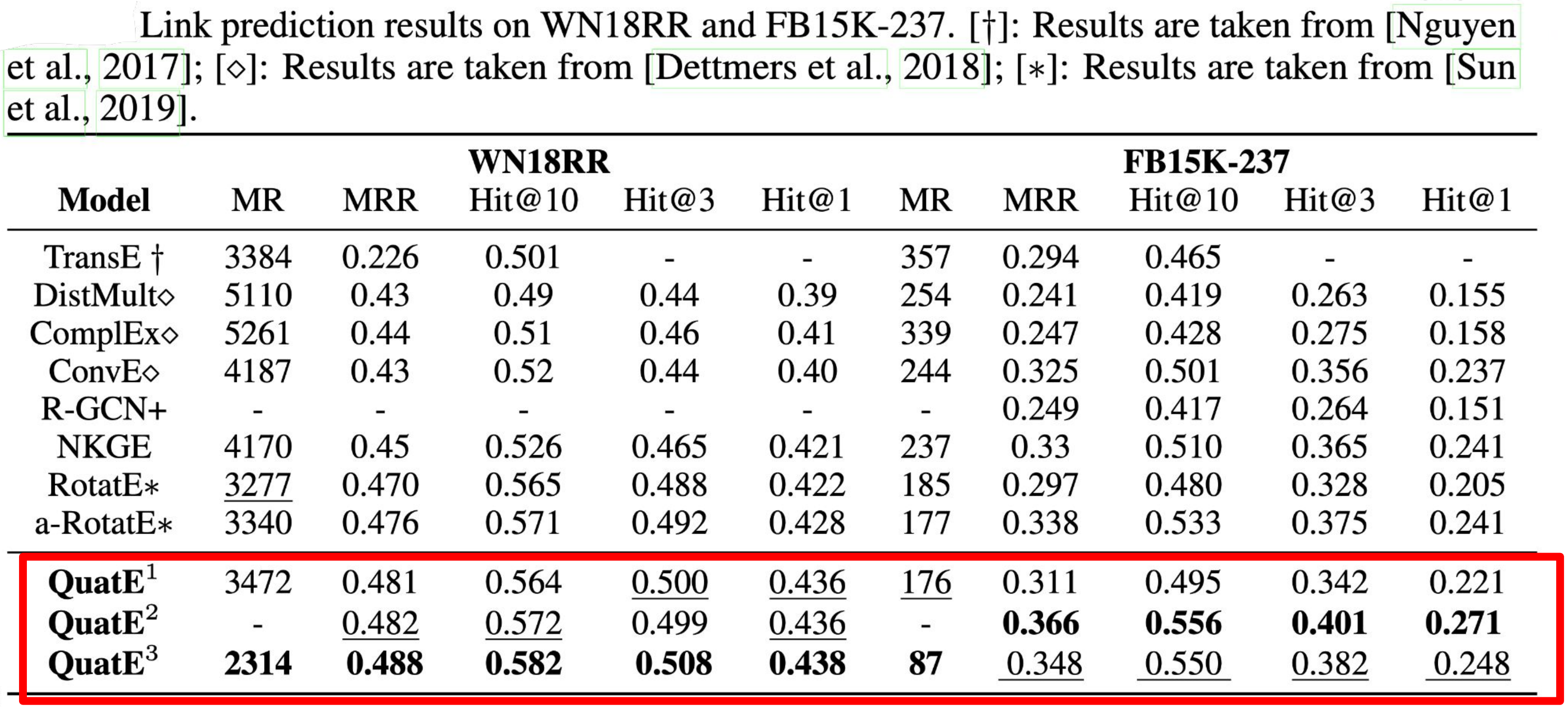


Link prediction results on WN18 and FB15K. Best results are in bold and second best results are underlined. [†]: Results are taken from [Nickel et al., 2016]; [◇]: Results are taken from [Kadlec et al., 2017]; [∗]: Results are taken from [Sun et al., 2019]. a-RotatE denotes RotatE with self-adversarial negative sampling. [QuatE^1]: without type constraints; [QuatE^2]: with N3 regularization and reciprocal learning; [QuatE^3]: with type constraints.

			WN18					**FB15K**		
Model	MR	MRR	Hit@10	Hit@3	Hit@1	MR	MRR	Hit@10	Hit@3	Hit@1
TransE†	-	0.495	0.943	0.888	0.113	-	0.463	0.749	0.578	0.297
DistMult◇	655	0.797	0.946	-	-	42.2	0.798	0.893	-	-
HolE	-	0.938	0.949	0.945	0.930	-	0.524	0.739	0.759	0.599
ComplEx	-	0.941	0.947	0.945	0.936	-	0.692	0.840	0.759	0.599
ConvE	374	0.943	0.956	0.946	0.935	51	0.657	0.831	0.723	0.558
R-GCN+	-	0.819	**0.964**	0.929	0.697	-	0.696	0.842	0.760	0.601
SimplE	-	0.942	0.947	0.944	0.939	-	0.727	0.838	0.773	0.660
NKGE	336	0.947	0.957	0.949	0.942	56	0.73	0.871	0.790	0.650
TorusE	-	0.947	0.954	0.950	0.943	-	0.733	0.832	0.771	0.674
RotatE	184	0.947	0.961	0.953	0.938	32	0.699	0.872	0.788	0.585
a-RotatE∗	309	0.949	0.959	0.952	0.944	40	0.797	0.884	0.830	0.746
QuatE^1	388	0.949	0.960	**0.954**	0.941	41	0.770	0.878	0.821	0.700
QuatE^2	-	**0.950**	0.962	**0.954**	0.944	-	**0.833**	**0.900**	**0.859**	**0.800**
QuatE^3	**162**	**0.950**	0.959	**0.954**	**0.945**	**17**	0.782	**0.900**	0.835	0.711

Paper: Quaternion Knowledge Graph Embeddings

QuatE: Quaternion Knowledge Graph Embeddings



Link prediction results on WN18RR and FB15K-237. [†]: Results are taken from [Nguyen et al., 2017]; [◇]: Results are taken from [Dettmers et al., 2018]; [∗]: Results are taken from [Sun et al., 2019].

	WN18RR					FB15K-237				
Model	MR	MRR	Hit@10	Hit@3	Hit@1	MR	MRR	Hit@10	Hit@3	Hit@1
TransE †	3384	0.226	0.501	-	-	357	0.294	0.465	-	-
DistMult◇	5110	0.43	0.49	0.44	0.39	254	0.241	0.419	0.263	0.155
ComplEx◇	5261	0.44	0.51	0.46	0.41	339	0.247	0.428	0.275	0.158
ConvE◇	4187	0.43	0.52	0.44	0.40	244	0.325	0.501	0.356	0.237
R-GCN+	-	-	-	-	-	-	0.249	0.417	0.264	0.151
NKGE	4170	0.45	0.526	0.465	0.421	237	0.33	0.510	0.365	0.241
RotatE∗	3277	0.470	0.565	0.488	0.422	185	0.297	0.480	0.328	0.205
a-RotatE∗	3340	0.476	0.571	0.492	0.428	177	0.338	0.533	0.375	0.241
QuatE^1	3472	0.481	0.564	0.500	0.436	176	0.311	0.495	0.342	0.221
QuatE^2	-	0.482	0.572	0.499	0.436	-	**0.366**	**0.556**	**0.401**	**0.271**
QuatE^3	**2314**	**0.488**	**0.582**	**0.508**	**0.438**	**87**	0.348	0.550	0.382	0.248

Paper: Quaternion Knowledge Graph Embeddings

Break time!

We’ll continue in 30 minutes

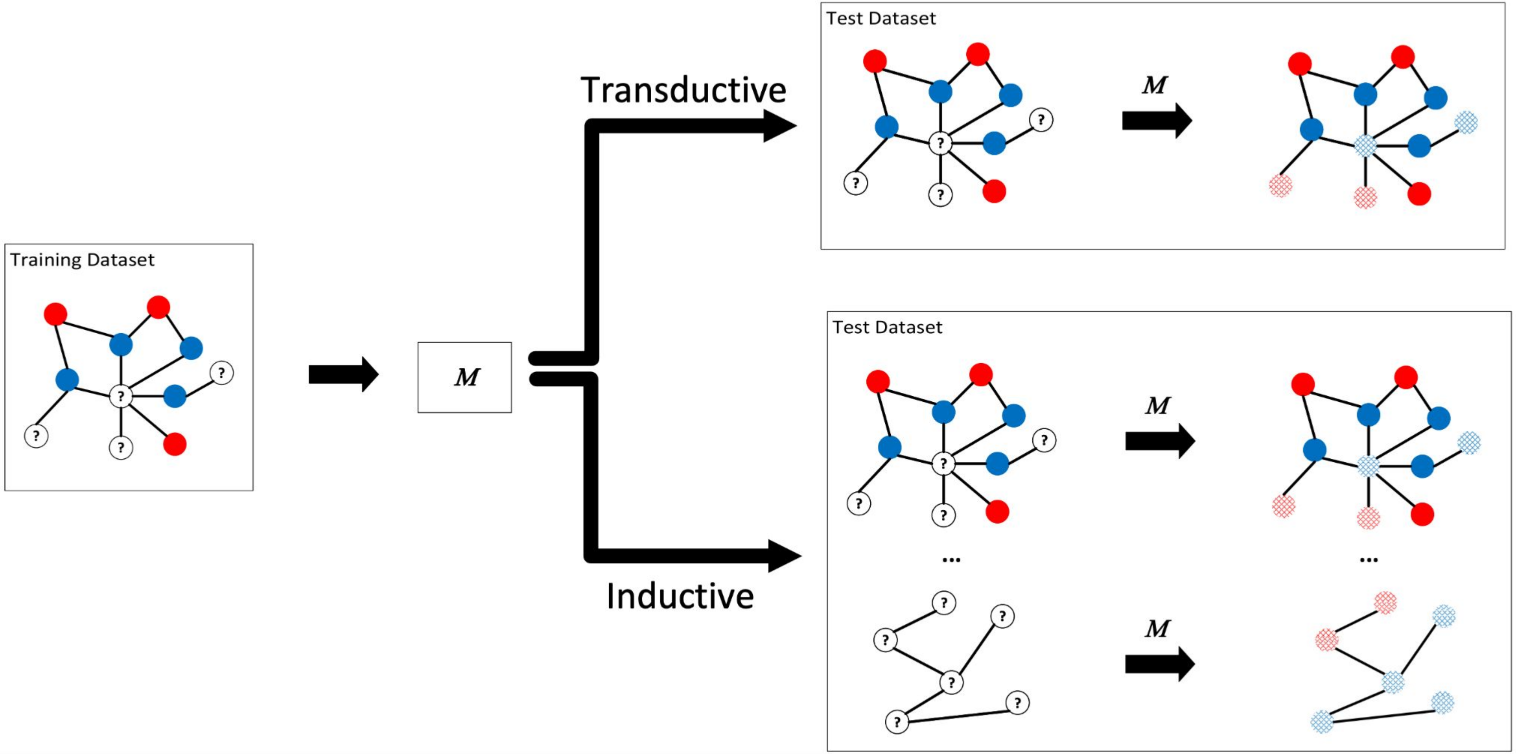
Graph Neural Networks

Michael Bronstein

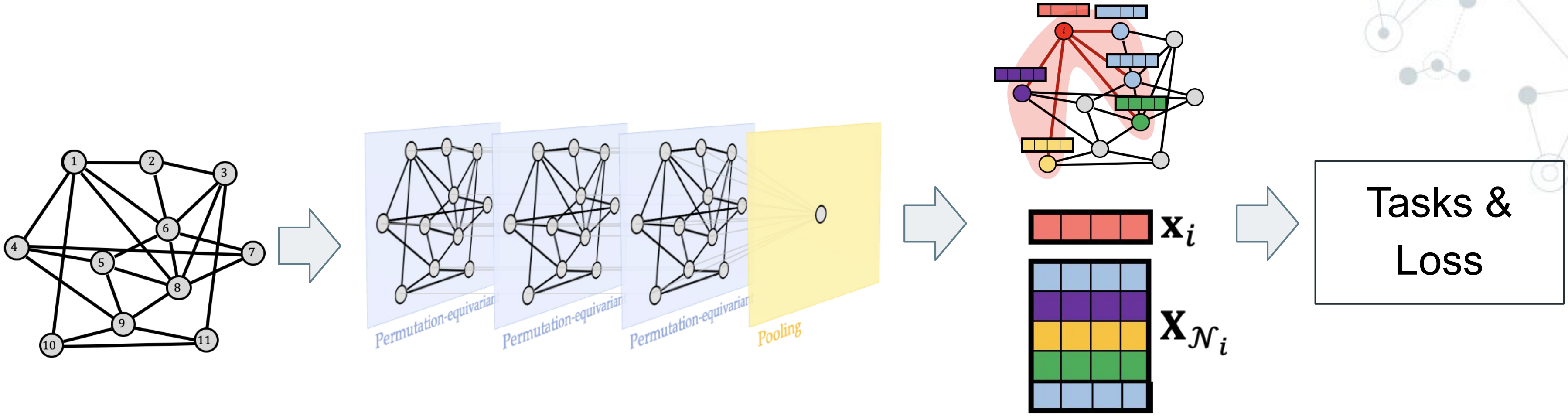
Beyond Shallow Embeddings: Deep Learning on Graphs

- Shallow embeddings are highly scalable due to their simplicity
  - Easy to train shallow embeddings for billions of nodes and trillions of edges
- However, this simplicity comes at a great cost
  - Shallow embeddings are transductive
  - Cannot generalize to new nodes / graphs
- Deep learning can allow us to have inductive node embeddings
  - Embed new nodes and new graphs

Inductive vs Transductive Embeddings



Graph Neural Networks



Tasks & Loss

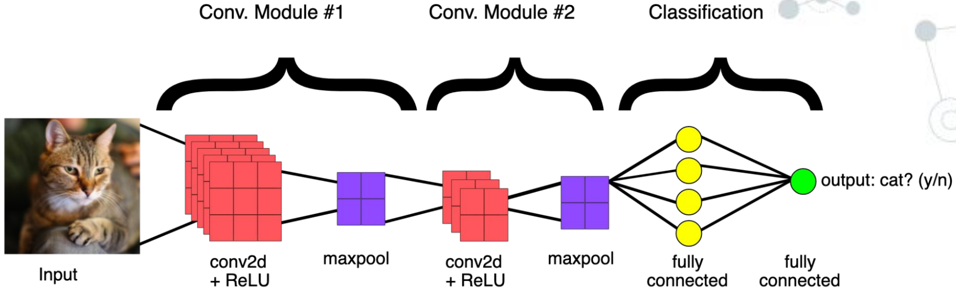
Input graph

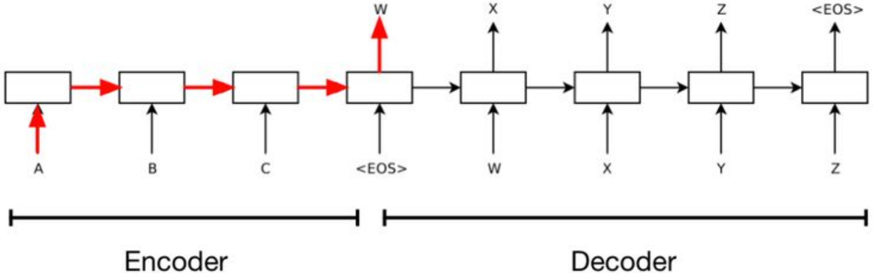
GNN

Node Embeddings

Challenges to Deep Learning on Graphs

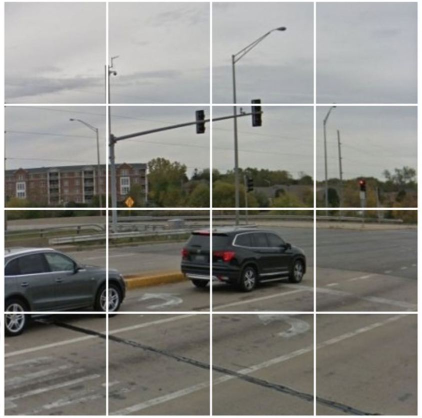
- Standard deep learning is designed for structured inputs
  - grid images
  - text sequences
- Performing deep learning on graphs is different than on images or text



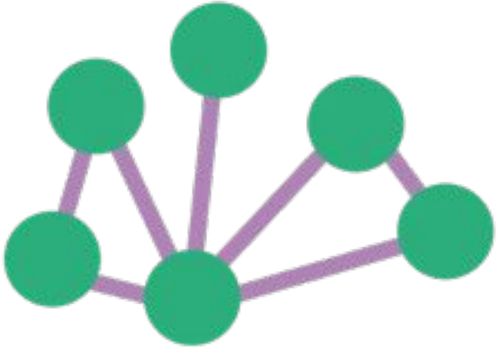


Why is Deep Learning on Graphs Hard?

