Building & Mining Knowledge Graphs

(KEN4256)

Lecture 7: Knowledge Graph Embeddings

Maastricht University

Institute of Data Science

id: KEN4256_L7
version: 1.2024.0
created: February 9, 2019
last modified: March 26, 2024
published on: March 26, 2024

Knowledge Graph Embeddings (KGEs)

- KG are **symbolic** representations of entities and relations, which is favourable for *logical inference*, question answering, and information retrieval
- **KG embeddings** encode the KGs into low-dimensional numerical vectors, which can be used as features for ML (*statistical inference*)

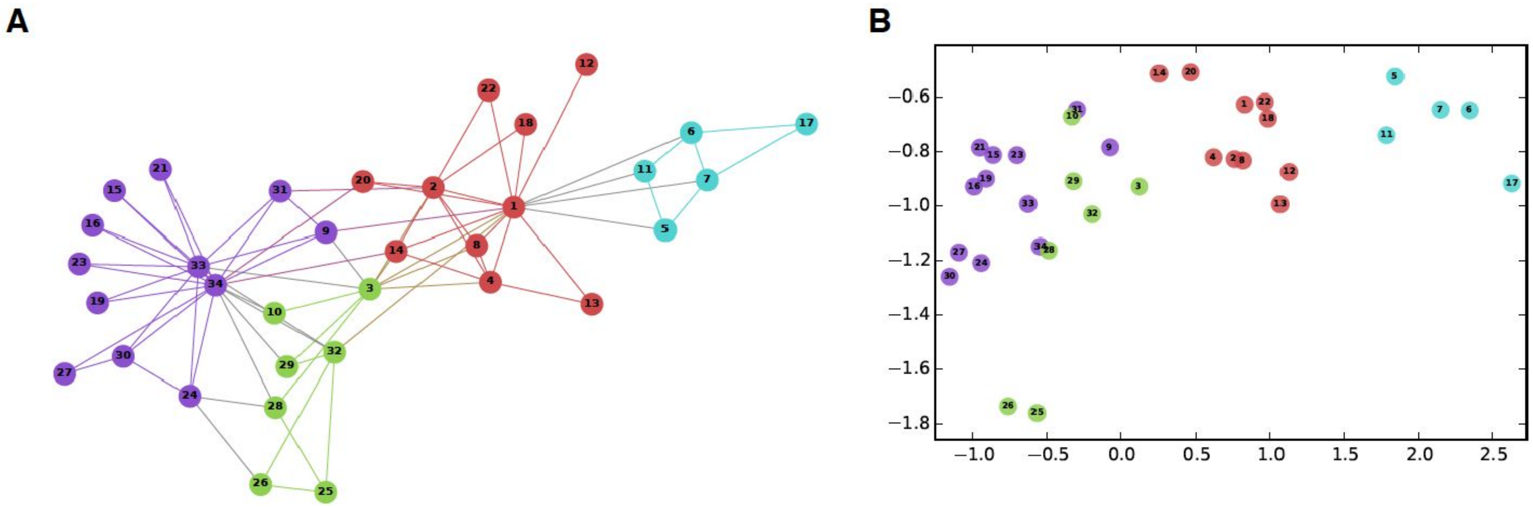
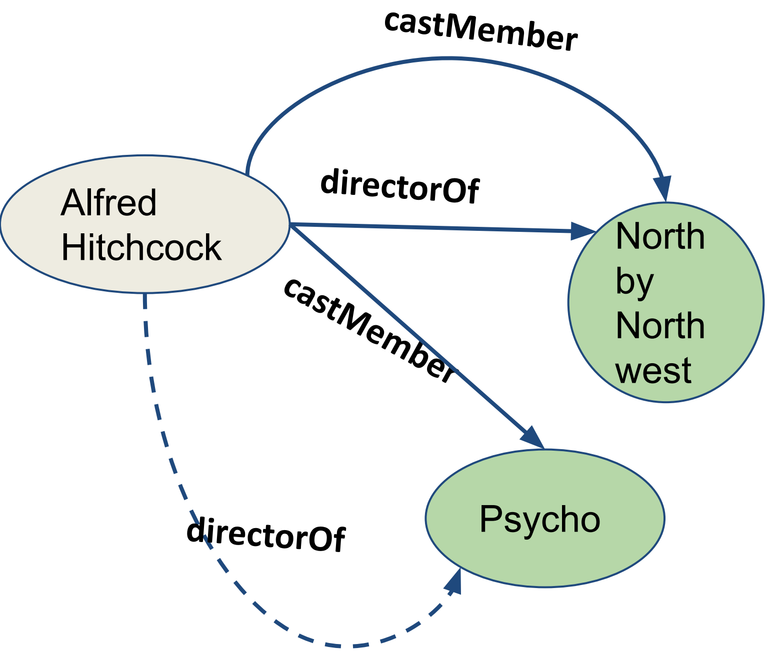


image source: Hamilton, W.L., Ying, R. and Leskovec, J., 2017. Representation learning on graphs: Methods and applications. *arXiv preprint arXiv:1709.05584*.

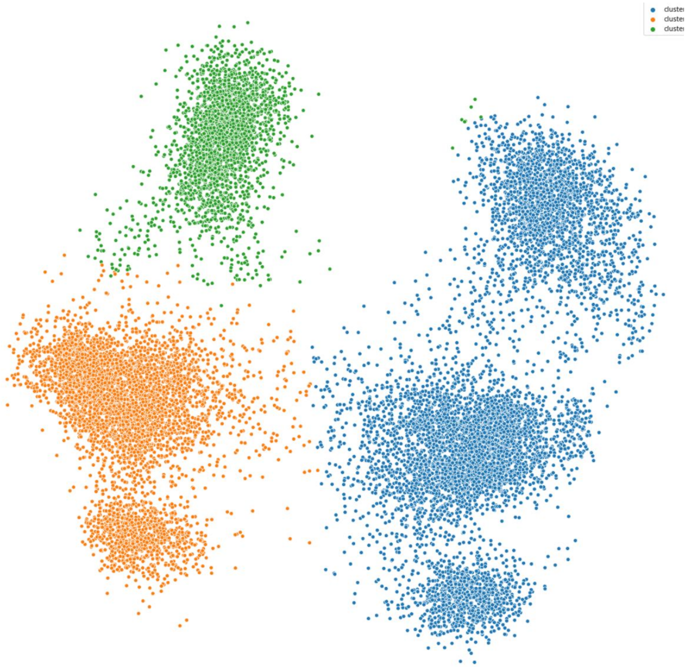
Knowledge Graph Completion Tasks



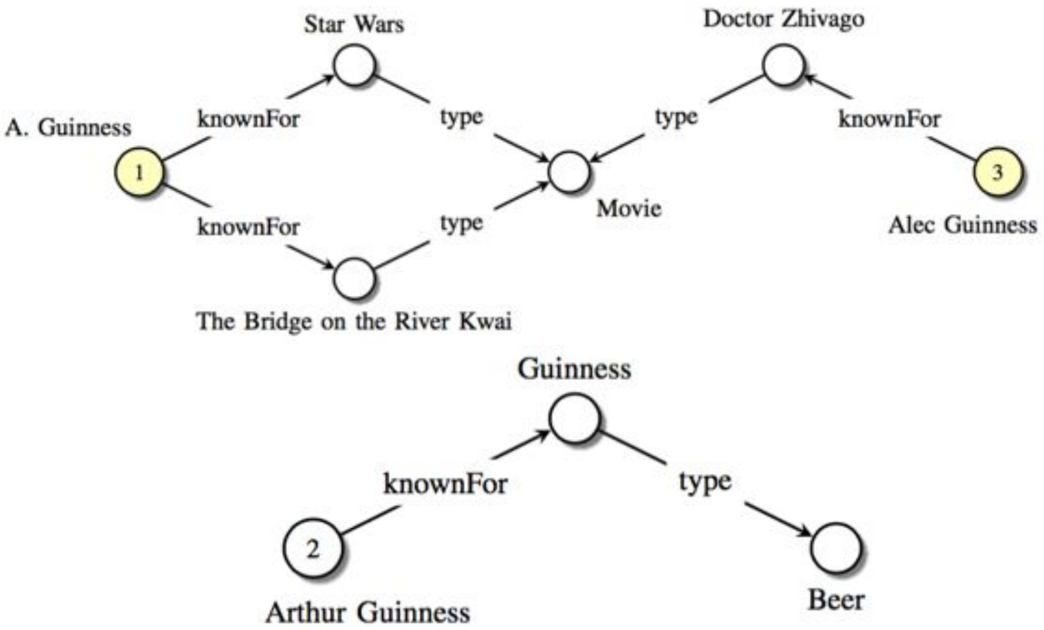
- **Node Classification**
  - categorize nodes to their semantic type: `AlfredHitchcock is a Person`
- **Link Prediction**
  - predict **s** given (**p, o**) or **o** given (**s, p**): `(?s, directorOf, psycho)`
- **Triple Classification**
  - Verify whether an unseen triple fact (**s, p, o**) is true or not: `(AlfredHitchcock, directorOf, psycho) is True or False?`

KGE Applications

Entity Clustering & Classification



Entity Alignment

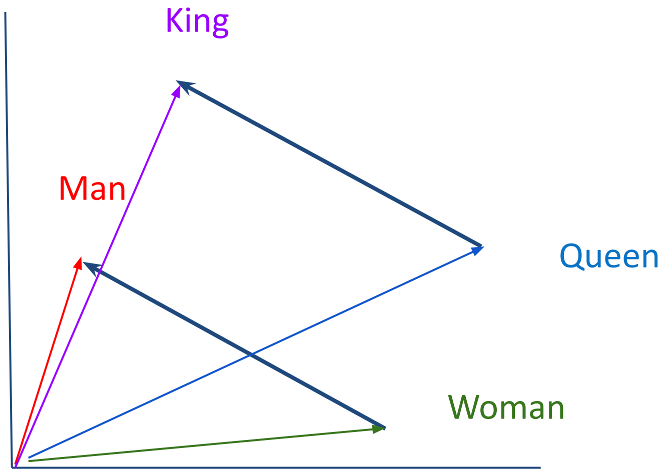


Slide: https://kge-tutorial-ecai2020.github.io/

Low-dimensional vector embeddings

Map high-dimensional data into low-dimensional vector space.