- Not all data has locality / lives on a grid
  - Graphs lack locality
  - While Images / text can be plot on a grid
- Graphs can be arbitrarily large
- There is no canonical node ordering for graphs



VS

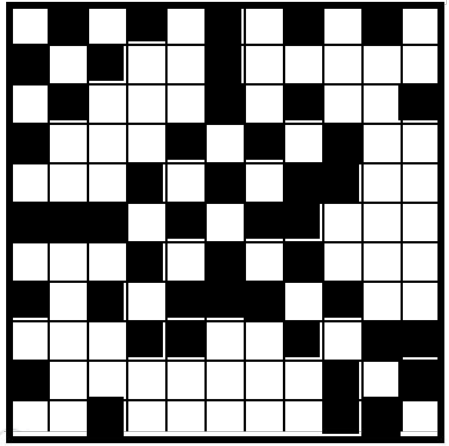


Graph Symmetries and Permutation Invariance

Graph symmetries: permutations

Adjacency matrix

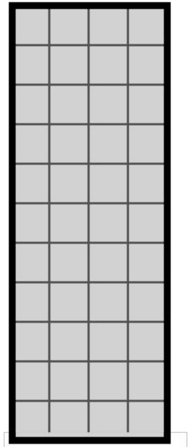
n×n



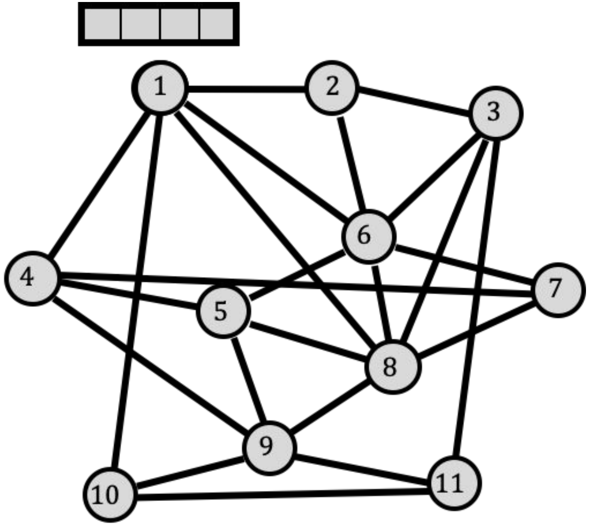
A

Feature matrix

n×d



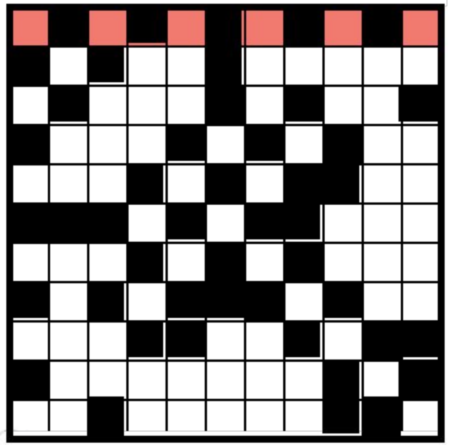
X



Graph symmetries: permutations

Adjacency matrix

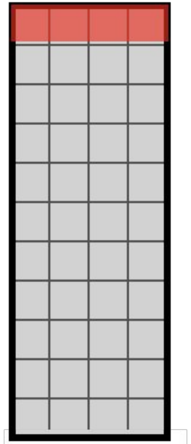
n×n



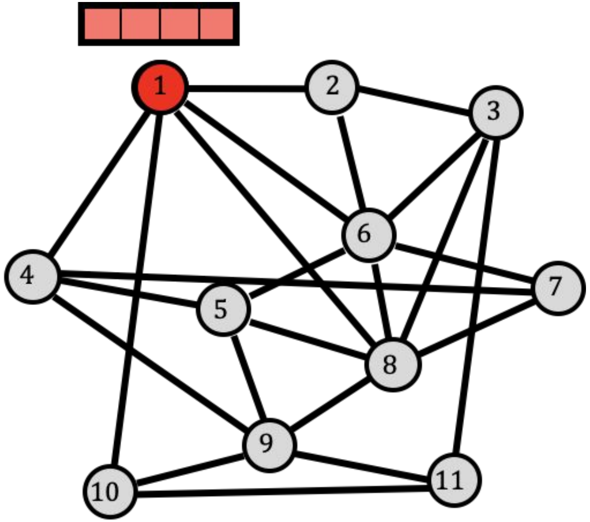
A

Feature matrix

n×d



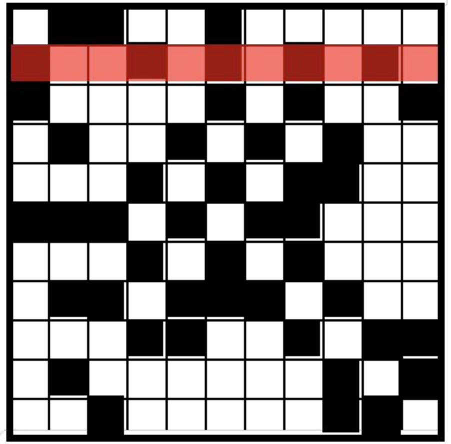
X



Graph symmetries: permutations

Adjacency matrix

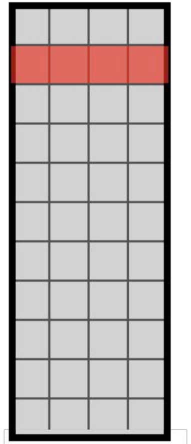
n×n



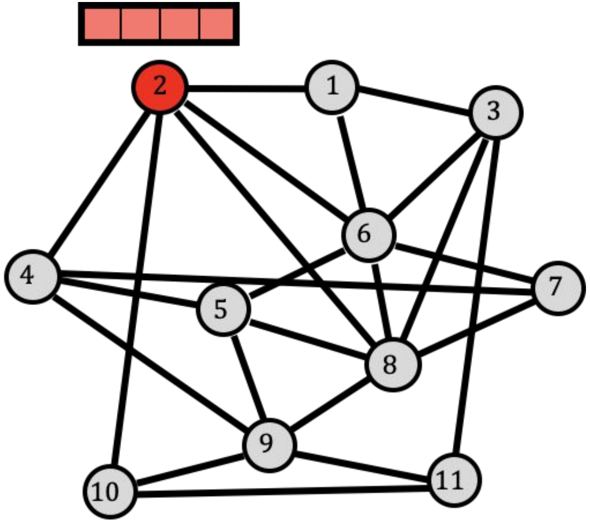
PAP^T

Feature matrix

n×d

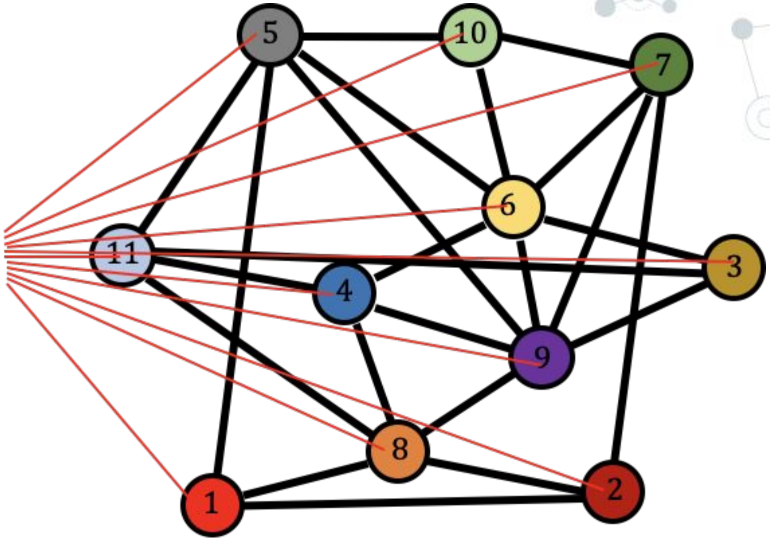


PX



Permutation invariance

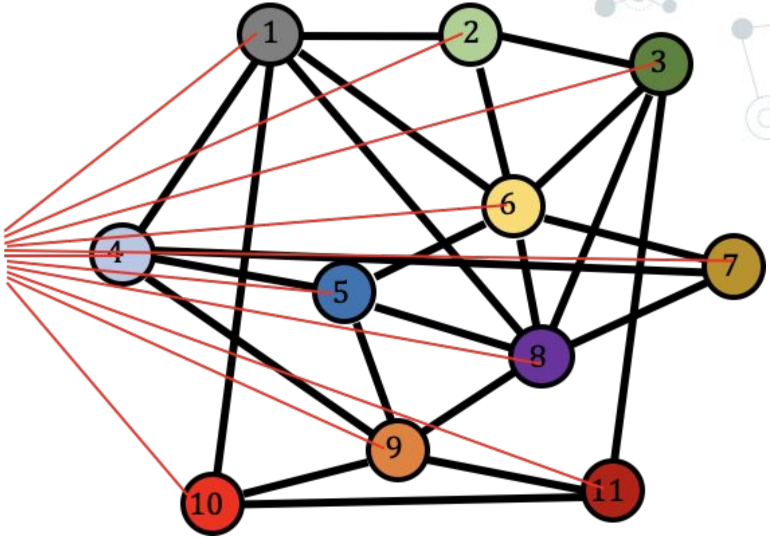
graph function $f(\mathbf{X},\mathbf{A})$



- Graph Neural networks consist of a shared function that operates on every node
  - The input are the collection of features in the neighbors of every node

Permutation invariance

graph function f(**X**,**A**)

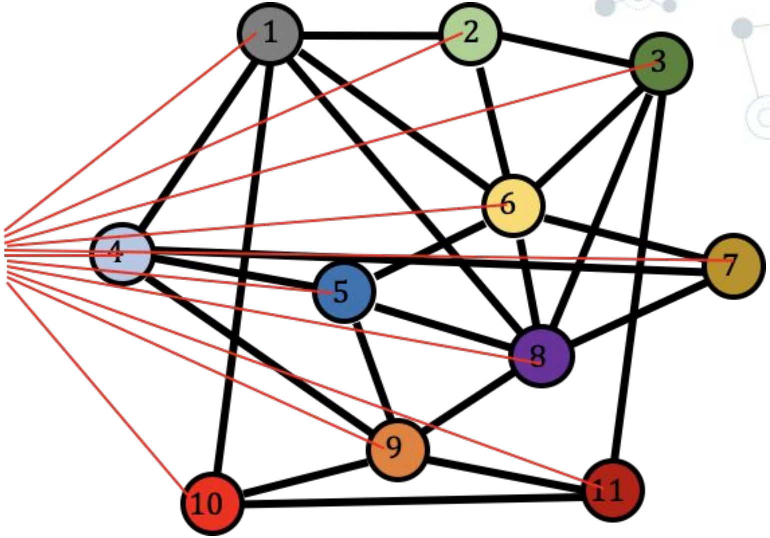


- Because we don't have any canonical ordering of the neighboring nodes, this graph function must be ***permutation invariant***

Permutation invariance

$$f(\mathbf{PX},\mathbf{PAP}^{\mathrm{T}}) = f(\mathbf{X},\mathbf{A})$$

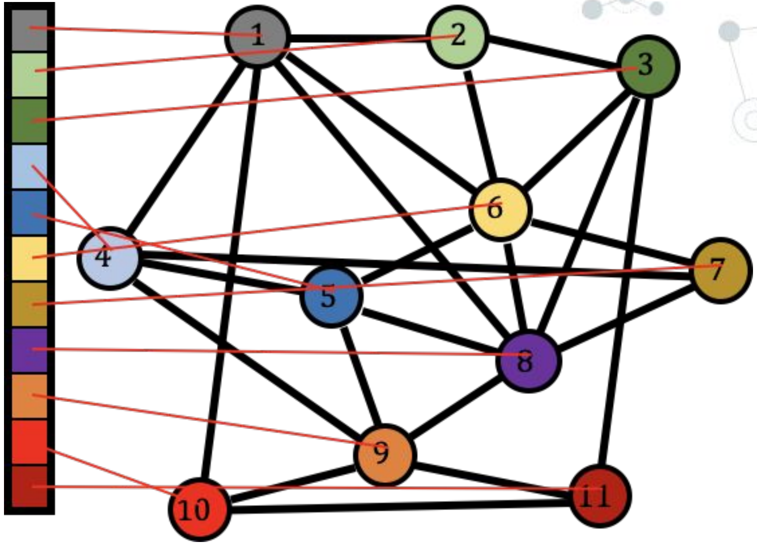
permutation invariant



◎ Because we don’t have any canonical ordering of the neighboring nodes, this graph function must be ***permutation invariant***

Permutation equivariance

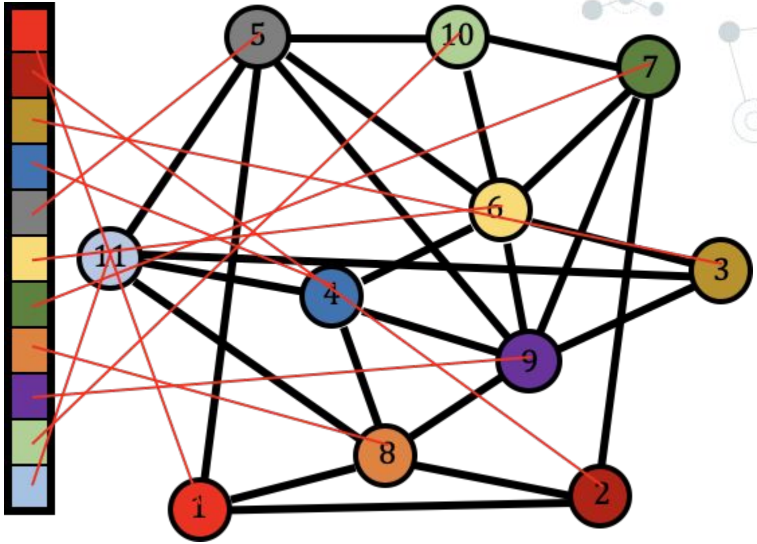
node function **F**(**X**,**A**)



- Apply this function to every node of the graph

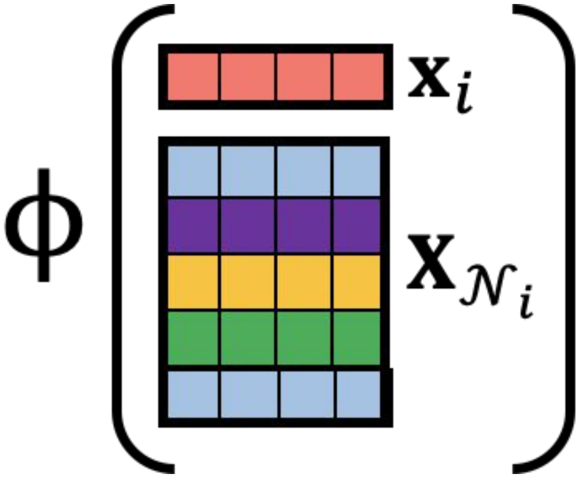
Permutation equivariance

node function $\mathbf{F}(\mathbf{X},\mathbf{A})$

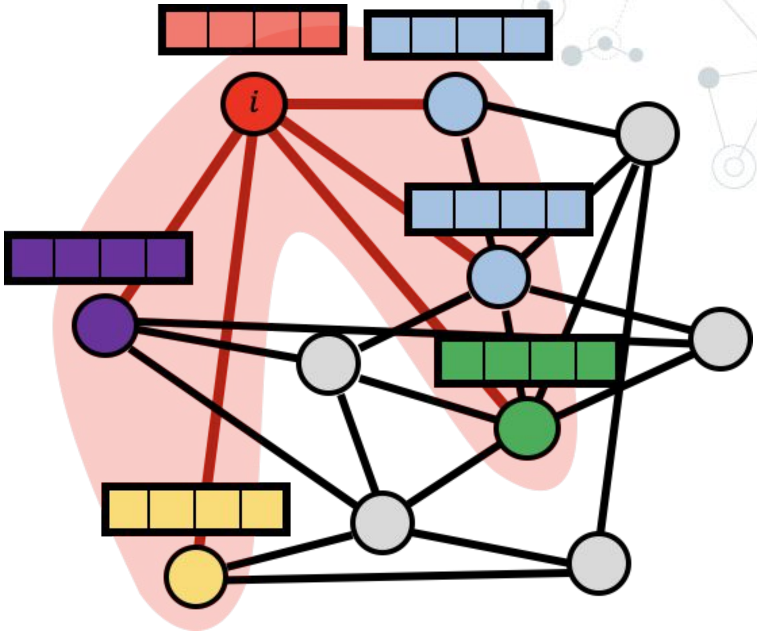


- Apply this function to every node of the graph

Local aggregation



permutation invariant



- Apply this function to every node of the graph
  - Picking the right function such that results in permutation equivariant node-wise function

Local aggregation

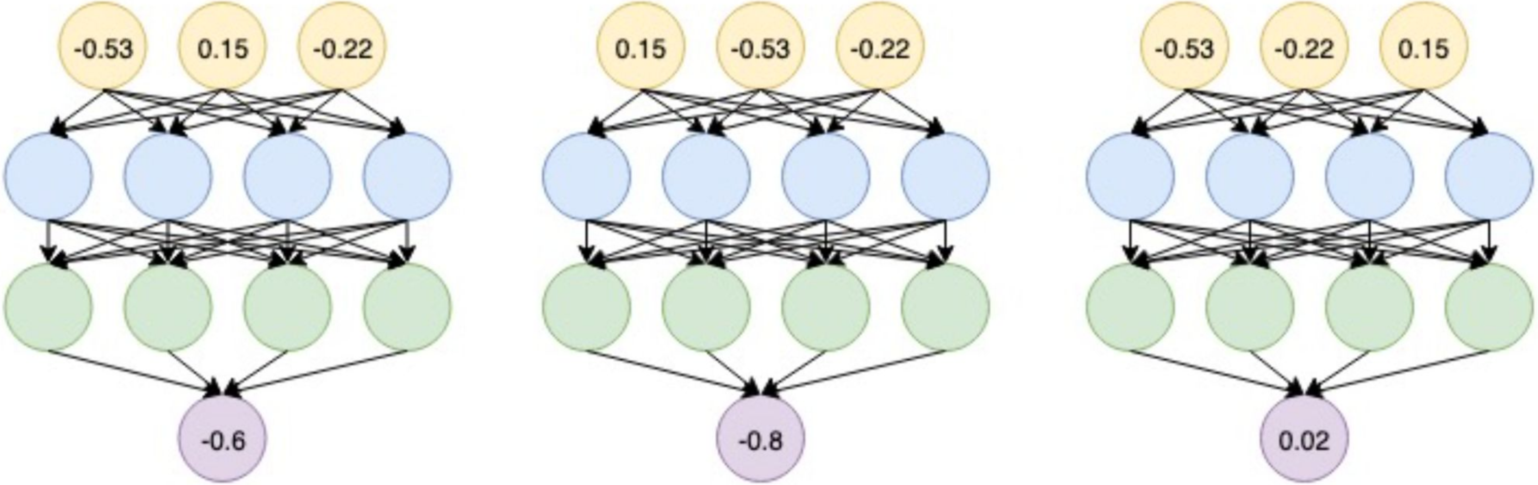
$$\mathbf{F}(\mathbf{X},\mathbf{A}) = \begin{pmatrix} - \phi\left(\mathbf{x}_1, \mathbf{X}_{\mathcal{N}_1}\right) - \\ \vdots \\ - \phi\left(\mathbf{x}_i, \mathbf{X}_{\mathcal{N}_i}\right) - \\ \vdots \\ - \phi\left(\mathbf{x}_n, \mathbf{X}_{\mathcal{N}_n}\right) - \end{pmatrix}$$

permutation equivariant

- Apply this function to every node of the graph
  - Picking the right function such that results in permutation equivariant node-wise function

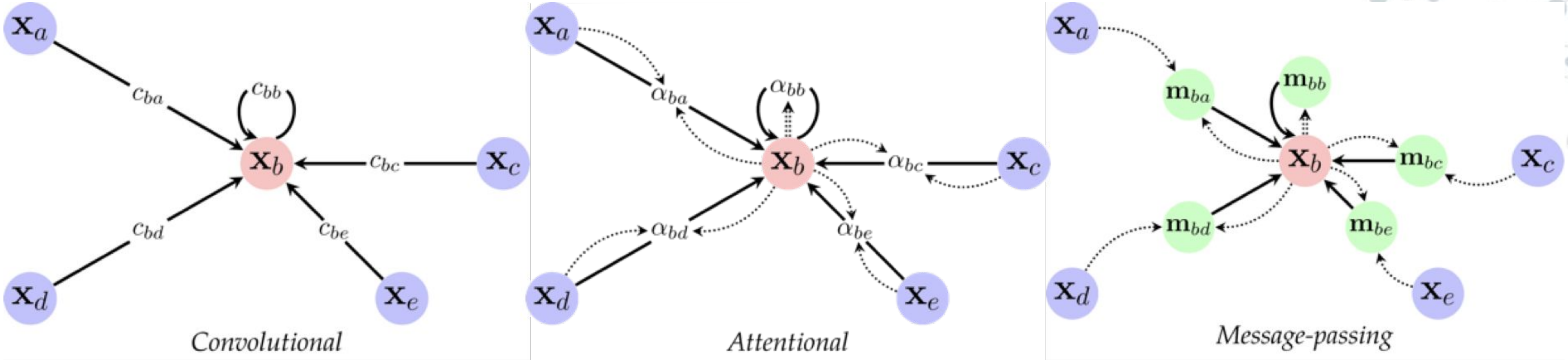
Are all Neural Network Architectures Permutation Equivariant?

- Not all neural architectures are permutation equivariant
  - Multi-layer perceptrons are not permutation invariant
  - Permuting the input changes the output



Need permutation equivariant / invariant architectures for GNNs

Flavors of GNNs



Convolutional *Attentional* *Message-passing*

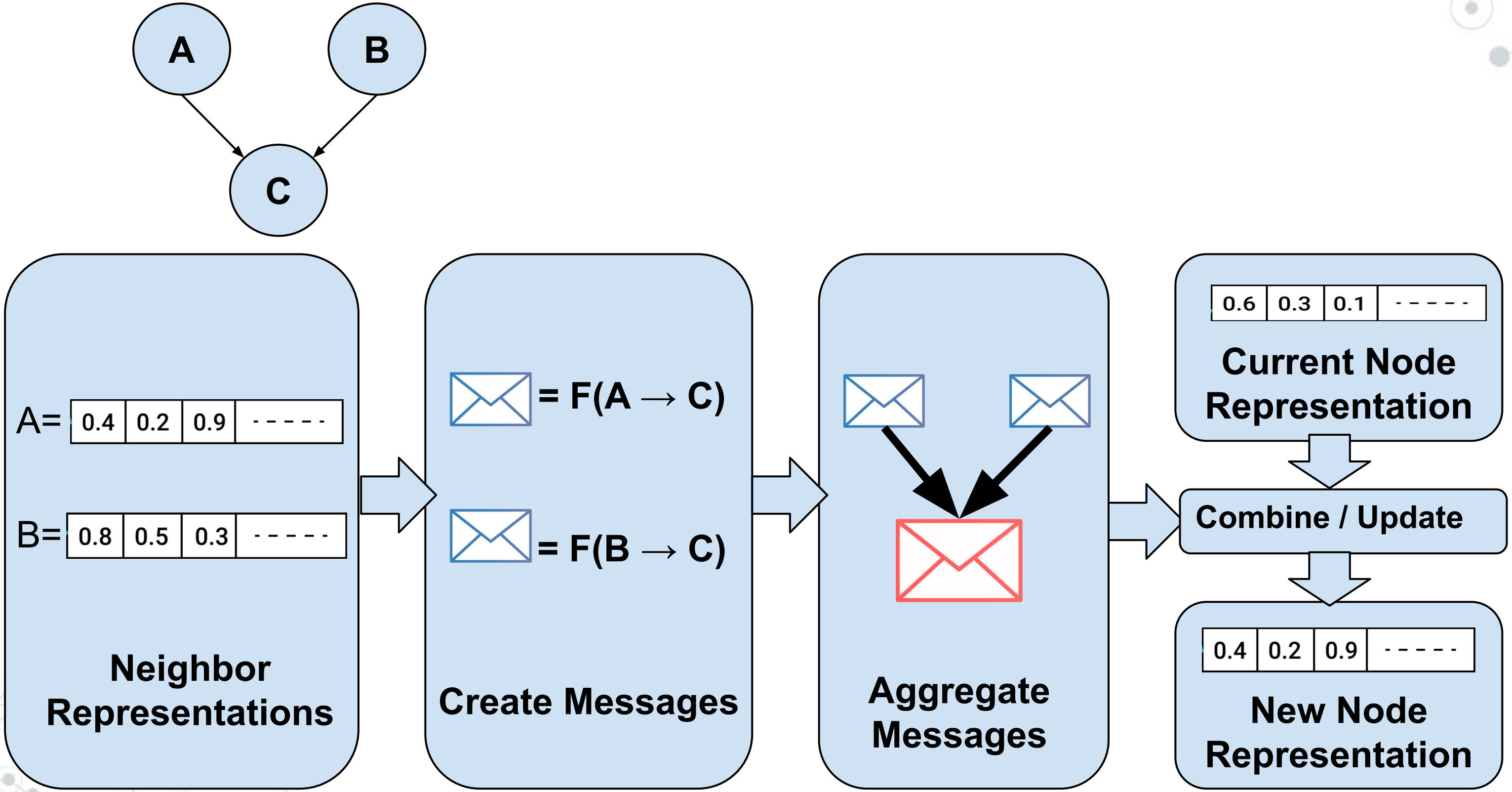
$$\mathbf{h}_i = \phi\left(\mathbf{x}_i, \bigoplus_{j \in \mathcal{N}_i} c_{ij}\psi(\mathbf{x}_j)\right)$$

$$\mathbf{h}_i = \phi\left(\mathbf{x}_i, \bigoplus_{j \in \mathcal{N}_i} a(\mathbf{x}_i, \mathbf{x}_j)\psi(\mathbf{x}_j)\right)$$

$$\mathbf{h}_i = \phi\left(\mathbf{x}_i, \bigoplus_{j \in \mathcal{N}_i} \psi(\mathbf{x}_i, \mathbf{x}_j)\right)$$

Neural Message Passing

Neural Message Passing



Simple Message Passing

Two-step process

1. Average messages from neighbors
2. Apply a neural network to passed messages + current node

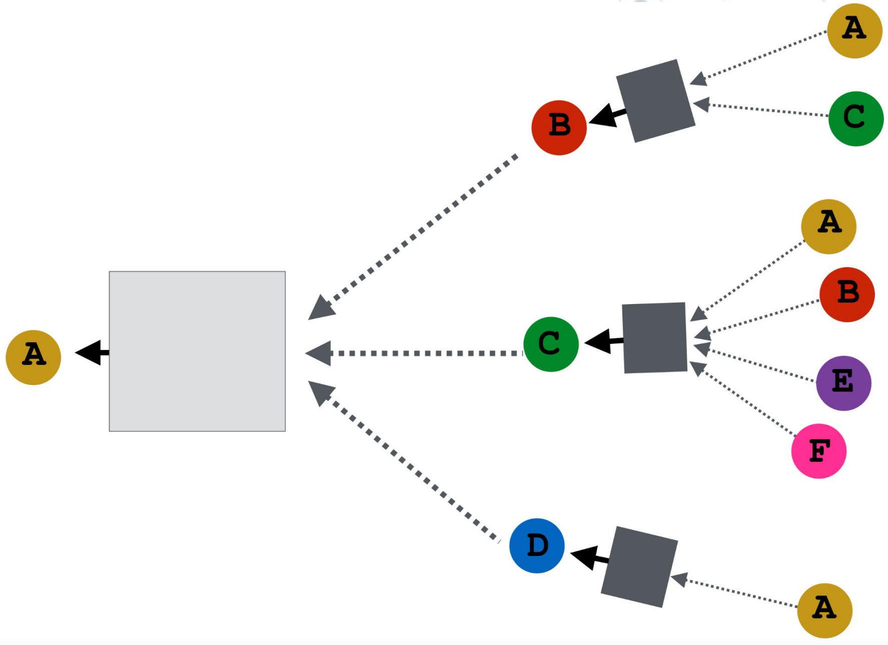
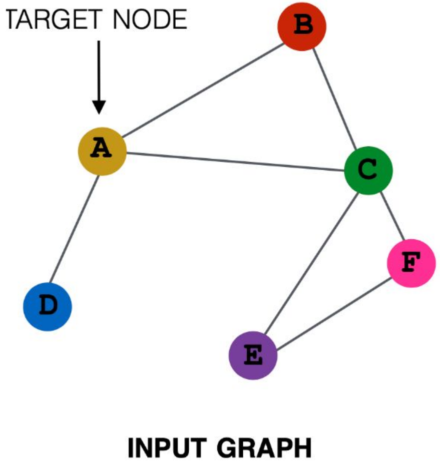