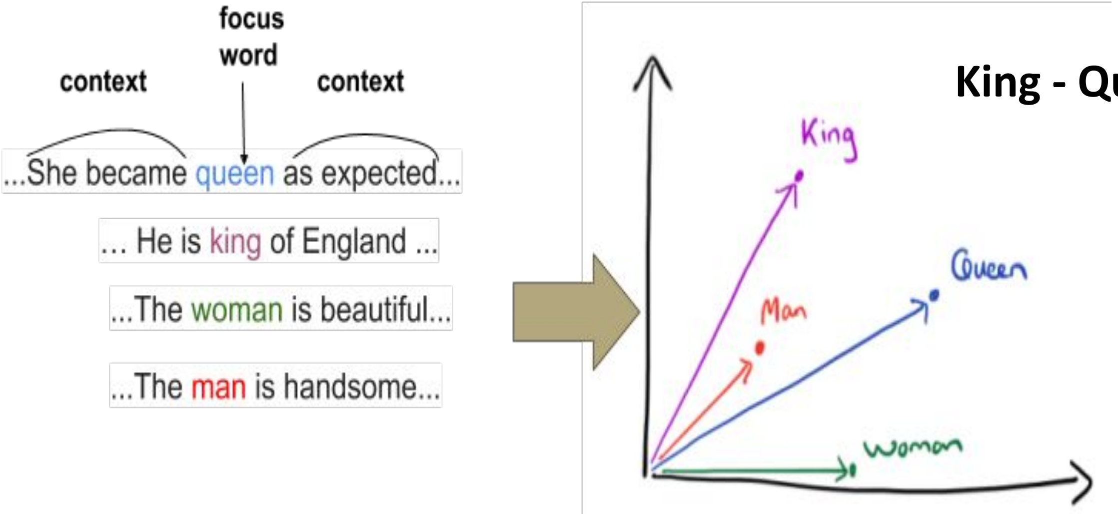
- Efficient representation
- Allow vector operations (addition, subtraction, cosine, etc.)
- Capture semantic relationships



King - Queen = Man - Woman

word2vec

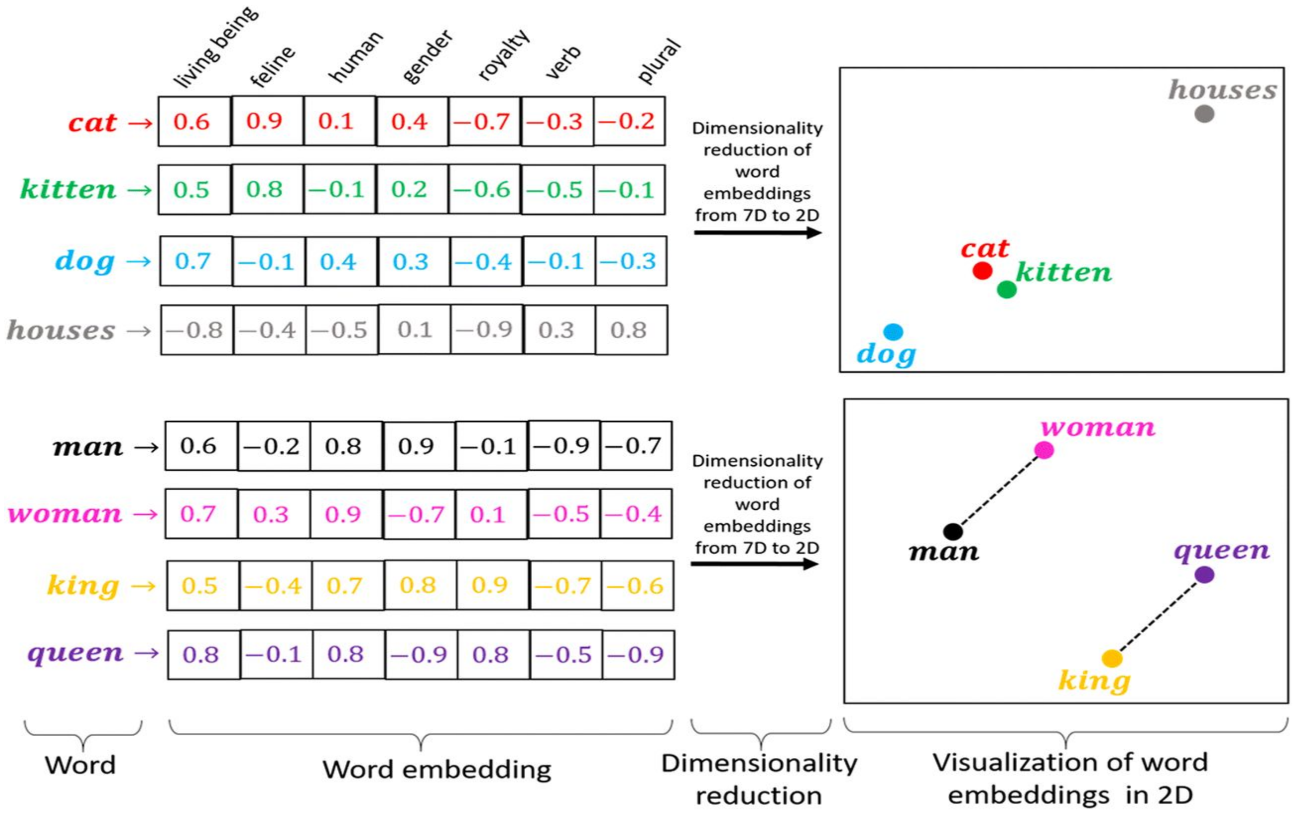
Represents each word with a low-dimensional vector, called word embedding where **semantically** and **syntactically** closer words appear closer in the vector space.



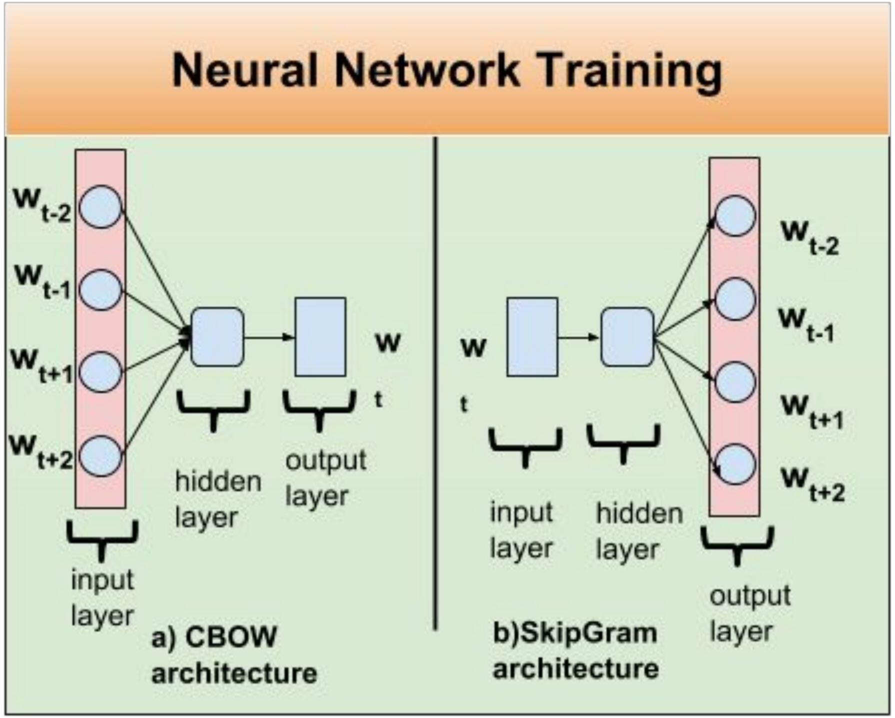
King - Queen = Man - Woman

Distributed Representations of Words and Phrases and their Compositionality, Mikolov T. etl., 2013.

Word Embeddings



word2vec architectures

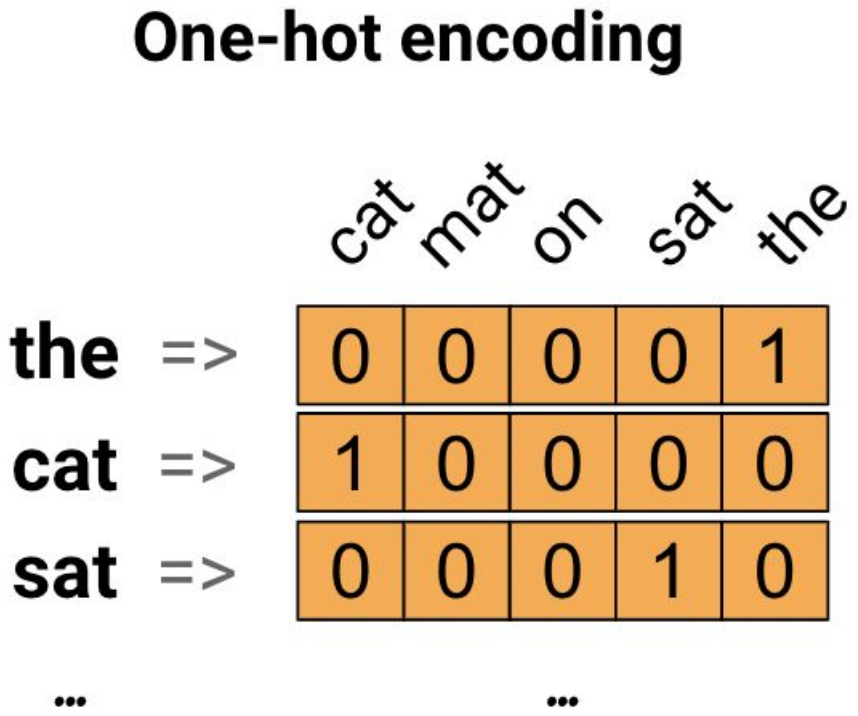


2 basic neural network models:

- **Continuous Bag of Word (CBOW):**
  - use a window of word to predict the target word
- **Skip-Gram (SG):**
  - use a word to predict the surrounding ones in window.