Diagram from Jure Leskovec, Stanford University

Simple Message Passing



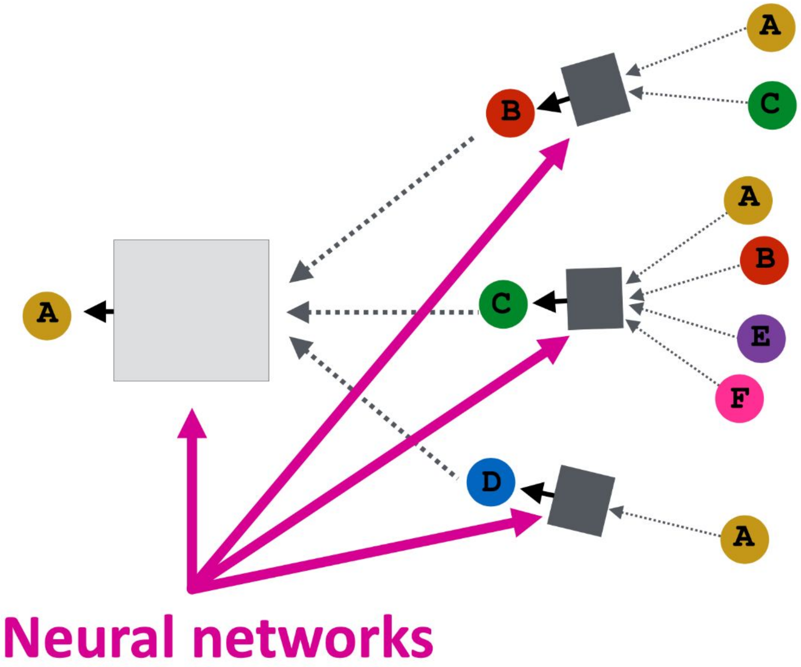
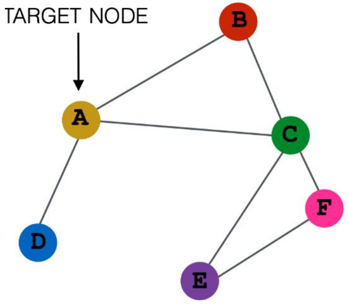


Diagram from Jure Leskovec, Stanford University

Simple Message Passing: Arbitrary Depth

Model can be applied at arbitrary proximity-depth (hops)

1. Nodes have embeddings at each layer
2. Layer 0 representations are the features of a node
3. Layer 1 representation gets message from nodes 1-hop away



INPUT GRAPH

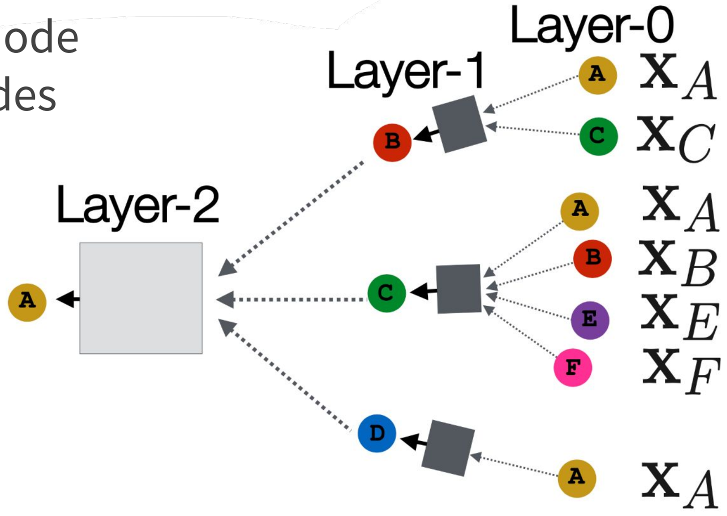


Diagram from Jure Leskovec, Stanford University

Simple Message Passing: Update Function

1. Pool messages
   a. averaging works
2. then apply a neural network

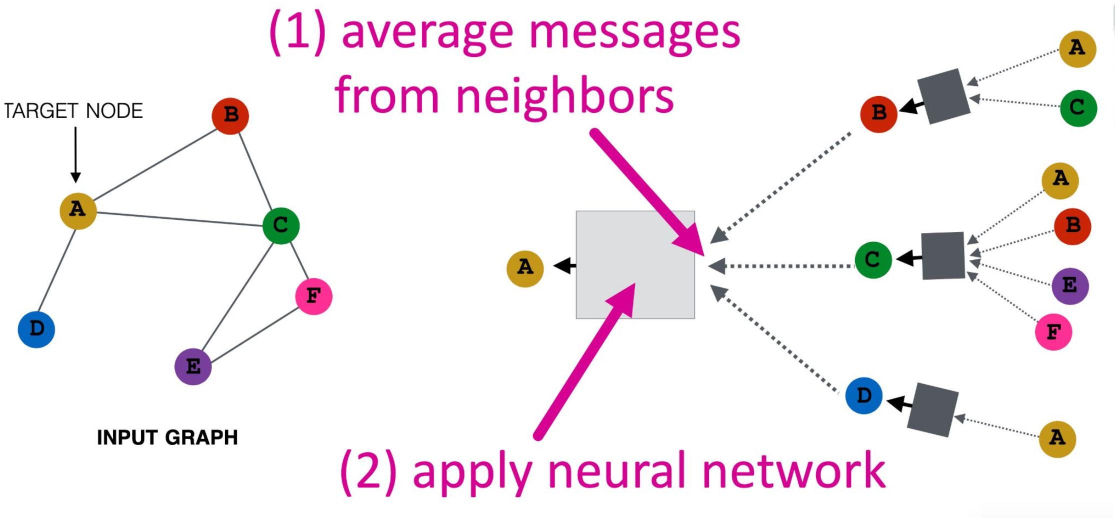


Diagram from Jure Leskovec, Stanford University

Neural Message Passing Example

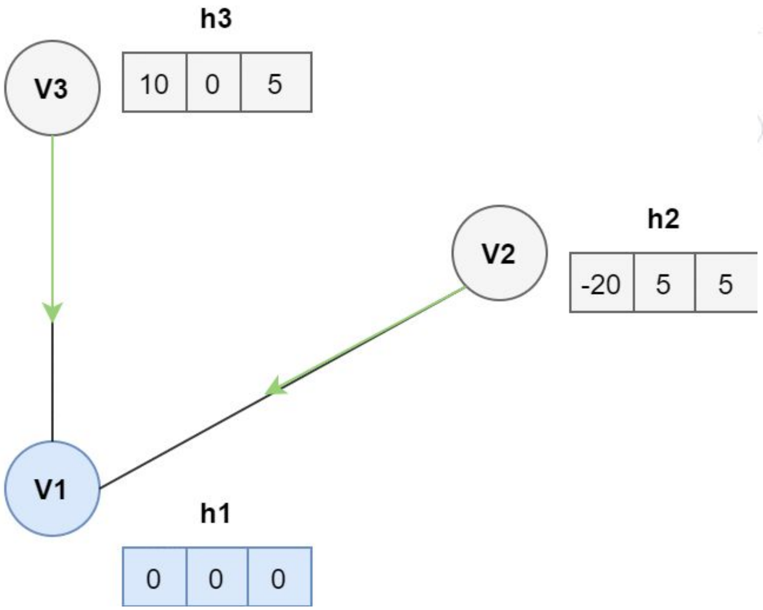
t=1

Simple message passing

$$m_v^{t+1} = \sum_{w \in N(v)} h_w^t$$

$$h_v^{t+1} = average(h_v, m_v^{t+1})$$

ht - hidden state for each node



[diagram source](#)

Neural Message Passing Example

t=1

Message is sum of neighbor's hidden states

$$m_v^{t+1} = \sum_{w \in N(v)} h_w^t$$

$$h_v^{t+1} = average(h_v, m_v^{t+1})$$

ht - hidden state for each node

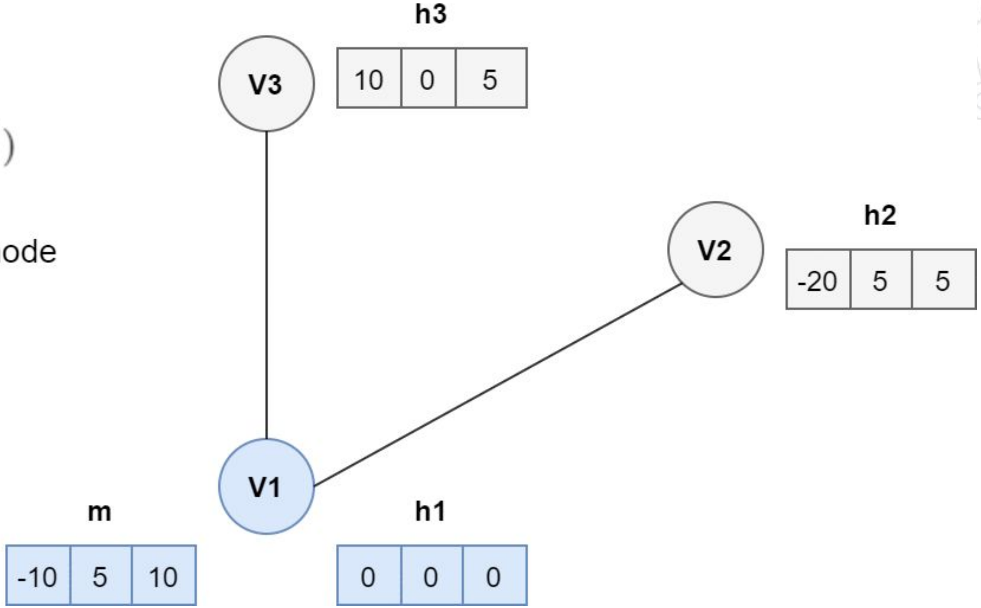


diagram source

Neural Message Passing Example

t=2

Update function is the average of current hidden state and message

$$m_v^{t+1} = \sum_{w \in N(v)} h_w^t$$

$$h_v^{t+1} = average(h_v, m_v^{t+1})$$

ıt - hidden state for each node

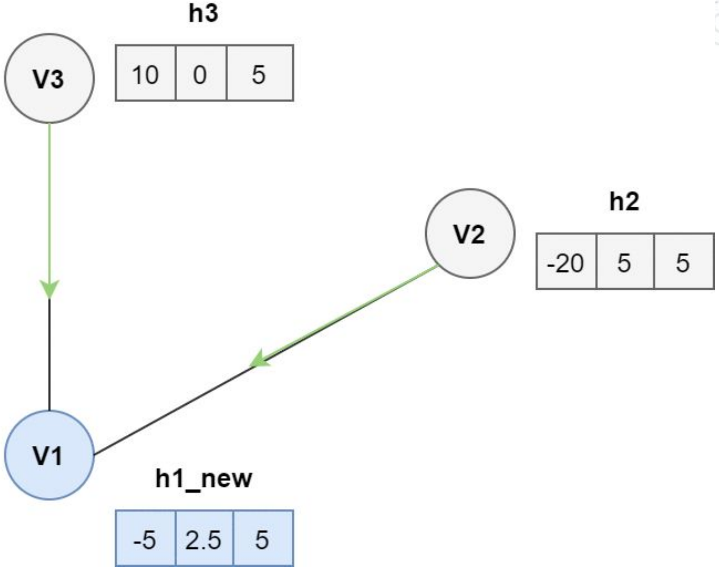
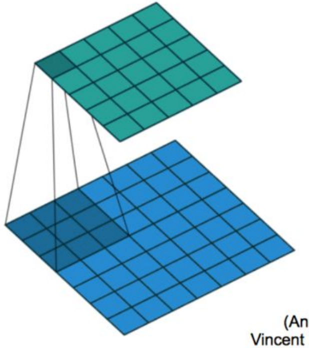


diagram source

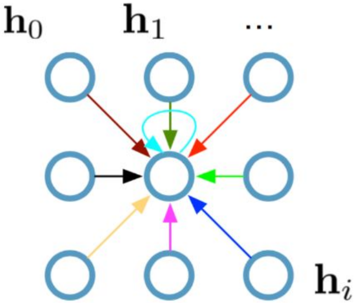
Graph Convolutional Networks

Standard Convolutional Neural Networks (CNN)

Single CNN layer with 3x3 filter:



(Animation by Vincent Dumoulin)



Update for a single pixel:

- Transform messages individually $\mathbf{W}_i\mathbf{h}_i$
- Add everything up $\sum_i \mathbf{W}_i\mathbf{h}_i$

$\mathbf{h}_i \in \mathbb{R}^F$ are (hidden layer) activations of a pixel/node

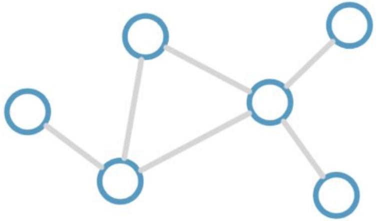
Full update:

$$\mathbf{h}_4^{(l+1)} = \sigma\left(\mathbf{W}_0^{(l)}\mathbf{h}_0^{(l)} + \mathbf{W}_1^{(l)}\mathbf{h}_1^{(l)} + \cdots + \mathbf{W}_8^{(l)}\mathbf{h}_8^{(l)}\right)$$

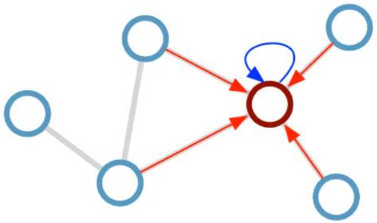
slide from Thomas Kipf, University of Amsterdam

Graph Convolutional Neural Networks (GNN)

Consider this undirected graph:



Calculate update for node in red:



Desirable properties:

- Weight sharing over all locations
- Invariance to permutations
- Linear complexity O(E)
- Applicable both in transductive and inductive settings

Update rule:

$$\mathbf{h}_i^{(l+1)} = \sigma\left(\mathbf{h}_i^{(l)}\mathbf{W}_0^{(l)} + \sum_{j\in\mathcal{N}_i}\frac{1}{c_{ij}}\mathbf{h}_j^{(l)}\mathbf{W}_1^{(l)}\right)$$

Scalability: subsample messages [Hamilton et al., NIPS 2017]

$\mathcal{N}_i$: neighbor indices

c_{ij} : norm. constant (fixed/trainable)

Paper: Semi-supervised Classification with Graph Convolutional Networks

*slide from Thomas Kipf, University of Amsterdam

Graph Convolutional Neural Networks (GNN)

parameters in layer k

Non-linear activation function (e.g., ReLU)

$$\boldsymbol{h}_v^k = \sigma(\boldsymbol{W}_k \sum_{u \in \mathrm{N}(v) \cup v} \frac{h_u^{k-1}}{\sqrt{|N(u)||N(v)|}})$$

node v's embedding at layer k

the neighbors of node v

Paper: Semi-supervised Classification with Graph Convolutional Networks

Graph Convolutional Neural Networks (GNN)

Aggregate from v's neighbors

$$\boldsymbol{h}_v^k = \sigma(\boldsymbol{W}_k \sum_{u \in \mathrm{N}(v)} \frac{h_u^{k-1}}{\sqrt{|N(u)||N(v)|}} +$$

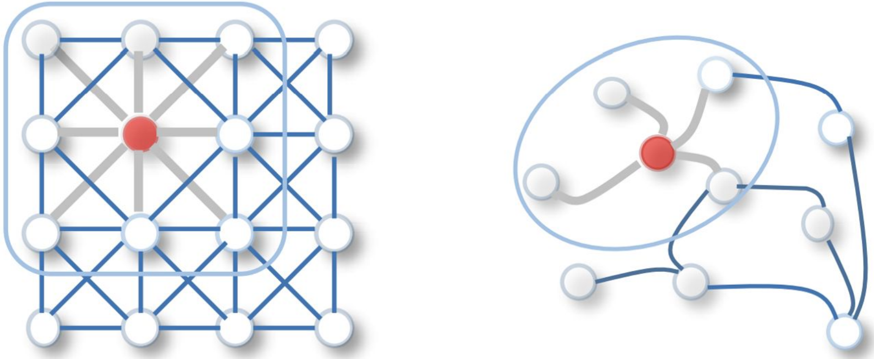
$$\boldsymbol{W}_k \sum_{v} \frac{h_v^{k-1}}{\sqrt{|N(v)||N(v)|}})$$

Aggregate from itself

Paper: Semi-supervised Classification with Graph Convolutional Networks

Relationships between CNNs and GNN

- A convolutional neural network (CNN) is a special case of a graph neural network
- While the size of the filter is pre-defined in a CNN, a GNN takes in nodes with arbitrary degree (neighboring nodes)

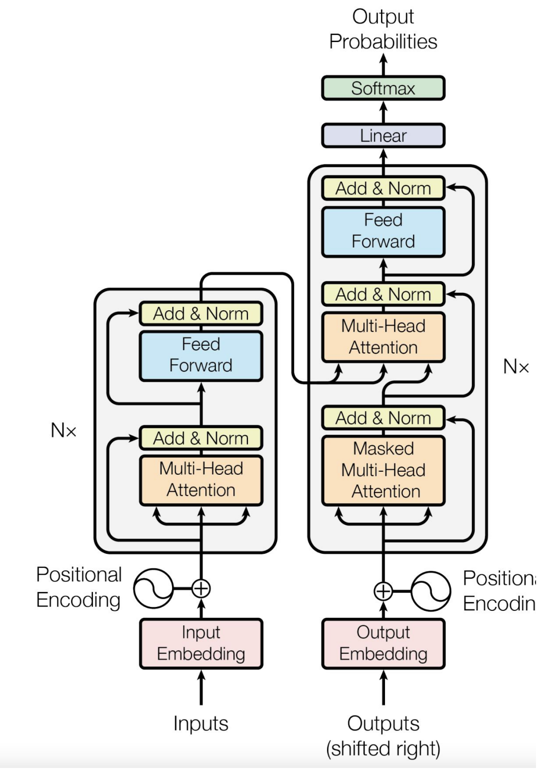


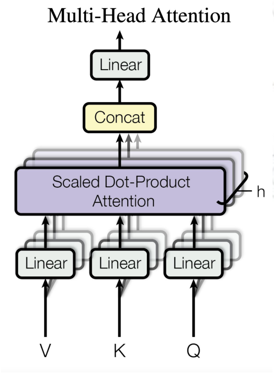
Attention-based Graph Neural Networks

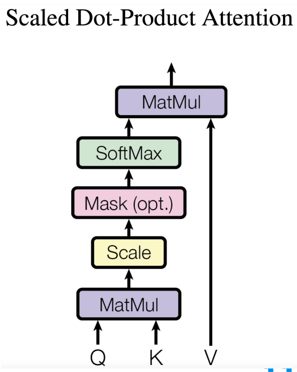
Introducing Transformers

- Transformers architectures have shown state-of-the-art performance in many NLP and vision tasks
- adopts the mechanism of self-attention, differentially weighting the significance of each part of the input data
- not all node's neighbors are equally important
  - Attend to the relevant neighbors

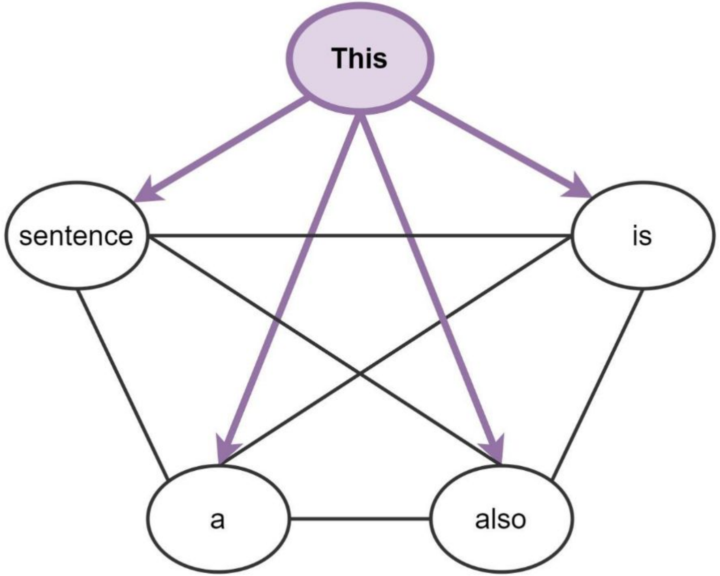
Paper: [Attention is all you Need]()







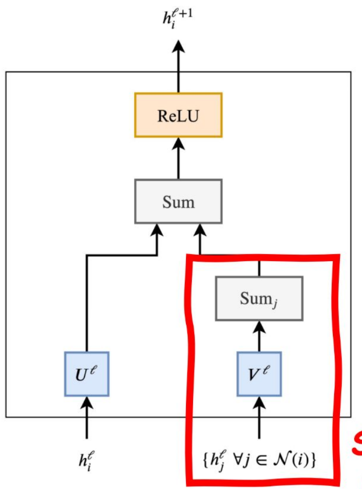
Transformers are Graph Neural Networks



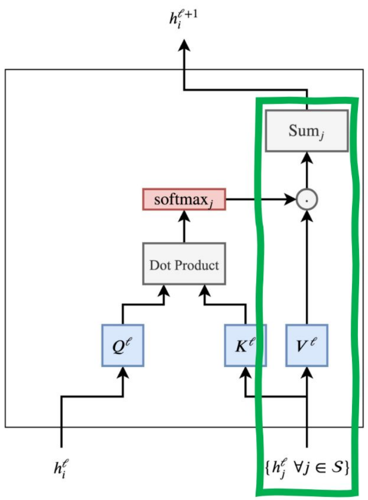
Consider a sentence as a fully connected graph of words...

Blogpost: [Transformers are Graph Neural Networks]

Transformers are Graph Neural Networks



Sum over local neighborhood



Sum over all words in S

$$h_i^{\ell+1} = \sigma\Big(U^\ell h_i^\ell + \boxed{\sum_{j \in \mathcal{N}(i)} \left(V^\ell h_j^\ell\right)\Big)},$$

$$i.\,e.\,,\ h_i^{\ell+1} = \boxed{\sum_{j \in \mathcal{S}} w_{ij}\left(V^\ell h_j^\ell\right),}$$

$$\text{where } w_{ij} = \text{softmax}_j\left(Q^\ell h_i^\ell \cdot K^\ell h_j^\ell\right),$$

Blogpost: Transformers are Graph Neural Networks

Graph Attention Networks

- Certain neighbors to a node are more important than others to its understanding
- Learn attention weights to identify the relevancy of nodes

GCN

$$\boldsymbol{h}_v^k = \sigma(\boldsymbol{W}_k \sum_{u \in \mathrm{N}(v) \cup v} \frac{h_u^{k-1}}{\sqrt{|N(u)||N(v)|}})$$

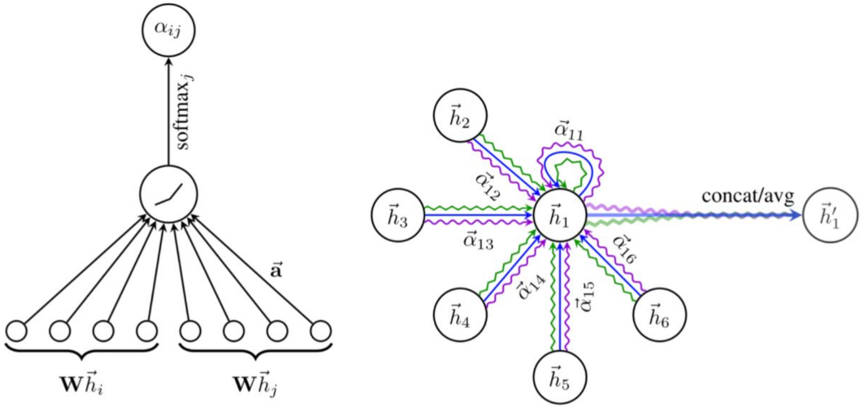
Graph Attention

$$\boldsymbol{h}_v^k = \sigma(\sum_{u \in \mathrm{N}(v) \cup v} \alpha_{v,u} \boldsymbol{W}^k h_u^{k-1})$$

Learned attention weights

Paper: Graph Attention Networks

Graph Neural Networks with Attention



[Figure from Veličković et al. (ICLR 2018)]

- Compared to GCNs
  - More expressive than GCNs
  - Slower than Graph Convolutional Networks

$$\vec{h}'_i = \sigma\left(\frac{1}{K}\sum_{k=1}^{K}\sum_{j\in\mathcal{N}_i}\alpha_{ij}^{k}\mathbf{W}^k\vec{h}_j\right) \qquad \alpha_{ij} = \frac{\exp\left(\text{LeakyReLU}\left(\vec{\mathbf{a}}^T[\mathbf{W}\vec{h}_i\|\mathbf{W}\vec{h}_j]\right)\right)}{\sum_{k\in\mathcal{N}_i}\exp\left(\text{LeakyReLU}\left(\vec{\mathbf{a}}^T[\mathbf{W}\vec{h}_i\|\mathbf{W}\vec{h}_k]\right)\right)}$$

Paper: Graph Attention Networks

*slide from Thomas Kipf, University of Amsterdam

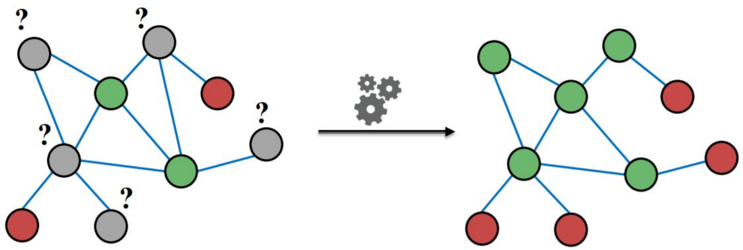
Training GNNs on Unsupervised and Supervised Tasks

Tasks to Learn Node Embeddings with GNNs

Unsupervised Objectives

Use graph structure as supervision

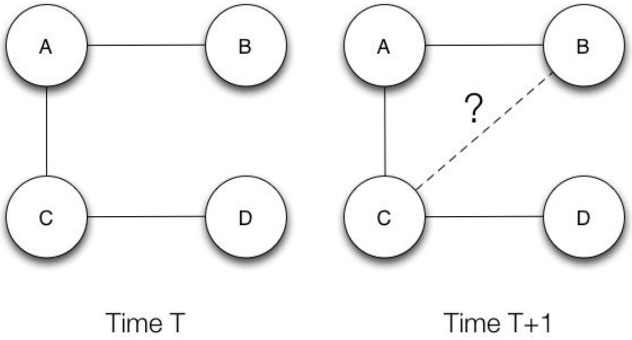
- Predict Node Similarity
  - Random Walk
  - DeepWalk
- Link prediction task
  - Hold-back edges and try to predict



Supervised Objectives

Externally labeled data

- Node classification
- Graph classification



Graph Representations for Recommender Systems

Ying Xiao

Graph embeddings give a dense representation per user and item; how do we incorporate them into recommender systems?

Web Scale Recommender System

- Task: recommend relevant *items* to *users*.
- Web scale: >10^6-10^9 items, >10^9 users.

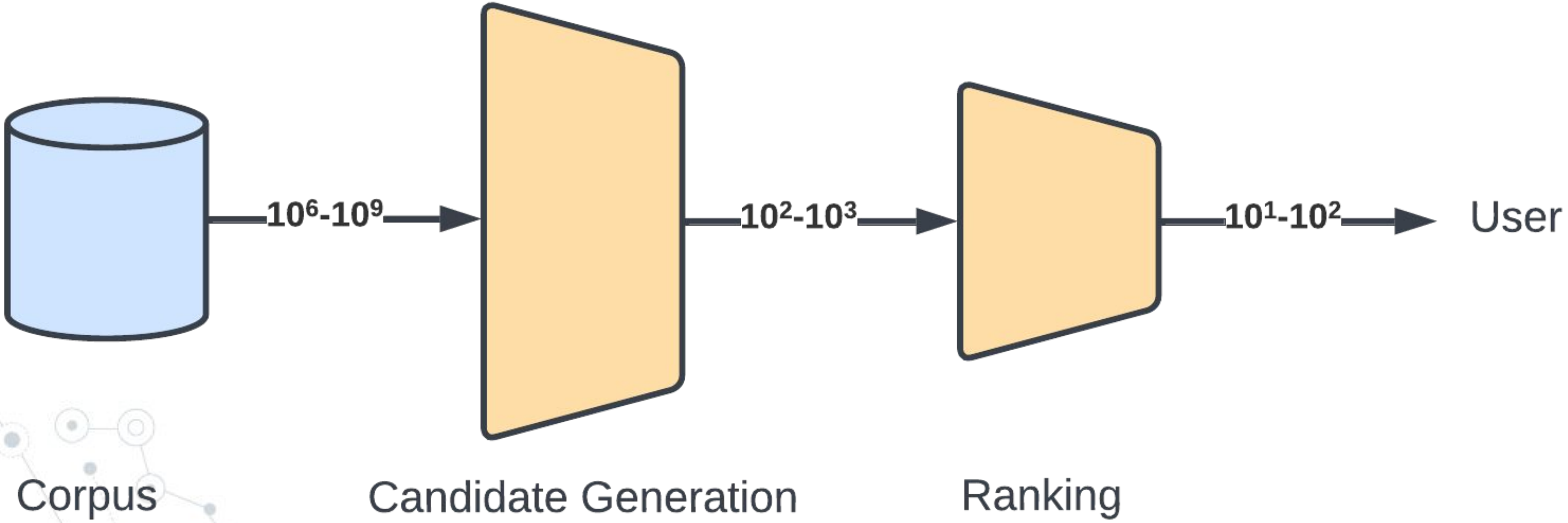
Applications: social media/networks, search, e-commerce, ads, video streaming, etc.