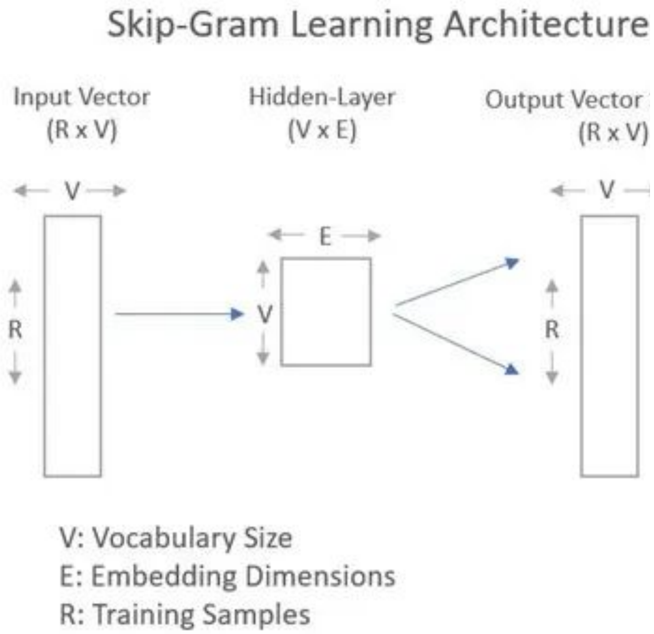
One-hot encoding for text data

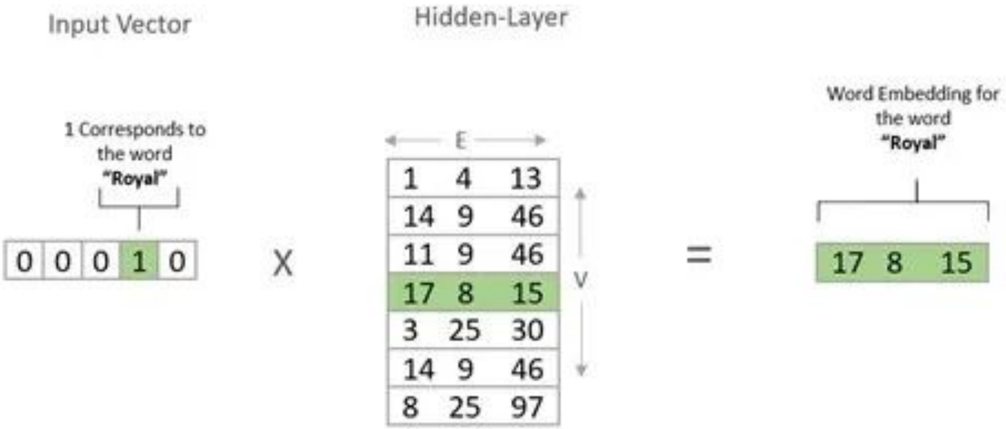
Sentence:
"The cat sat on the mat".



https://www.tensorflow.org/text/guide/word_embeddings

Skip-Gram Algorithm





Source: https://www.hackdeploy.com/word2vec-explained-easily/

Knowledge Graph Embedding

Word
I
like
apple

Input

Word2vec →

Word	Embedding
I	[0.2, …, 0.5]
like	[0.3, …, 0.8]
apple	[0.9, …, 0.2]

Output

Semantic Triple
(skytree, location, tokyo)
(Satya Nadella, CEO, Microsoft)
(Steve Jobs, founder, Apple)

Input

Knowledge Embedding →

Entity/Relation	Embedding
skytree	[0.1, …, 0.5]
location	[0.4, …, 0.9]
tokyo	[0.5, …, 0.2]
Satya Nadella	[0.1, …, 0.5]
CEO	[0.6, …, 0.8]
Microsoft	[0.3, …, 0.7]
Steve Jobs	[0.4, …, 0.6]
founder	[0.4, …, 0.3]
Apple	[0.7, …, 0.2]

Output

Source: https://towardsdatascience.com/summary-of-translate-model-for-knowledge-graph-embedding-29042be64273

KG Representation in Vector Space

- **Key Idea**: Model entities and relations in embedding space
- Edges in KG are represented as triple or 3-**tuple (h,r,t)** denoting the head (subject), relation (predicate), and tail (object).
  - Given a triple (h,r,t), the goal is that the embedding of (h,r) should be close to the embedding of t
- How to **embed** (h,r) ?
- How to define **score** $f_r(h,t)$?
  - Score f_r is high if (h,r,t) exists, else f_r is low

Structure for KGE Models

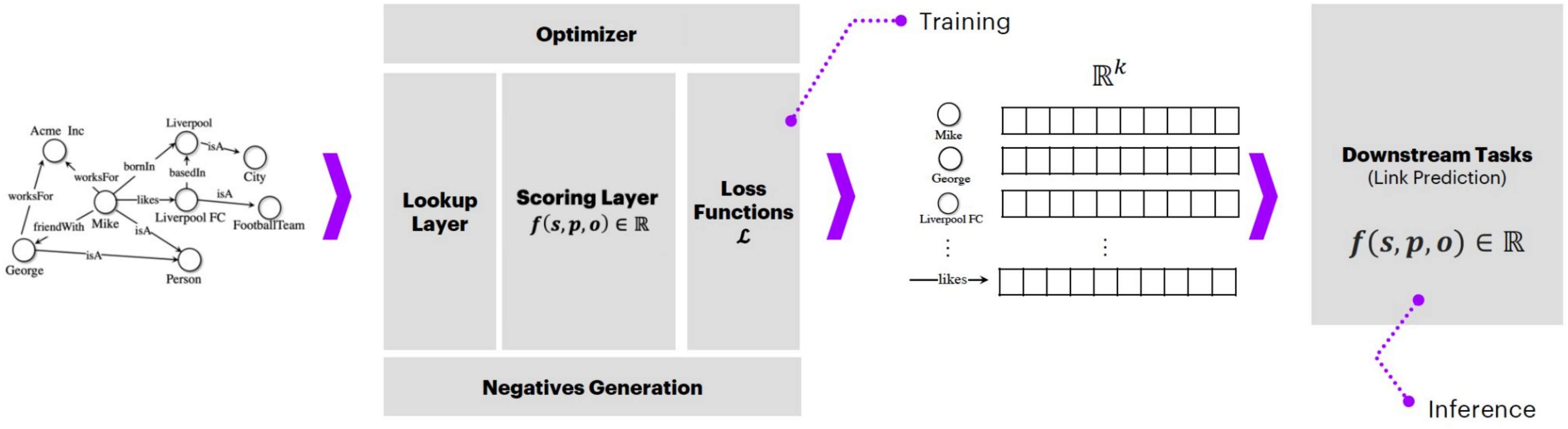


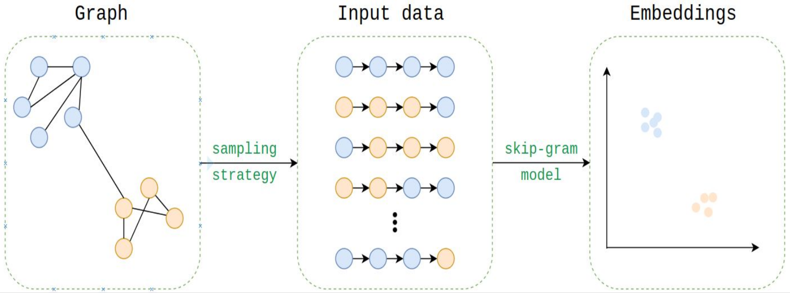
Image: https://kge-tutorial-ecai2020.github.io/

KGE Approaches

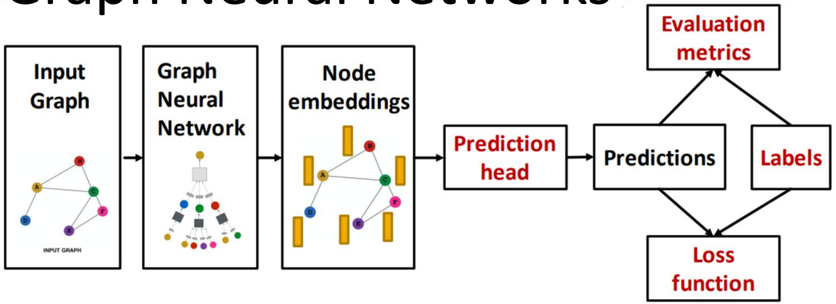
We focus on GNN in this course!