Topic for this session

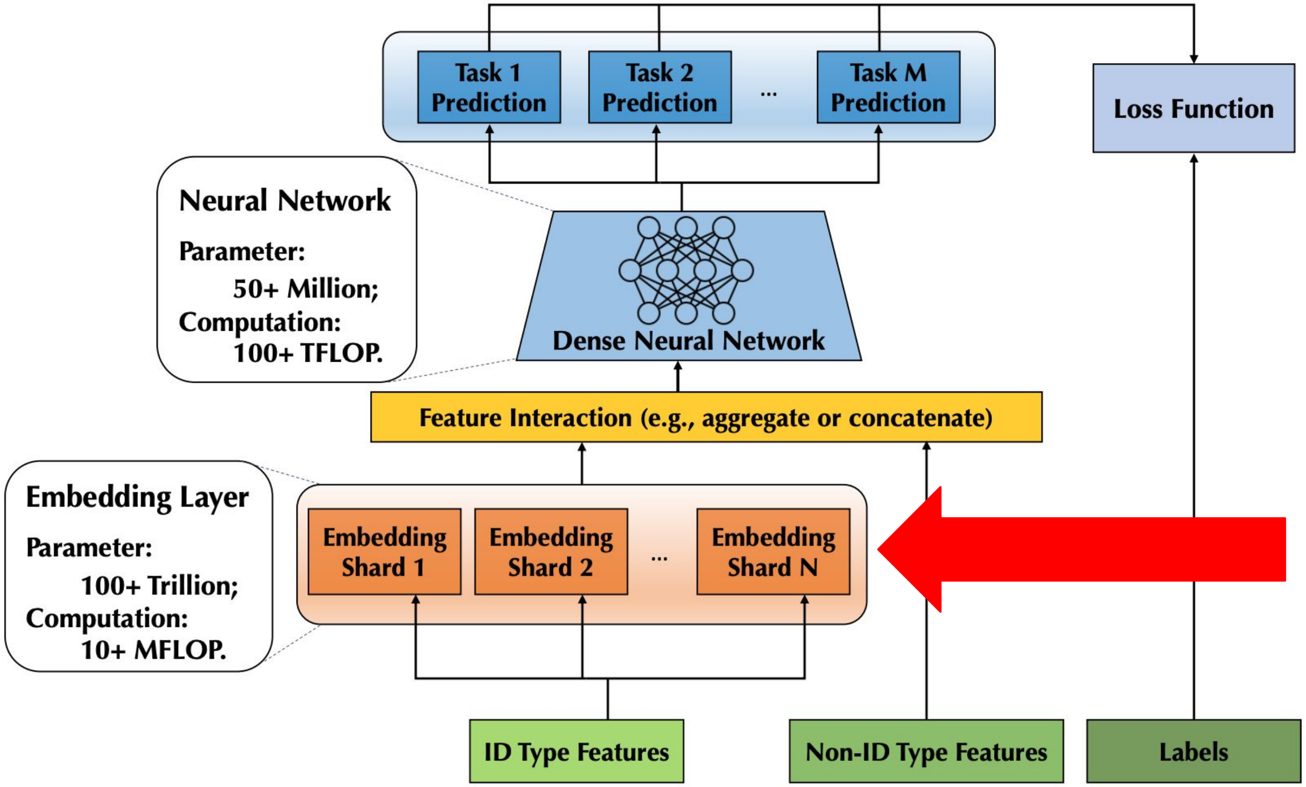
- How to integrate graph embeddings into web scale recommendation systems.
- Not discussed: methods that examine paths/neighbourhoods to directly provide recommendations or refine embeddings.
- See also: A Survey on Knowledge Graph-Based Recommender Systems

Two-stage

- Candidate Generation: retrieval task.
- Ranking models: high precision ranking task.



Ranking models



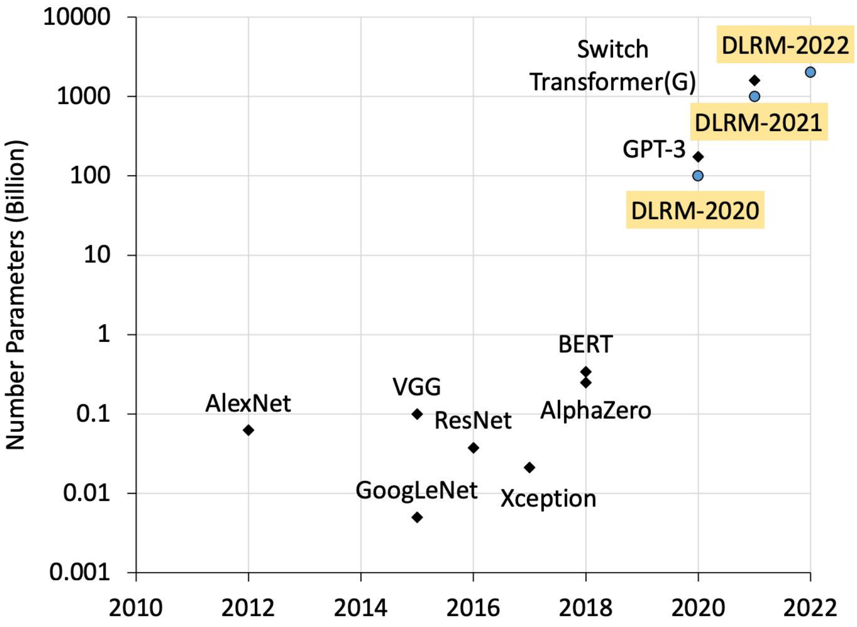
Large trainable embedding already

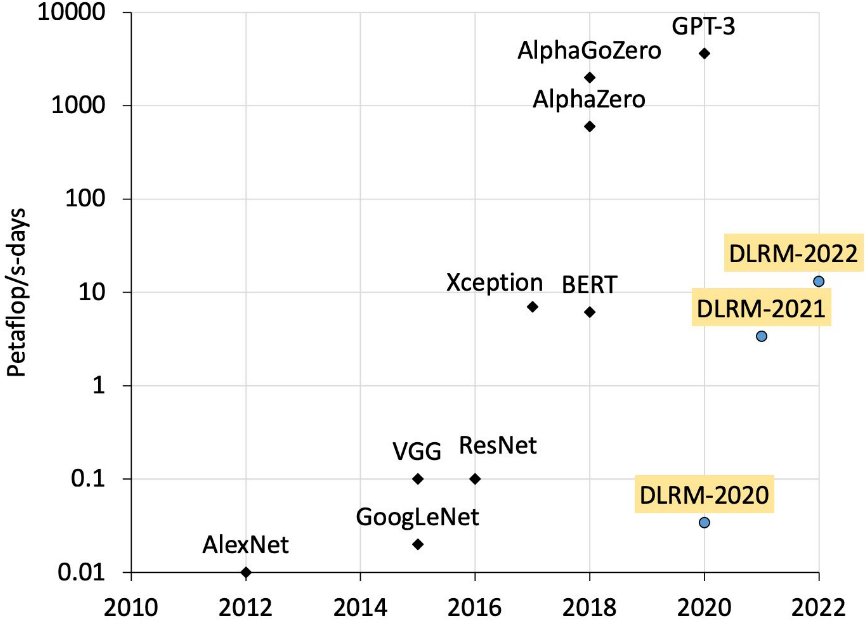
Paper: [Persia: An Open, Hybrid System Scaling Deep Learning-based Recommenders up to 100 Trillion Parameters]

Large sparse features / large trainable embedding tables

- ID features – ids of items a user has previously found relevant – lead to huge tables (10^9-10^{12} params).
- Only recently easily trainable on GPU in torch (torchrec) and TensorFlow (NVidia HugeCTR SOK).

Trainable embeddings: significant infrastructure investment





Paper: Software-Hardware Co-design for Fast and Scalable Training of Deep Learning Recommendation Models

Pre-trained embeddings & end-to-end trained embeddings

Advantages:

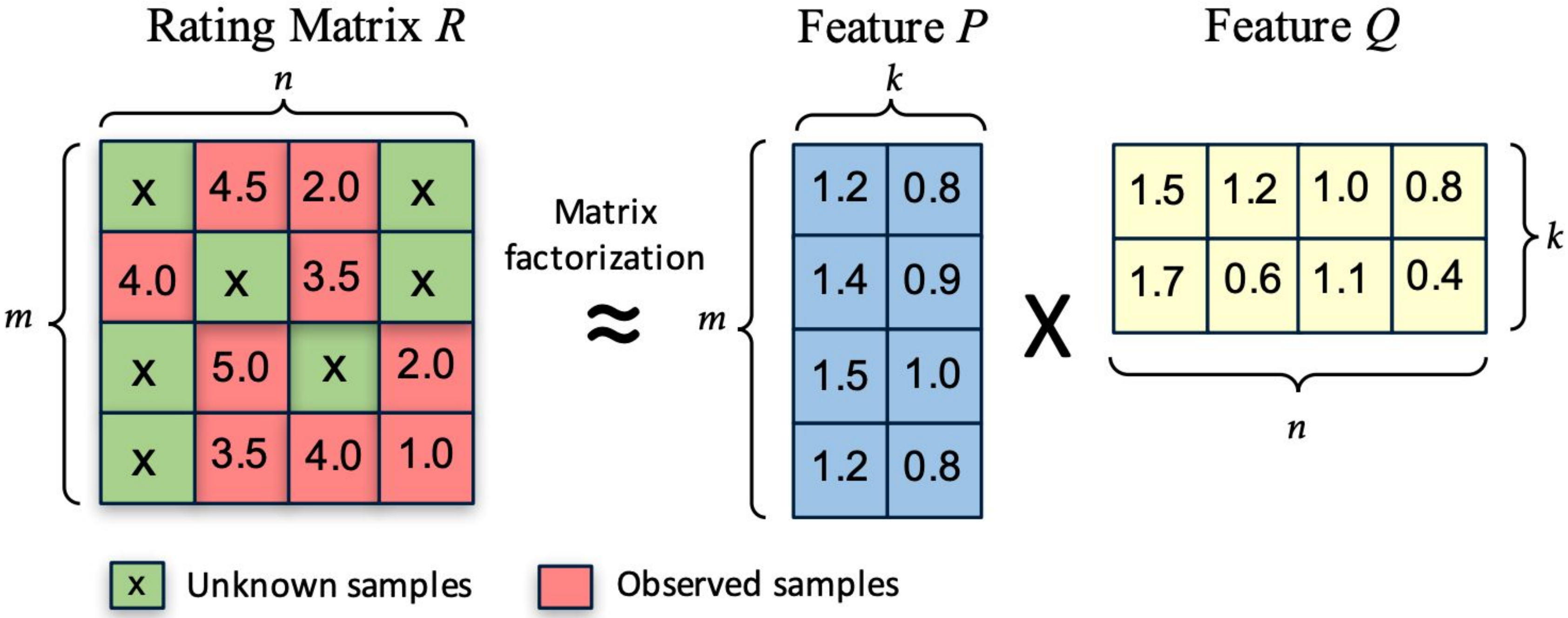
- Infrastructural simplicity.
- Applicability to many tasks.
- Use data from different tasks.

Disadvantages

- Lack of task specificity (i.e., performance).

Pre-trained and end-to-end trained embeddings are NOT mutually exclusive. You probably want both for key applications!

Inductive bias: pairwise interactions between item + user



x	4.5	2.0	x
4.0	x	3.5	x
x	5.0	x	2.0
x	3.5	4.0	1.0

1.2	0.8
1.4	0.9
1.5	1.0
1.2	0.8

1.5	1.2	1.0	0.8
1.7	0.6	1.1	0.4

Paper: CuMF$_{SGD}$: Fast and Scalable Matrix Factorization

Bilinear product

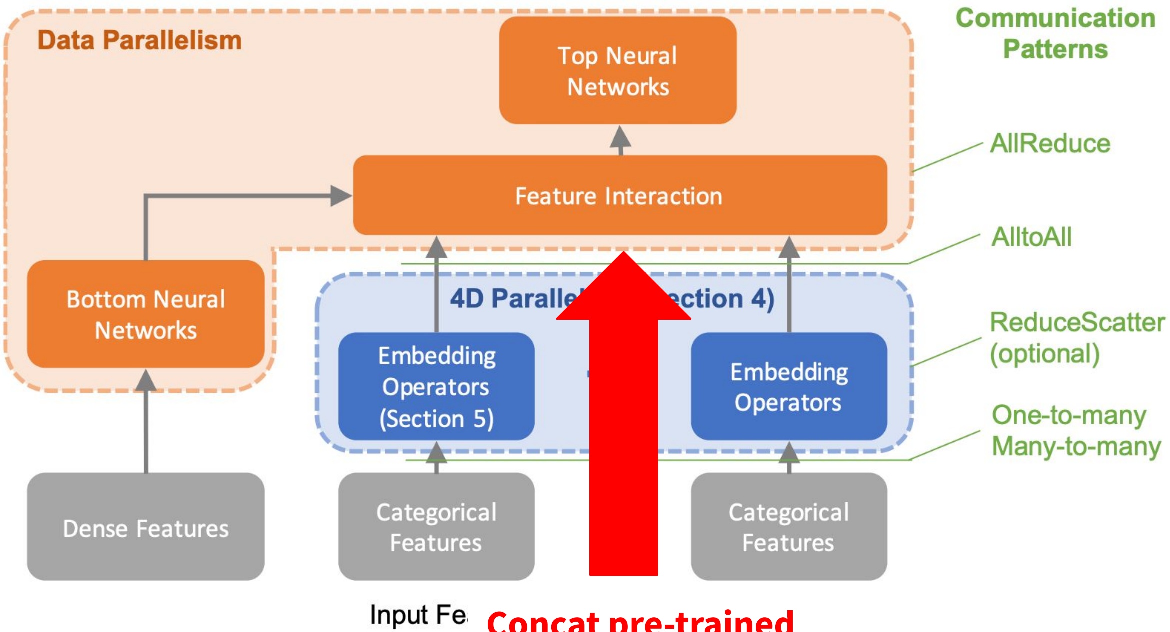
Prediction: <user vector$_i$, item vector$_j$>

Key properties:

- Linear in user vector$_i$, linear in item Q$_j$.
- ~~Output is a single scalar~~.

Intuition: capture interaction in mathematically simple, but still expressive way.

DLRM: Gram matrix of all embeddings for entities.



Concat pre-trained embedding here.

Paper: [Deep Learning Recommendation Model for Personalization and Recommendation Systems]()

DLRM: basic idea

Start with *many* embeddings per user/item pair:

- Project them to the same dimension.
- Compute *all* inner-products of these embeddings.
- Concatenate n choose 2 unique ones with dense inputs.

This makes it easy to add new pre-trained embeddings.

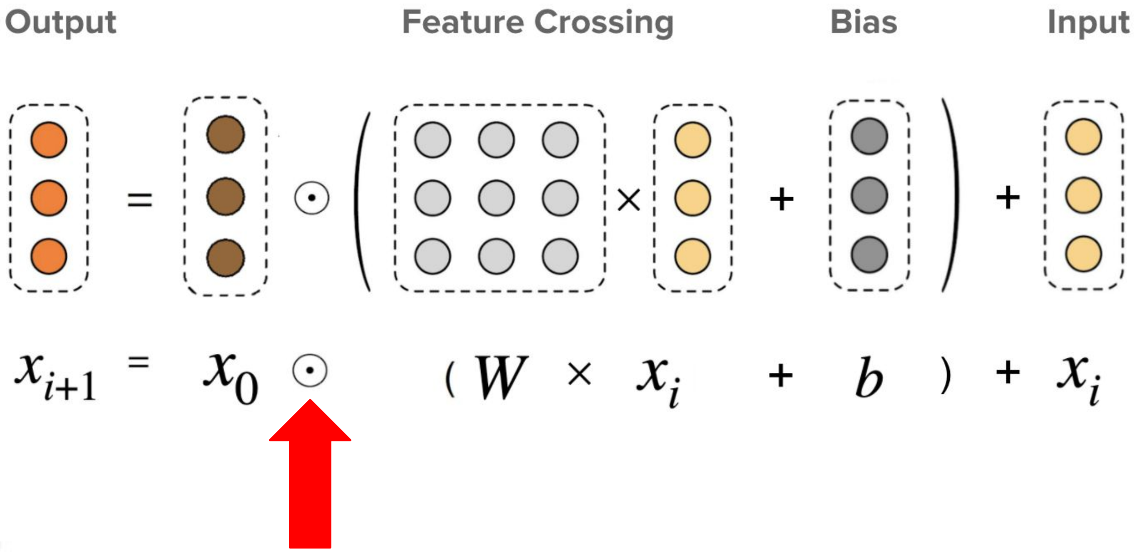
Deep and Cross Network v1 and v2

1. Capture interaction with more than a single scalar.
2. Stack the interaction layers.

Paper: Deep & Cross Network for Ad Click Predictions

Paper: DCN V2: Improved Deep & Cross Network and Practical Lessons for Web-scale Learning to Rank Systems

Interaction layers: more than a single scalar



$$x_{i+1} = x_0 \odot (W \times x_i + b) + x_i$$

Component-wise product

Practical Consideration 1: Normalization

- Most DNNs assume that neurons are approximately mean 0, variance 1 (e.g., batch norm, layer norm, MLP layer initializations).
- Try normalizing pre-trained embeddings before feeding into model

Paper: TwHIN: Embedding the Twitter Heterogeneous Information Network for Personalized Recommendation

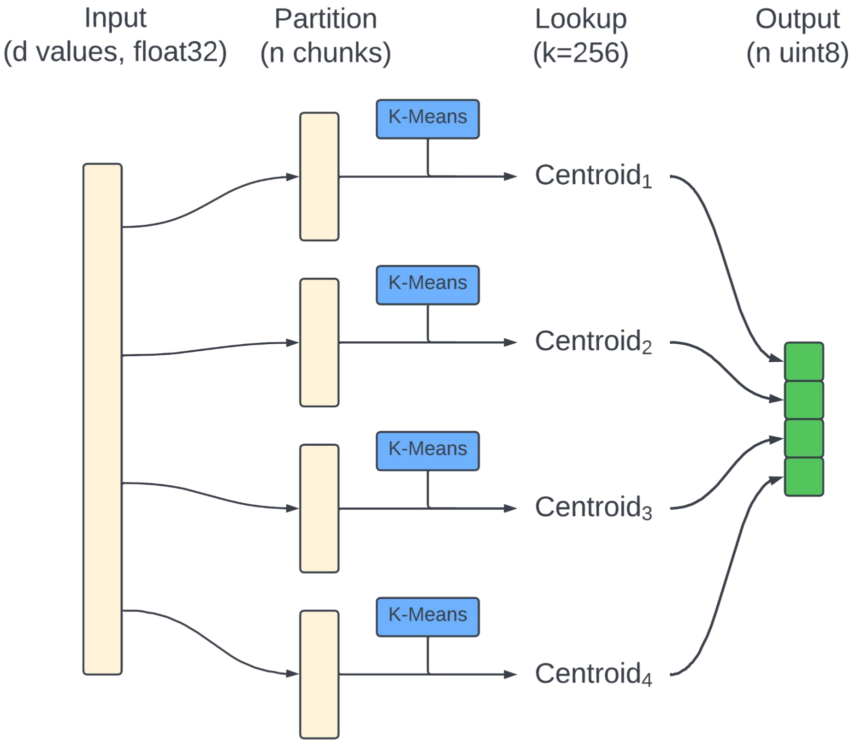
Practical Consideration 2: Space/IO Efficiency

Embeddings can be made very space efficient:

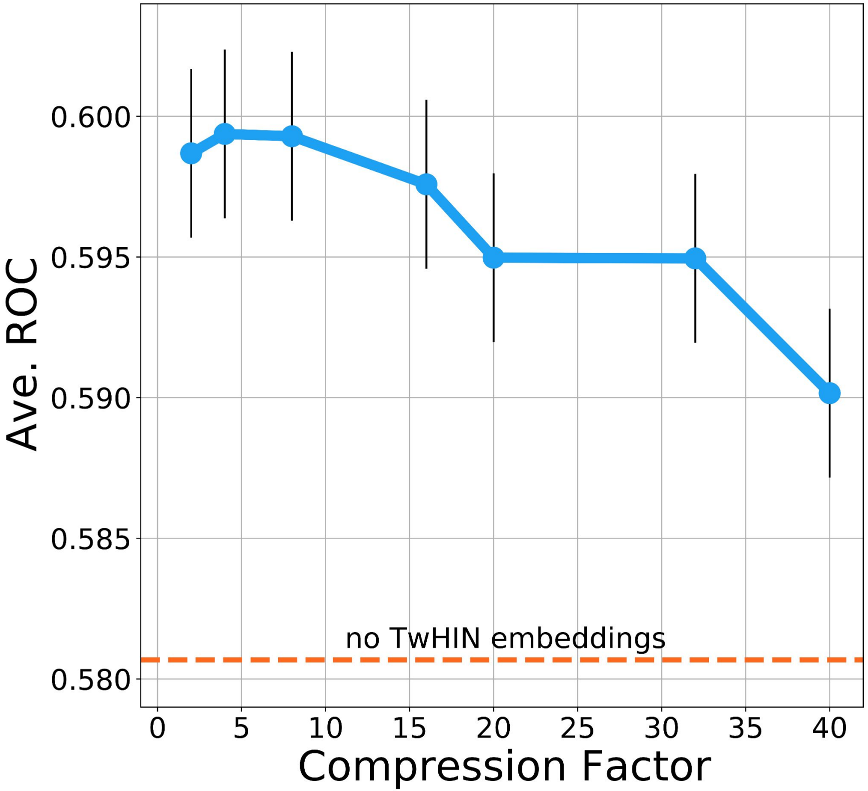
- Compress with product quantization (PQ).
- Large compression ratios (>75%) without affecting downstream task metrics
- Fast implementations in Faiss; decoding trivial.

Paper: *Product Quantization for nearest neighbor search*

Product Quantization



PQ effect on downstream task



Practical Consideration 3: Drift Mitigation

- Over time, we want to retrain the model, but at time *t+1*, don’t want embedding too different from time *t*.
- Principled approach – constrain difference between embeddings at different times.
  - Works well, but doubles memory.
- More efficient approach – initialize training at time t+1 with parameters from time *t*.

Practical Consideration 4: Redundancy with pre-existing features

- For pre-existing models, graph embeddings may be *very* redundant with pre-existing features.
  - Especially when there lots of hand-crafted features with lots of data.
- Limits model improvements when adding graph embeddings.

Redundancy with pre-existing features

This is actually *a desirable situation:*

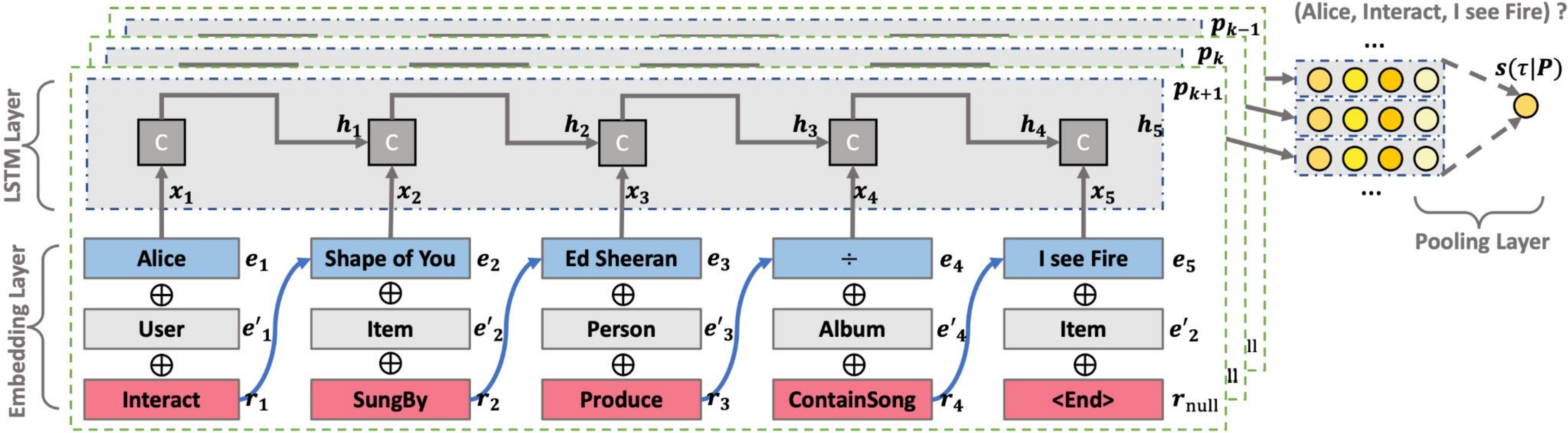
1. Add graph embedding.
2. Run feature selection on pre-existing features.
3. Remove many of them (85% for Twitter use case).
4. Reclaim the IO/compute budget for other model improvements such as scaling up model architecture.

History Aggregation

- Idea: aggregate embeddings of relevant items per user
- Aggregation types: pooling, RNNs, attention.
- Broad in scope: *many* research papers.
- Example: DKN
  - After embedding, run attention between the candidate item and items previously relevant to a user.

Paper: DKN: Deep Knowledge-Aware Network for News Recommendation

Path dependent methods



- Extract paths, run rnn over paths, pool for prediction.

Paper: Explainable Reasoning over Knowledge Graphs for Recommendation

Summary: graph embeddings in ranking models

- Complementary to large trained embeddings, though typically *much* easier to get started with.
- Need to have both user and item representations.
- Plethora of practical tricks to make it work better.

Candidate Generation

Cand Gen Family	Definition	Example
Item-based (content-based)	Using item similarity, query similar items to what a user prefers.	User faves travel tweets, so suggest similar travel tweets.
Collaborative filtering	Suggest preferred items from similar users to a user.	User A and B are similar, A likes travel tweets, so suggest travel tweets to B,

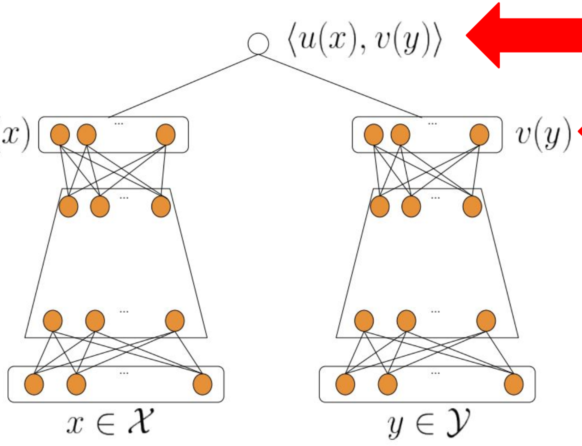
Heuristic and model based candidate generation

- Many candidate generation strategies are heuristics (e.g., most popular/recent items).
- Pre-trained embeddings fall into a family of ML model based techniques.

Model-based Candidate Generation

Approximate nearest neighbor (ANN) based dense retrieval

- Retrieval from an index of items, or
- RS Models factored into two towers:



Replace with ANN

Precompute

Precompute

Figure from Yi et al., [Sampling-bias-corrected neural modeling for large corpus item recommendations,](#) 2019.

Plug and Play

Adding a graph embedding to candidate generation system tends to be straightforward e.g.,

- Take your embeddings, put them in an ANN index, query the ANN index at retrieval time.
- Add graph embedding to a two-tower model.