Random Walk based Embeddings

RDF2Vec



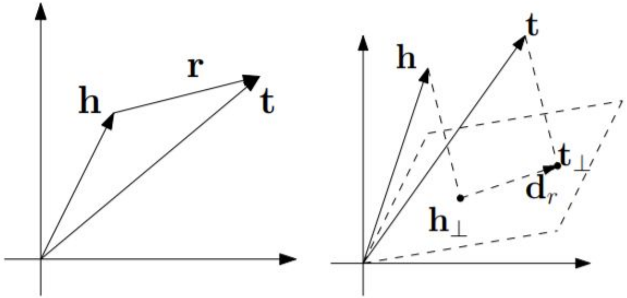
Graph Neural Networks



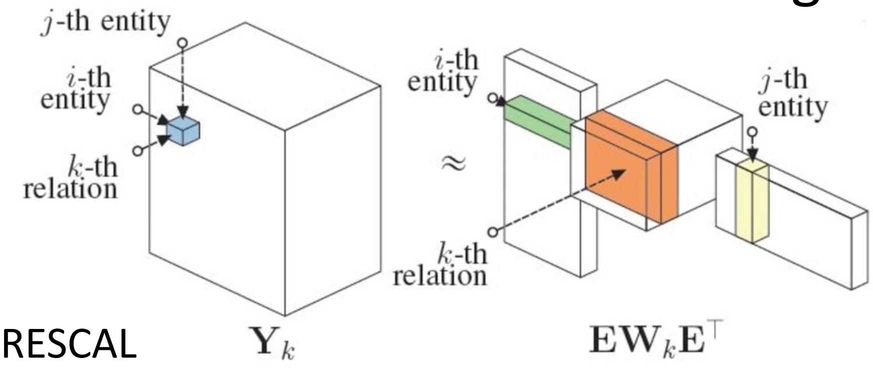
Translation based Embeddings

TransE

RotatE

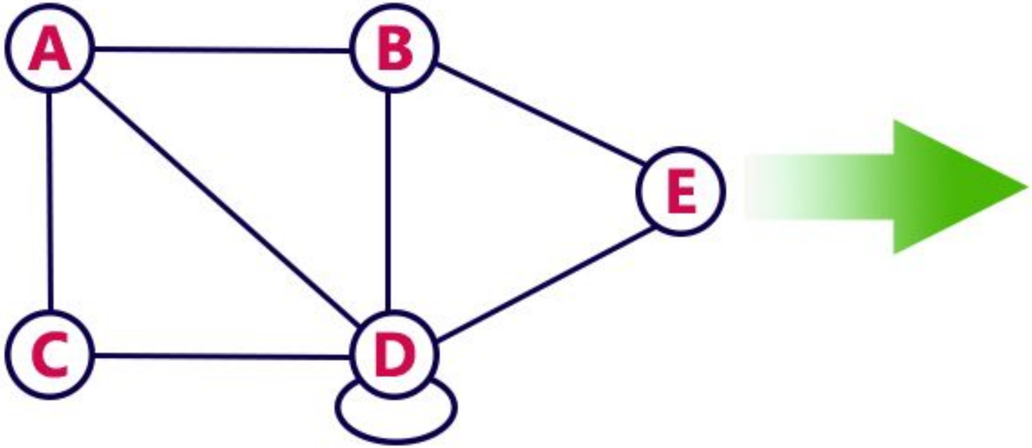


Factorization based Embeddings



RESCAL

One-hot encoding for graph data



$$
\begin{array}{c|ccccc|}
 & A & B & C & D & E \\
\hline
A & 0 & 1 & 1 & 1 & 0 \\
B & 1 & 0 & 0 & 1 & 1 \\
C & 1 & 0 & 0 & 1 & 0 \\
D & 1 & 1 & 1 & 1 & 1 \\
E & 0 & 1 & 0 & 1 & 0
\end{array}
$$

http://www.btechsmartclass.com/data_structures/graph-representations.html

Low-dimensional vector

$$
\begin{array}{c|c}
 & \begin{array}{ccccc} A & B & C & D & E \end{array} \\
\begin{array}{c} A \\ B \\ C \\ D \\ E \end{array} &
\begin{pmatrix}
0 & 1 & 1 & 1 & 0 \\
1 & 0 & 0 & 1 & 1 \\
1 & 0 & 0 & 1 & 0 \\
1 & 1 & 1 & 1 & 1 \\
0 & 1 & 0 & 1 & 0
\end{pmatrix}
\end{array}
$$

representational vectors

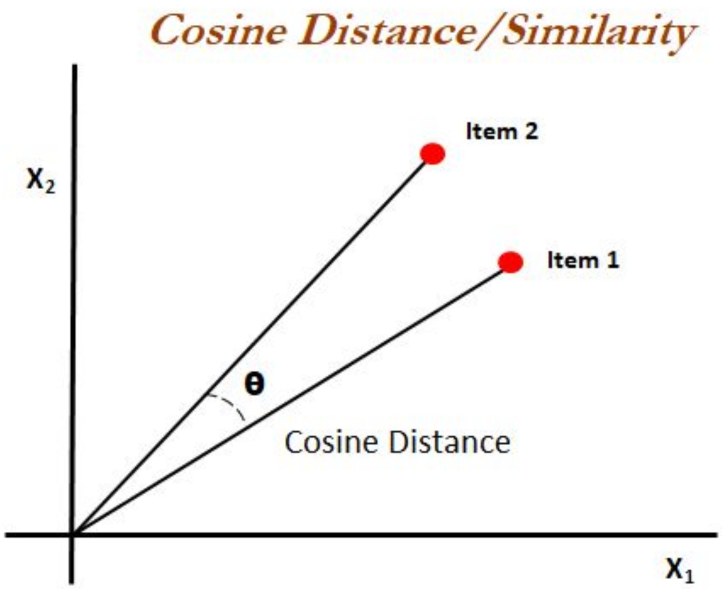
operational vectors

A	0.6	0.1	-0.7	0.5
B	0.3	0.2	-0.4	0.3
C	0.5	0.3	-0.6	0.6
D	0.8	0.4	-0.1	0.9
E	0.7	0.8	-0.2	0.2

one-hot vector representation of entities
sparse vector
vector length = total number of entities

vector representation produced by an embedding algorithm
dense vector
vector length << total number of entities

Scoring function: cosine similarity

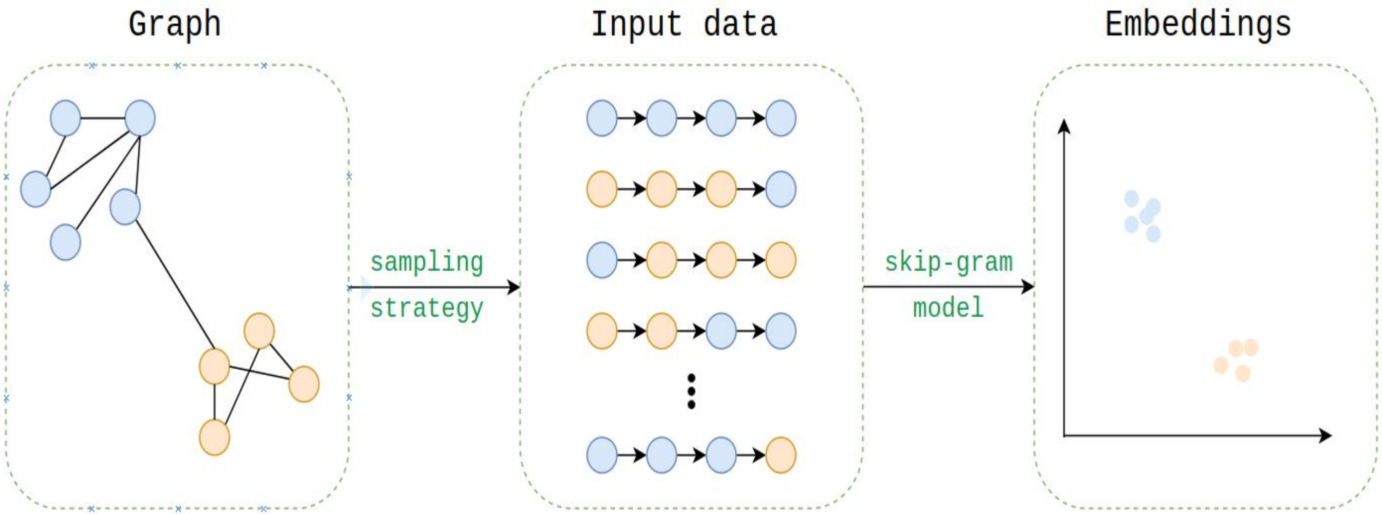


- Let us consider the vector space $\mathbb{R}^n$ over $\mathbb{R}$ for $n = 2$.
- This space is the well-known 2D Euclidean space.
- Each point on the plane is the edge of vector that has its start on the origin of the plane.
- A widely adopted metric used to calculate the similarity (closeness) of two vectors is the cosine of the angle θ between the two vectors.
- Let $u, v \in \mathbb{R}^2$, where $u = (u_1, u_2)$ & $v = (v_1, v_2)$

we have that… $sim(u, v) = cos\theta = \frac{u \cdot v}{||u|| \cdot ||v||} = \frac{\sum_{i=1}^{2} u_i v_i}{\sqrt{\sum_{i=1}^{2} u_i^2}\sqrt{\sum_{i=1}^{2} v_i^2}}$

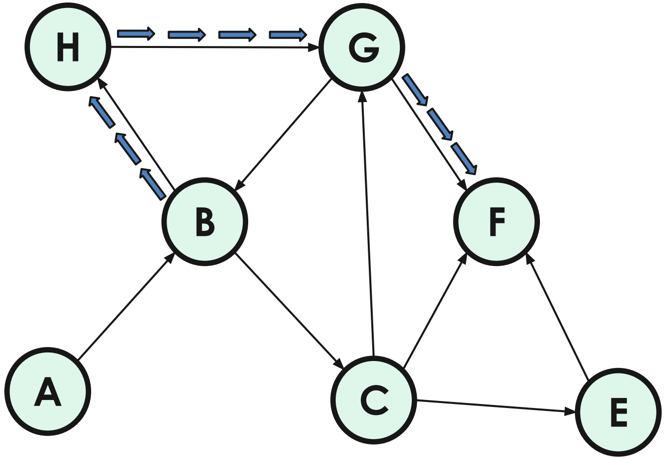
RDF2Vec

generates (random) **walks** on the knowledge graph data to be used as input for **word2vec** neural network

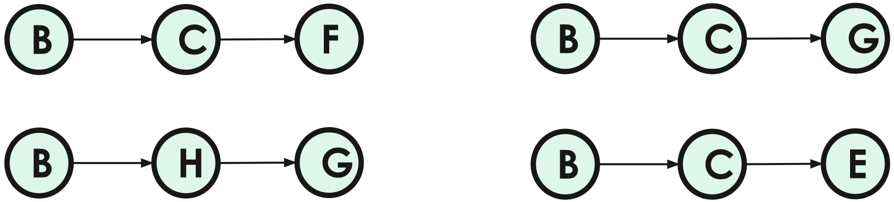


Rdf2vec: Rdf graph embeddings for data mining. Ristoski, Petar, and Heiko Paulheim. *International Semantic Web Conference*. Springer, Cham, 2016.