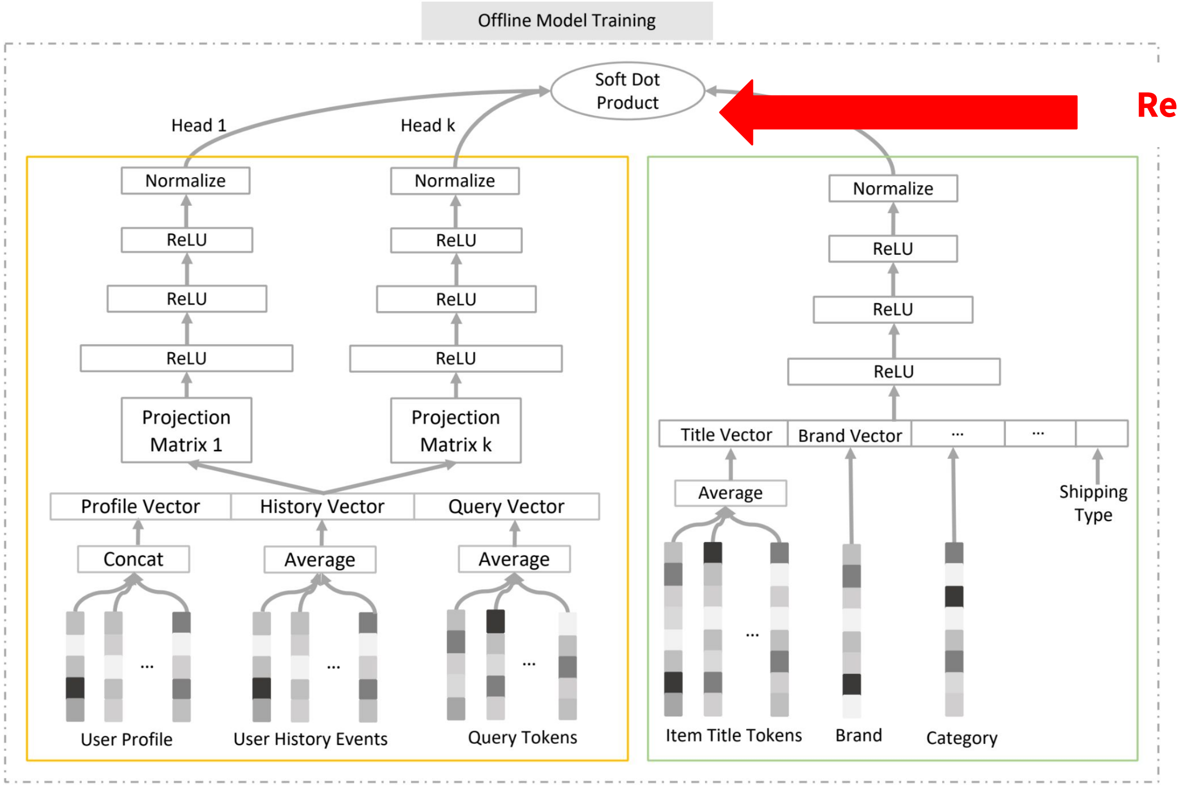
Packages: HNSW, Faiss

k-NN retrieval “Locality implies similarity”

- We retrieve items that are close to a user in embedding space.
- Retrieved items are close in embedding space too.
- => Retrieved items are similar to each other.

When items are too similar→ issues with diversity, multi-modal interests, polysemy in search.

Deep Personalized and Semantic Retrieval (DPSR)



Replace with ANN

Idea: query the kNN index with *k* embeddings.

Paper, *Towards Personalized and Semantic Retrieval: An End-to-End Solution for E-commerce Search via Embedding Learning*

Pinnersage

Given an item embedding, build a user representation:

- Cluster previously relevant items.
- For each cluster, compute the medoid (not centroid).
- For each user, weight the clusters with time decay.

To generate candidates: retrieve from ANN based on 3 medoids, importance sampled.

Paper: PinnerSage: Multi-Modal User Embedding Framework for Recommendations at Pinterest

PinnerSage Results

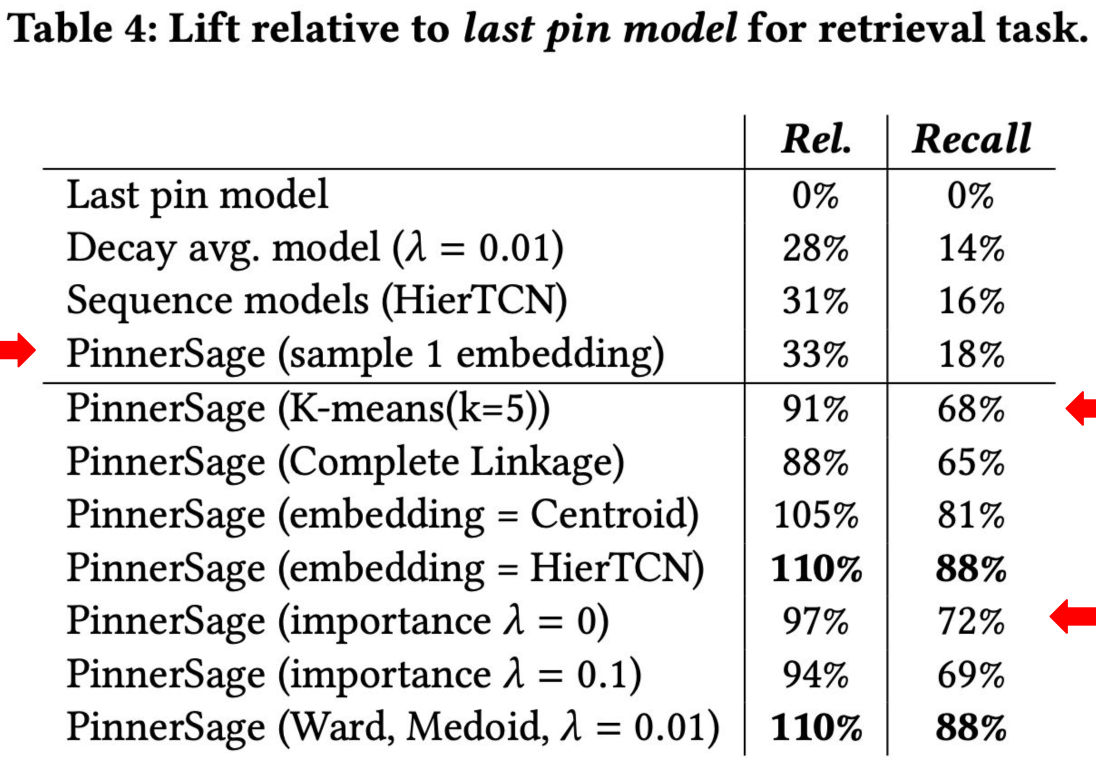


Table 4: Lift relative to *last pin model* for retrieval task.

	Rel.	*Recall*
Last pin model	0%	0%
Decay avg. model ($\lambda = 0.01$)	28%	14%
Sequence models (HierTCN)	31%	16%
PinnerSage (sample 1 embedding)	33%	18%
PinnerSage (K-means(k=5))	91%	68%
PinnerSage (Complete Linkage)	88%	65%
PinnerSage (embedding = Centroid)	105%	81%
PinnerSage (embedding = HierTCN)	**110%**	**88%**
PinnerSage (importance $\lambda = 0$)	97%	72%
PinnerSage (importance $\lambda = 0.1$)	94%	69%
PinnerSage (Ward, Medoid, $\lambda = 0.01$)	**110%**	**88%**

Item Clustering →

← Multiple Queries

← Tuning time decay

k-NN Embed: Multiple querying on top of a kNN system

- (Globally) cluster all the items in your embedding.
- Model each user as a mixture over item clusters:

$$p(\text{item}|\text{user}) = \sum_{\text{cluster}} p(\text{cluster}|\text{user}) \cdot p(\text{item}|\text{user}, \text{cluster})$$

- Idea: data per user is sparse, so use data from adjacent users since we know they're similar.

Paper: *kNN-Embed: Locally Smoothed Embedding Mixtures For Multi-interest Candidate Retrieval*

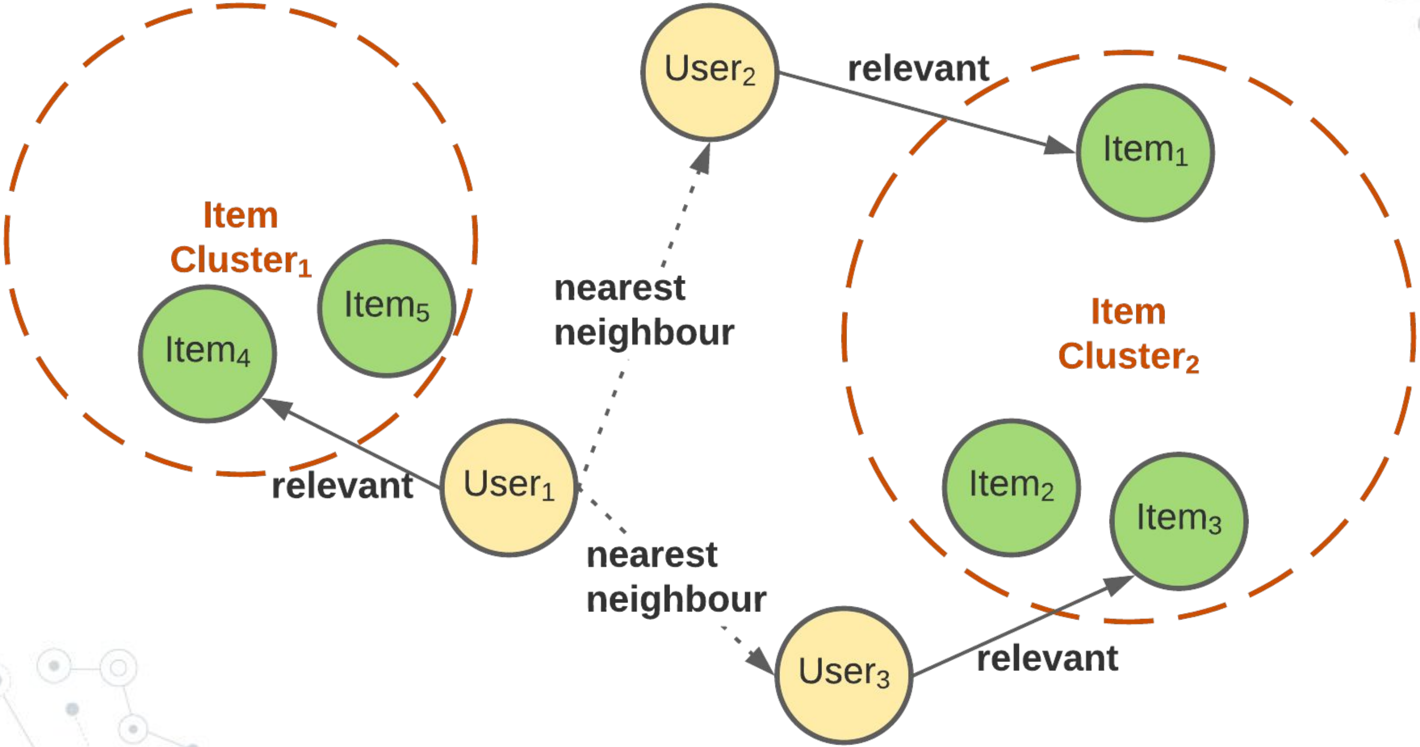
k-NN Embed:

User's preference over clusters: smooth this with neighboring users' preference over clusters.

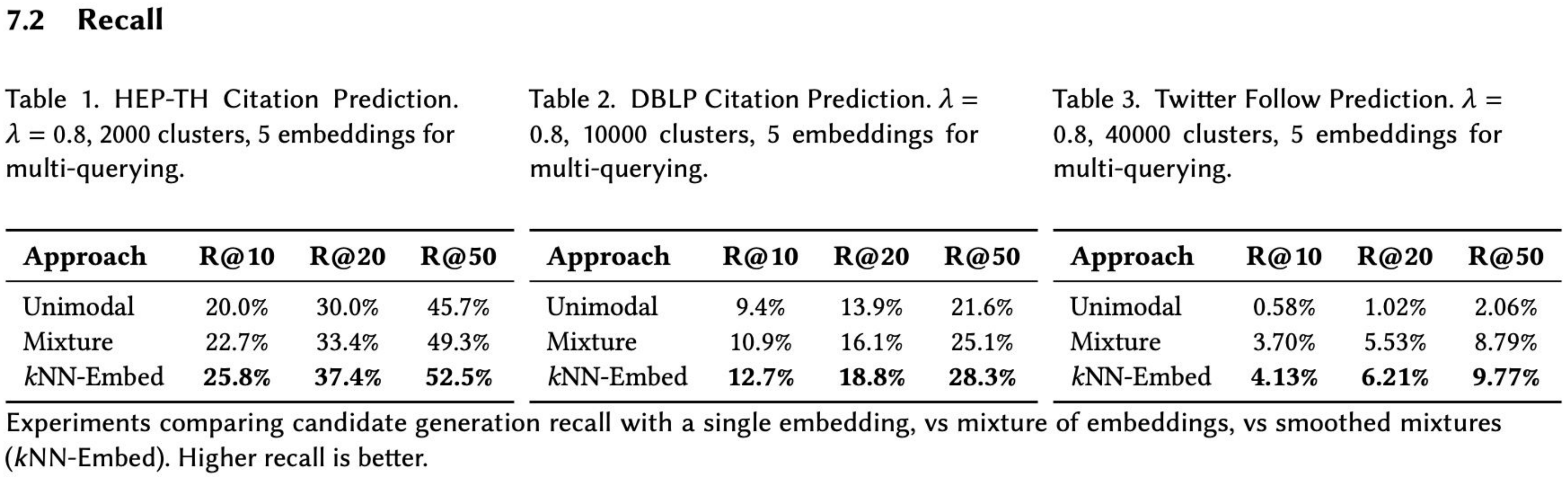
$$p(\text{item}|\text{user}) = \sum_{\text{cluster}} p(\text{cluster}|\text{user}) \cdot p(\text{item}|\text{user}, \text{cluster})$$

ANN retrieval – query from the centroid of this user in the cluster smoothed with centroids of neighbouring users

k-NN Embed: Expand ANN search by using similar users



k-NN Embed: Improvements in diversity



7.2 Recall

Table 1. HEP-TH Citation Prediction. $\lambda = 0.8$, 2000 clusters, 5 embeddings for multi-querying.

Approach	R@10	R@20	R@50
Unimodal	20.0%	30.0%	45.7%
Mixture	22.7%	33.4%	49.3%
*k*NN-Embed	**25.8%**	**37.4%**	**52.5%**

Table 2. DBLP Citation Prediction. $\lambda = 0.8$, 10000 clusters, 5 embeddings for multi-querying.

Approach	R@10	R@20	R@50
Unimodal	9.4%	13.9%	21.6%
Mixture	10.9%	16.1%	25.1%
*k*NN-Embed	**12.7%**	**18.8%**	**28.3%**

Table 3. Twitter Follow Prediction. $\lambda = 0.8$, 40000 clusters, 5 embeddings for multi-querying.

Approach	R@10	R@20	R@50
Unimodal	0.58%	1.02%	2.06%
Mixture	3.70%	5.53%	8.79%
*k*NN-Embed	**4.13%**	**6.21%**	**9.77%**

Experiments comparing candidate generation recall with a single embedding, vs mixture of embeddings, vs smoothed mixtures (*k*NN-Embed). Higher recall is better.

Summary: graph embedding in candidate generation

- Plays nice with ANN based candidate generation.
- Multiple querying, and more sophisticated techniques, allow us increase diversity in retrieved candidates.

Thanks!

Any questions?

Come chat with us about our KDD 2022 Applied Data Science Paper!

Paper: TwHIN: Embedding the Twitter Heterogeneous Information Network for Personalized Recommendation

Poster Session: Monday, August 15, 7:00 pm to 8:30 pm.

Oral: Thursday, August 18, 10:00 AM-12:00 PM (~10:50 AM), Room 3 (Graph Learning & Social Network).