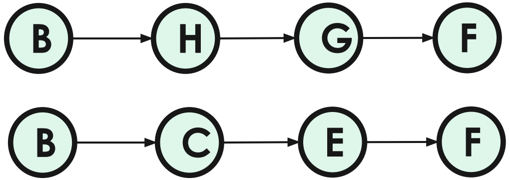
Random walks on graphs



Random walks of **length $\mathbf{k} = 2$** starting from **node $\mathbf{B}$**

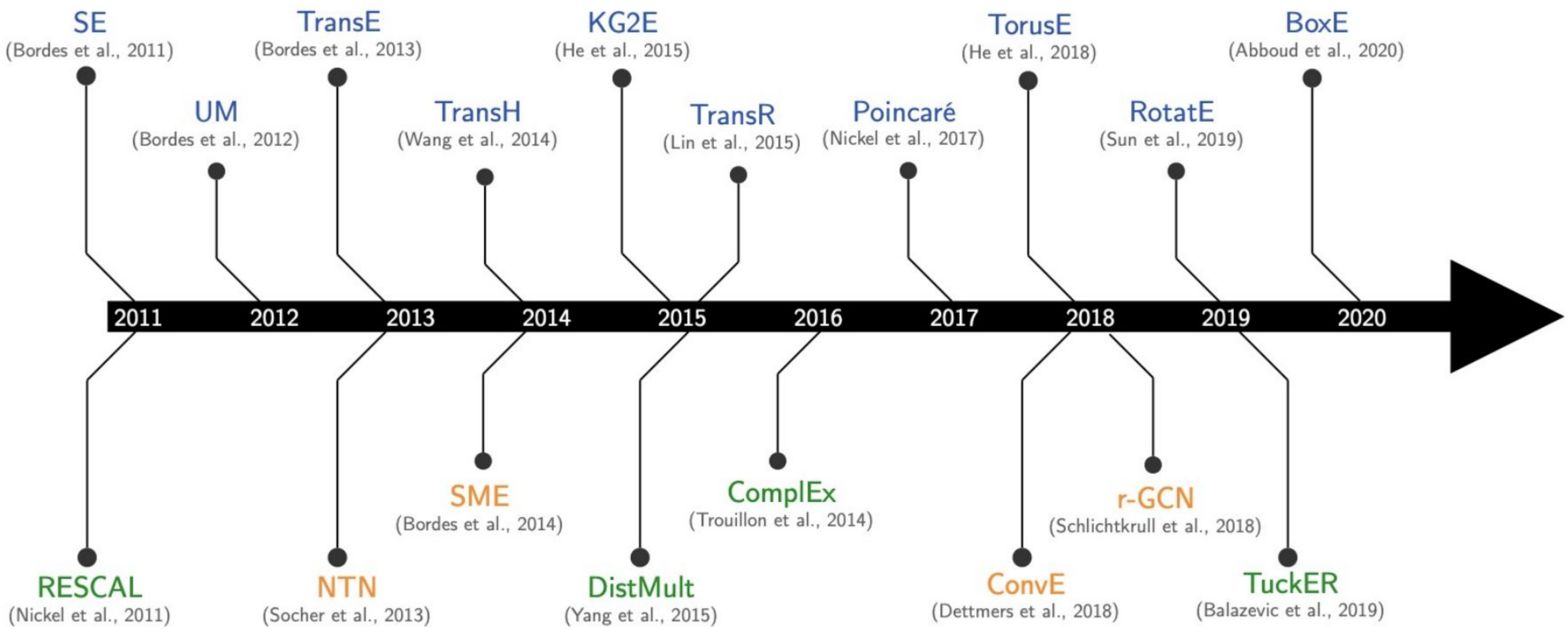


Random walks of **length $\mathbf{k} = 3$** starting from **node $\mathbf{B}$**



How many random walks and **of what length** need to be started from each node to cover the graph?

Many KG Embedding Models



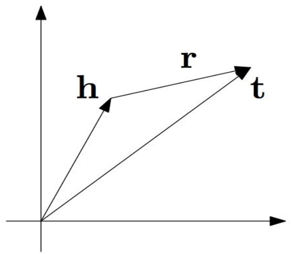
TransE

If a fact ***(h,r,t)*** holds, then ***h +r*** should be close to ***t***

Otherwise ***h+r*** should be distant to ***t***

Scoring function is L1 or L2 norm (distance measure)

$= \| e_h + r_p - e_t \|$



Pros:

- Simple
- Computationally efficient

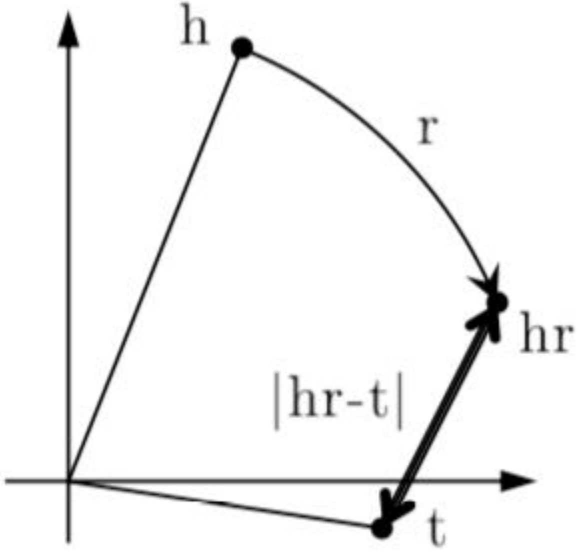
Cons:

- Cannot capture some properties (transitivity, anti-symmetry)
- Cannot handle 1-N, M-N relations

RotatE

relations modelled as rotations in complex space R: elementwise product between complex embeddings.

$$f_{RotatE} = -||\mathbf{e}_s \circ \mathbf{r}_p - \mathbf{e}_o||_n$$



Capturing KG properties

- **Symmetry**
  - A relation RR on a set A is **symmetric** if, for all $x,y \in A$ whenever xRy, then yRx. In other words, if one element is related to another, the reverse is also true.
  - <Alice marriedTo Bob> . <Bob marriedTo Alice>
- **Asymmetry**
  - A relation R on a set A is **asymmetric** if, for all $x,y \in A$, whenever xRy then it is not the case that yRx. This means that if one element is related to another, the reverse relation cannot exist between them.
  - <Alice childOf Jack>
- **Inversion**
  - The **inverse** of a relation R on a set A, denoted as R^{-1}, is defined such that for all $x,y \in A$, $xR^{-1}y$ if and only if yRx. Essentially, the inverse of a relation reverses the direction of that relation.
  - <Alice childOf Jack>
  - <Jack fatherOf Alice>
- **Composition**
  - The **composition** of two relations R and S on a set A, denoted as $R \circ S$, is defined such that for all $x,z \in A$, $x(R \circ S)z$ if and only if there exists a $y \in A$ such that xRy and ySz. This describes a situation where the existence of intermediary elements allows for a compound relationship to be established between two elements.
  - <Alice childOf Jack> + <Jack siblingOf Mary> => <Alice nieceOf Mary>

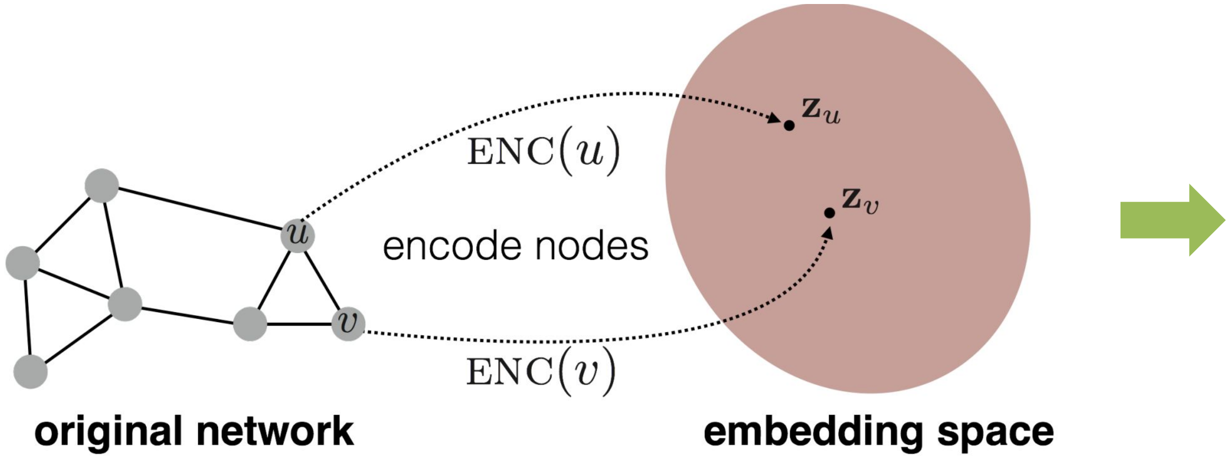
Knowledge Graph Embedding

Intuition: Predicate embedding captures the relation (+/x) between head and tail embeddings

Model	Score Function	Symmetry	Antisymmetry	Inversion	Composition
SE	$-\lVert \boldsymbol{W}_{r,1}\mathbf{h} - \boldsymbol{W}_{r,2}\mathbf{t} \rVert$	✗	✗	✗	✗
TransE	$-\lVert \mathbf{h} + \mathbf{r} - \mathbf{t} \rVert$	✗	✓	✓	✓
TransX	$-\lVert g_{r,1}(\mathbf{h}) + \mathbf{r} - g_{r,2}(\mathbf{t}) \rVert$	✓	✓	✗	✗
DistMult	$\langle \mathbf{h}, \mathbf{r}, \mathbf{t} \rangle$	✓	✗	✗	✗
ComplEx	$\mathrm{Re}(\langle \mathbf{h}, \mathbf{r}, \overline{\mathbf{t}} \rangle)$	✓	✓	✓	✗
RotatE	$-\lVert \mathbf{h} \circ \mathbf{r} - \mathbf{t} \rVert$	✓	✓	✓	✓

Graph Neural Networks

Goal: encode nodes so that similarity in embedding space (i.e. dot product) approximates similarly in original graph



- Node classification
- Link prediction
- Graph classification
- Anomalous node detection
- Clustering
- etc…

Learning Node Embeddings

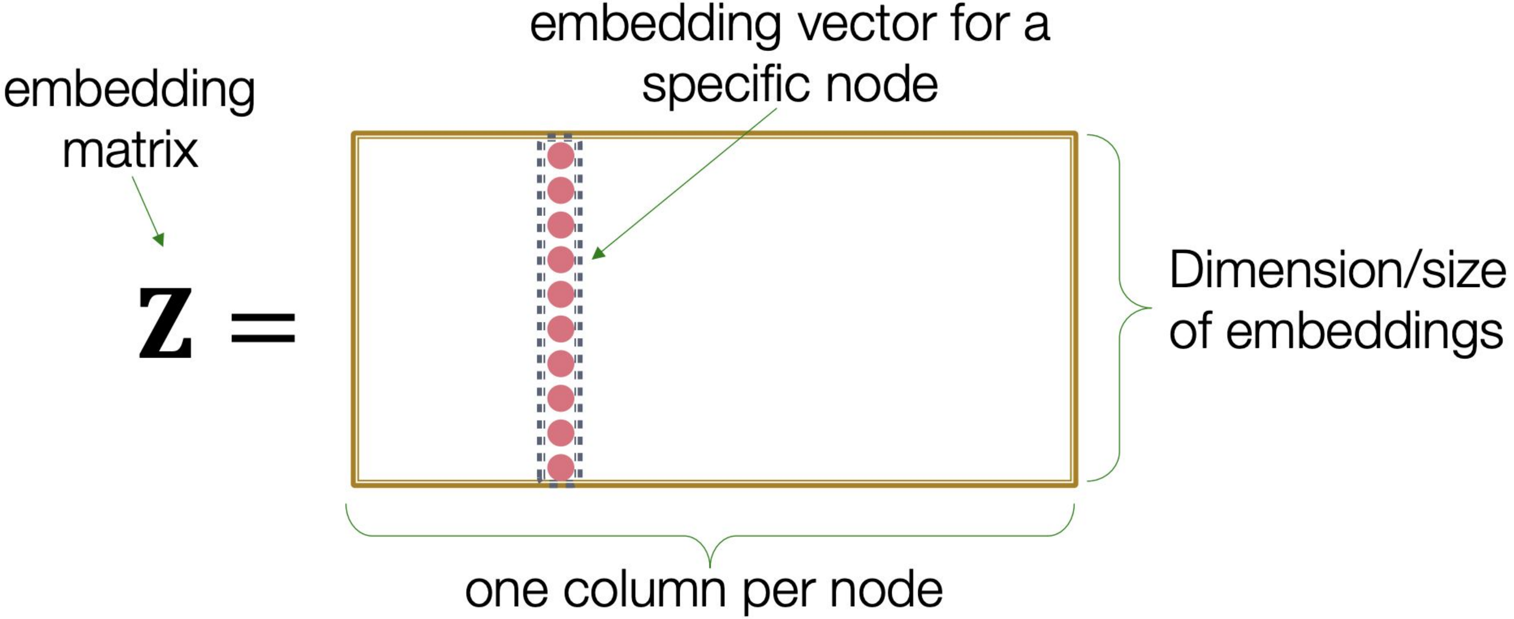
- **Encoder** maps from nodes to embeddings
- Define a node **similarity function** (i.e., a measure of similarity in the original network)
- **Decoder** maps from embeddings to the similarity score
- Optimize the parameters of the encoder so that:

$$\text{similarity}(u,v) \cong \underbrace{\text{ENC}(u)}_{\text{Embedding } z_u} \;.\; \underbrace{\text{ENC}(v)}_{\text{Embedding } z_v}$$

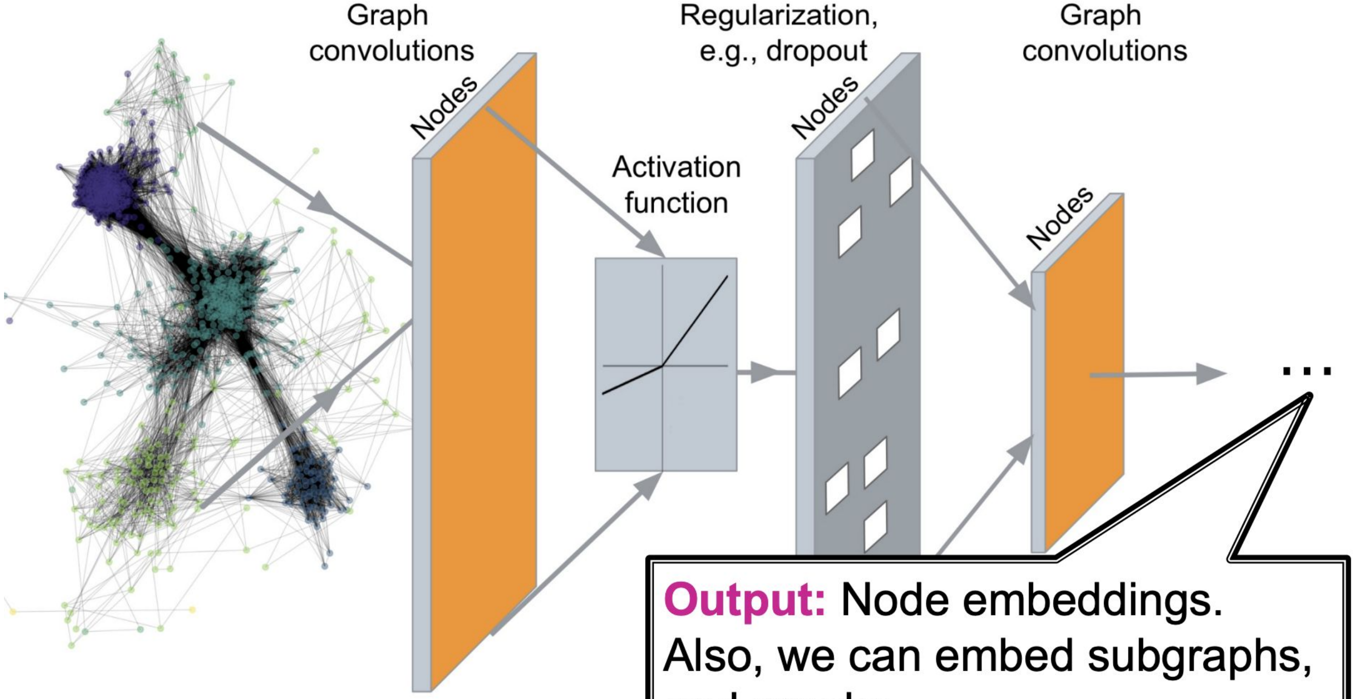
dot product operator

"Shallow" Encoding

Simplest encoding approach: encoder is just an embedding-lookup



Deep Graph Encoders

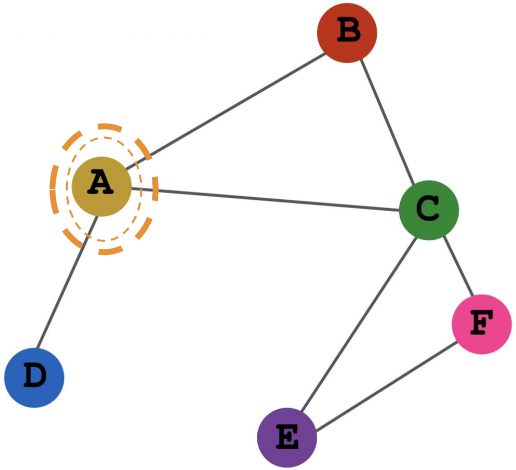


Output: Node embeddings. Also, we can embed subgraphs, and graphs

Node Prediction

Directly train the model for node prediction

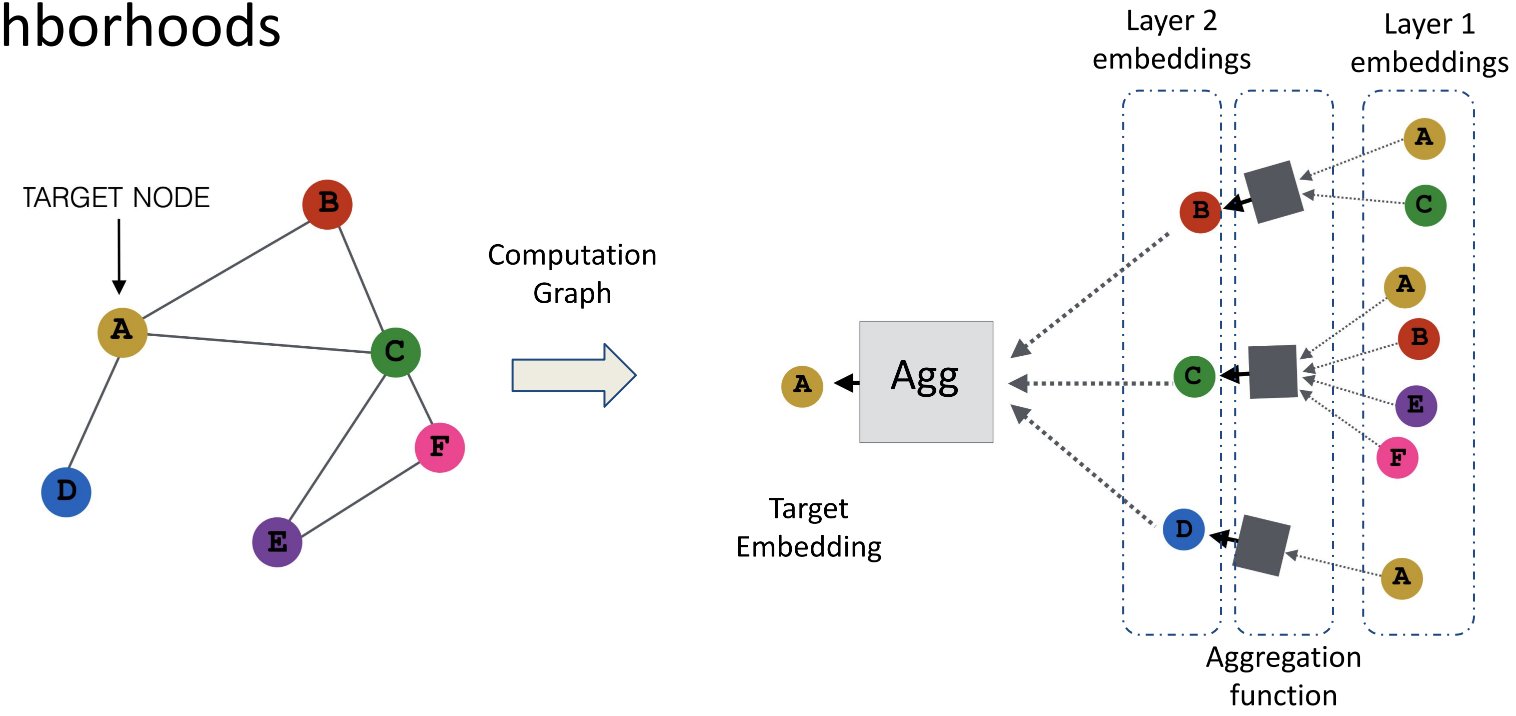
Drug type?



- Node’s neighborhood defines a computation graph
- Learn how to propagate information across the graph to compute node features
- Nodes aggregate information from their neighbors using neural networks

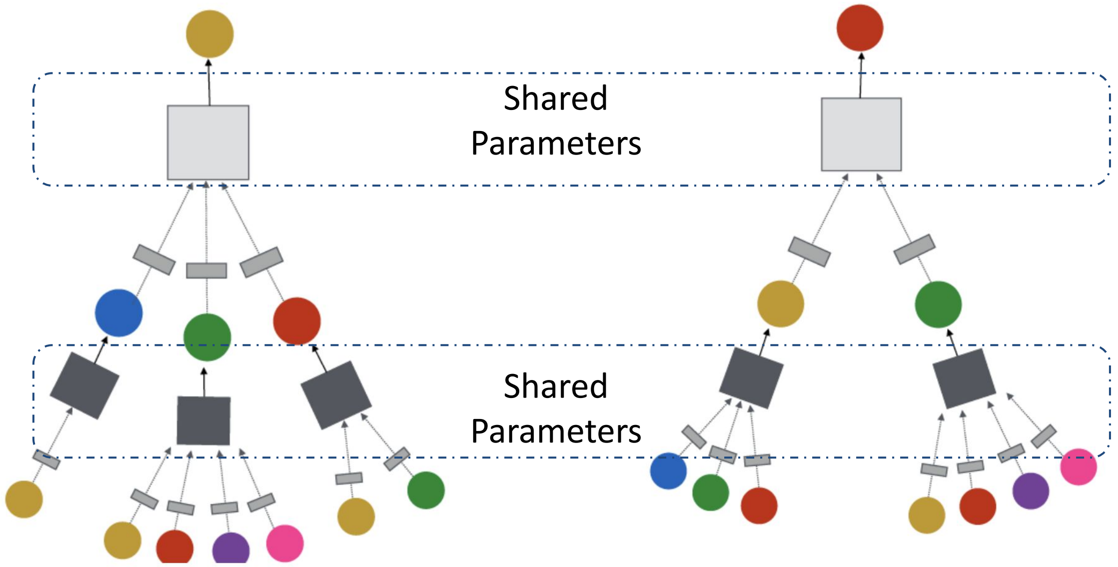
Aggregate Neighbour Embeddings

Key idea: Generate node embeddings based on local network neighborhoods



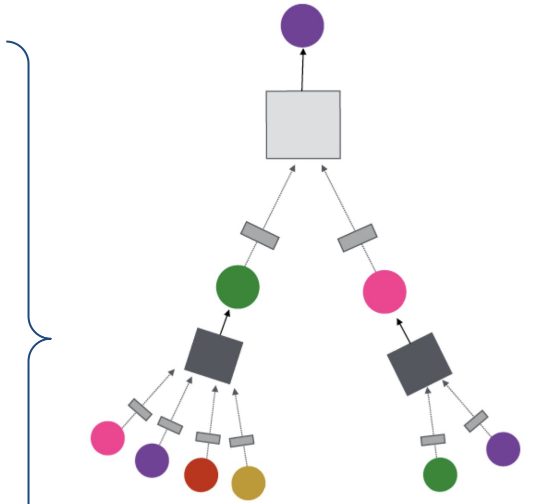
Inductive Capacity

The same aggregation parameters learned using neural networks are shared



Computation graph of
Node A

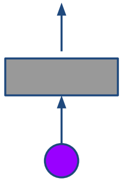
Computation graph of
Node B



Apply to new nodes, e.g., train on protein interaction graph from model organism A and generate embeddings on newly collected data about organism B

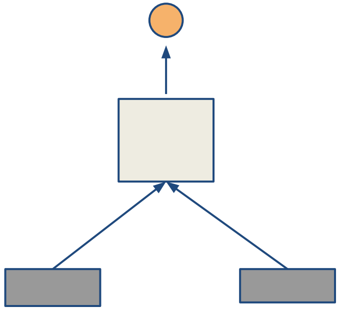
A Single GNN Layer

There are two types of parameters in the model:



Message computation: each node u creates an information message $\mathbf{m}_u$, represented by its embedding vector $\mathbf{h}_u$, parameterized with neural network to send to other nodes

Example: a linear layer $\mathbf{m}_u = \mathbf{W}\,\mathbf{h}_u$

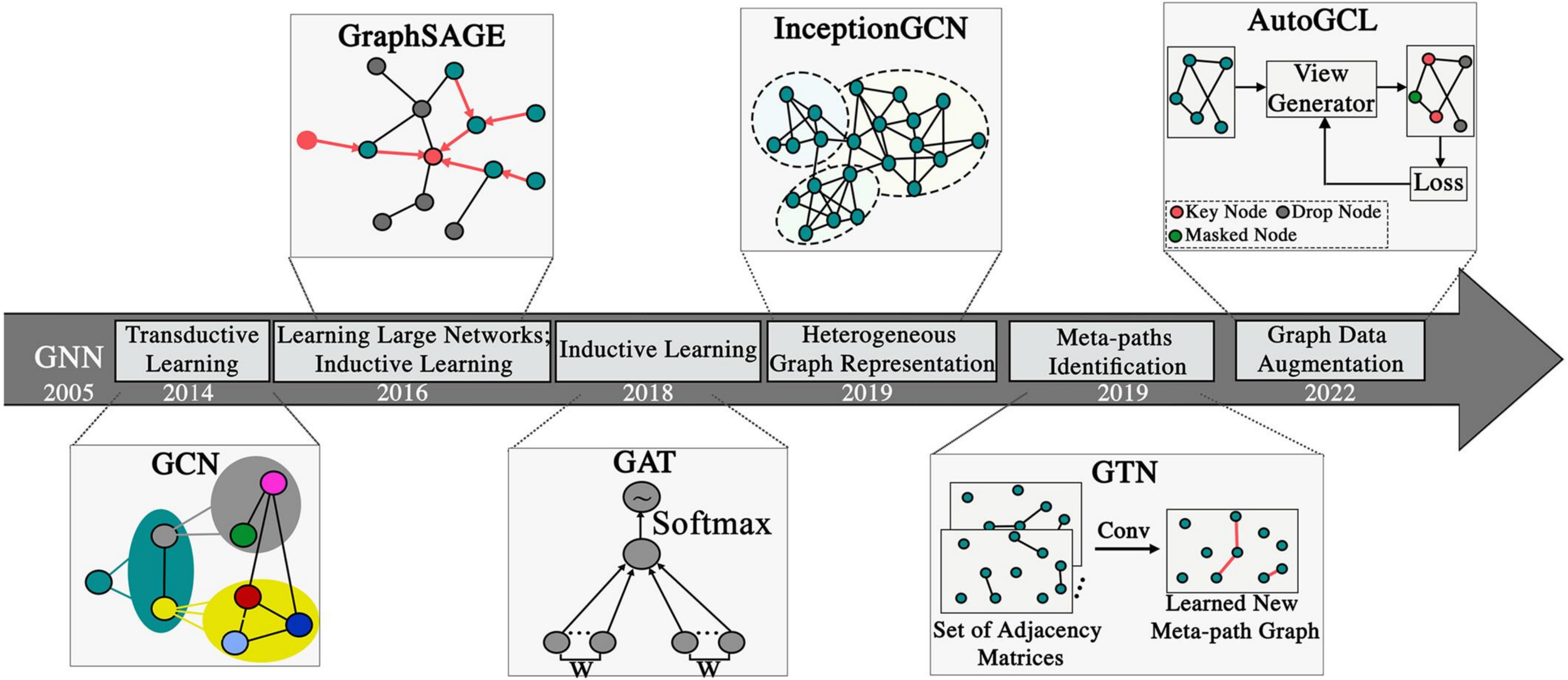


Message Aggregation: node v will aggregate messages from its neighbors u

Example: sum(), mean(), or max() aggregator

$\mathbf{h}_v = \mathrm{Mean}(\{\mathbf{m}_u \mid u \text{ a neighbour of } v\})$

Other Variations

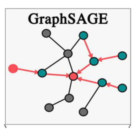


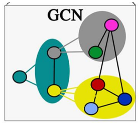
iRADIOLOGY, Volume: 1, Issue: 2, Pages: 151-166, First published: 27 June 2023, DOI: (10.1002/ird3.20)

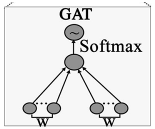
In Practice

To make GNN work really well, need also the following techniques:

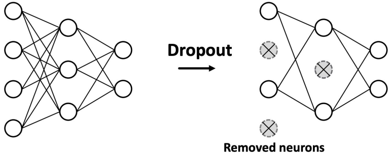
- A GNN module





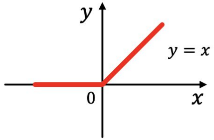


- Batch Normalization, stabilizes training by re-centering and re-scaling parameters
- Dropout, prevents overfitting by dropping random links

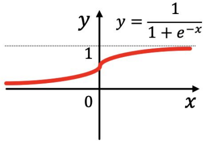


- Activation, improves expressivity

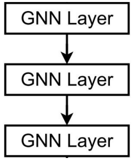
ReLu



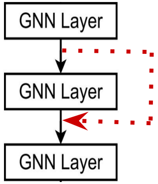
Sigmoid



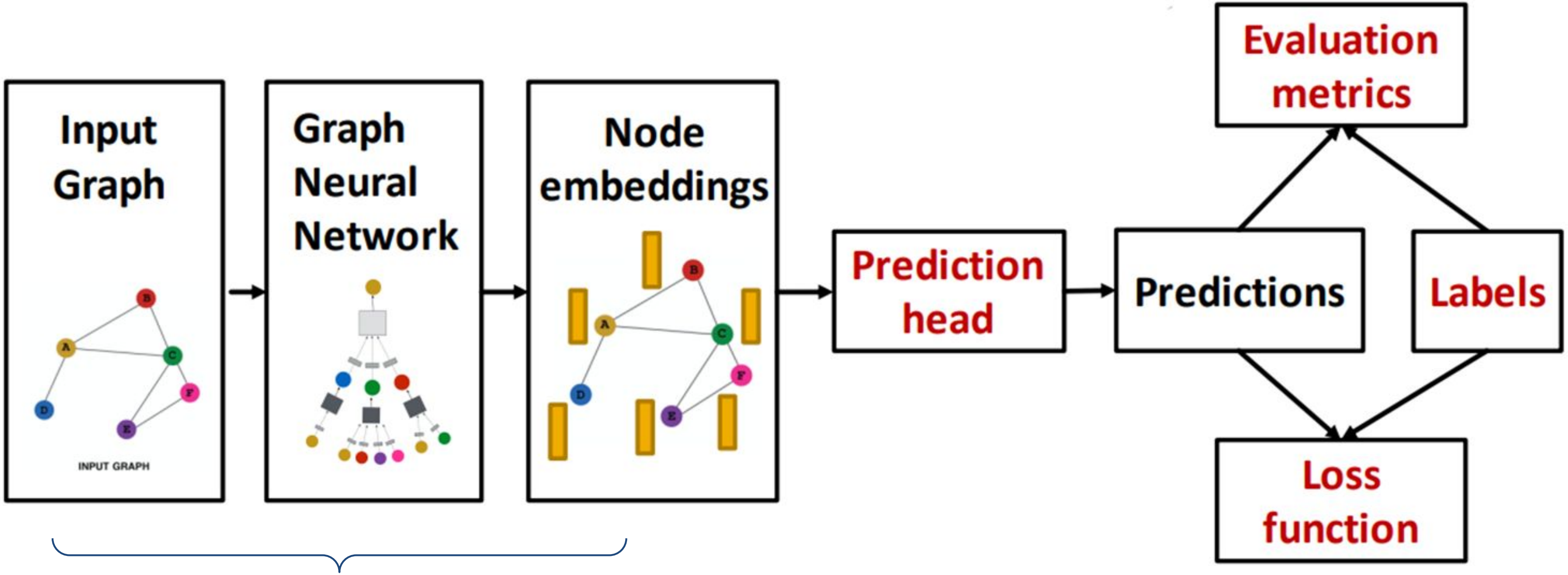
- Stack GNNs, increases expressivity and works great in practice



- Skip connections, to prevent over-smoothing due to larger receptive fields by adding back original input

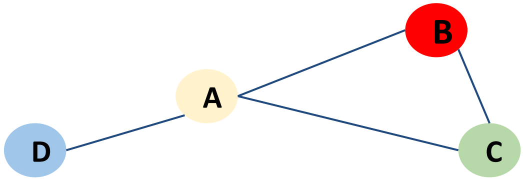


Training Framework

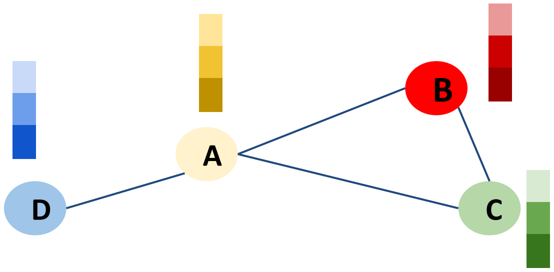


Forward Pass

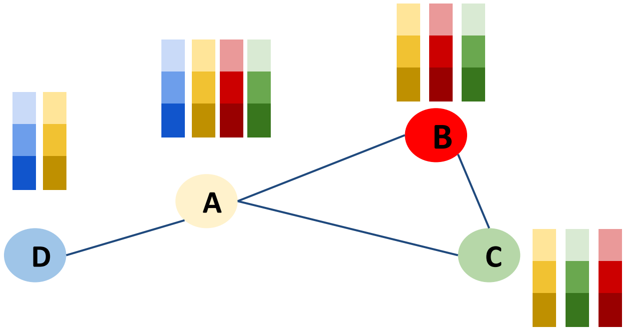
Forward Pass



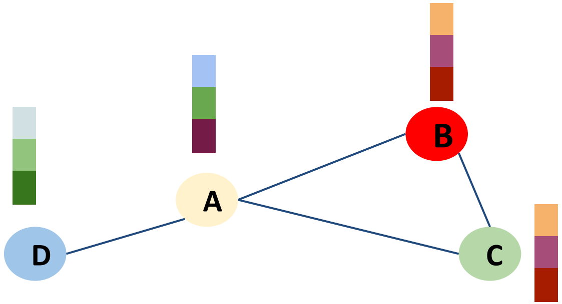
Compute messages



Propagate messages

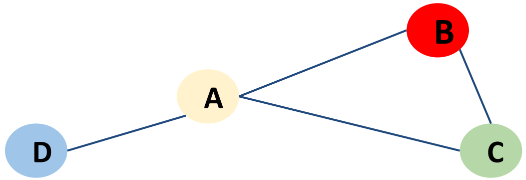


Aggregate

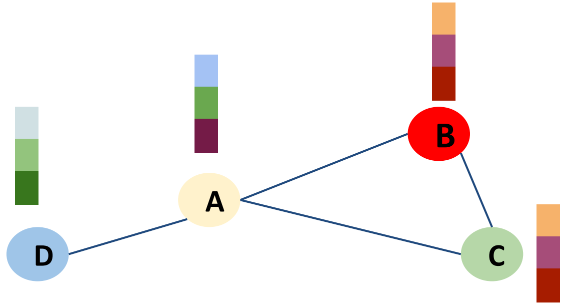


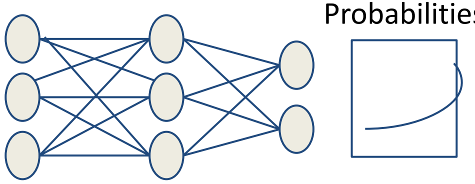
Prediction & Learning

The goal of learning is to minimize the loss between predictions and labels:



Forward Pass





Output Probabilities

Prediction Head is typically a fully connected layer (of size embedding dimension x number of classes), followed by prediction function

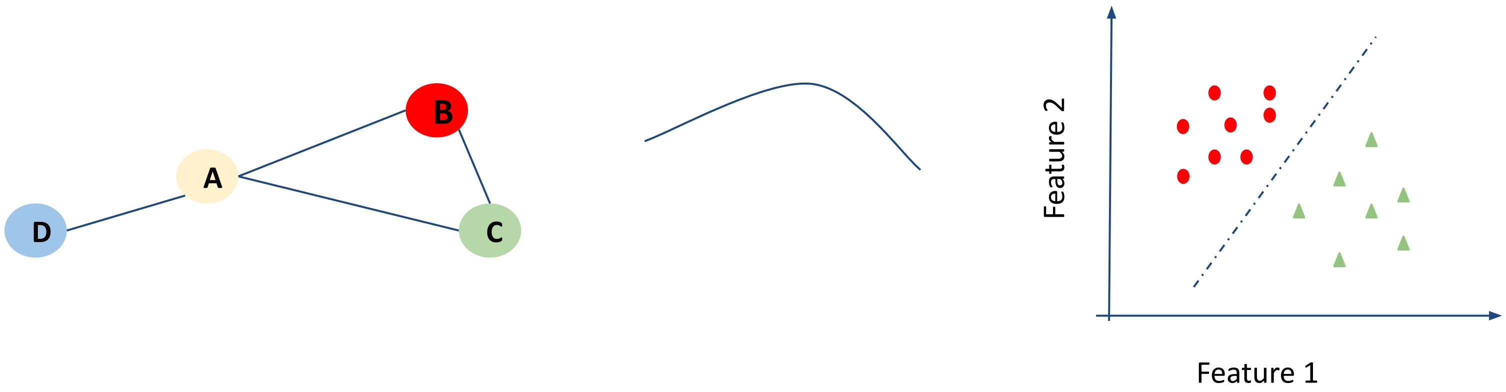
Prediction Head & Loss Computation

Node	Embeddings	Output	Labels
A		0.4	1
B		0.7	1
C		0.91	1
D		0.1	0

Backpropogate these errors to update the parameters

2D Interpretation

With linear layers, this process is the same as learning embeddings such that node representations are linearly separable by a hyper-plane, i.e. **binary/multi-label classification with graph input**



Evaluation Metrics

For Binary Node Classification, the following metrics can be used:

Accuracy: $$\frac{TP + TN}{|\text{dataset}|}$$

Precision (P): $$\frac{TP}{TP + FP}$$

Recall (R): $$\frac{TP}{TP + FN}$$

F1 - score: $$\frac{2\,P \times R}{P + R}$$