ITCS 6156/8156 Fall 2024
Machine Learning

Graph Neural Network

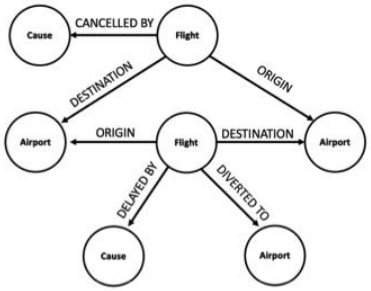
Instructor: Hongfei Xue
Email: hongfei.xue@charlotte.edu
Class Meeting: Tue & Thu, 4:00 PM – 5:15 PM, WWH 130

UNIVERSITY OF NORTH CAROLINA CHARLOTTE

Some content in the slides is based on https://web.stanford.edu/class/cs224w/

Why Graphs?

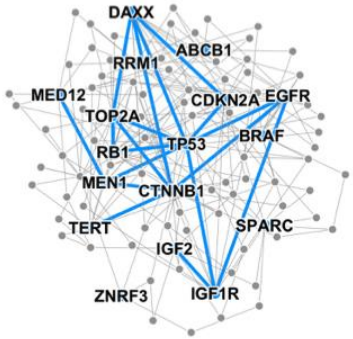
- Graphs are a general language for describing and analyzing entities with relations/interactions.



Event Graphs

Image credit: SalientNetworks

Computer Networks



Disease Pathways

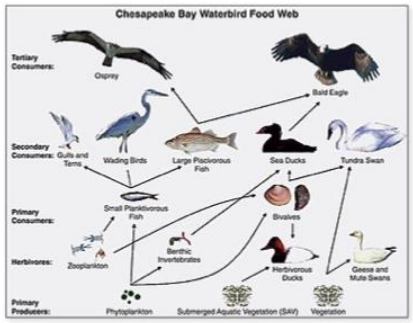


Image credit: Wikipedia

Food Webs

Image credit: Pinterest

Particle Networks

Image credit: visitlondon.com

Underground Networks

Why Graphs?

- Graphs are a general language for describing and analyzing entities with relations/interactions.

Image credit: Medium

Social Networks

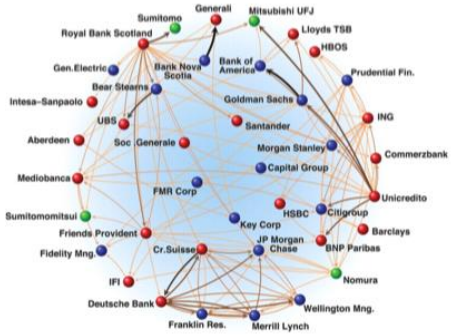


Image credit: Science

Economic Networks

Image credit: Lumen Learning

Communication Networks

Citation Networks

Image credit: Missoula Current News

Internet

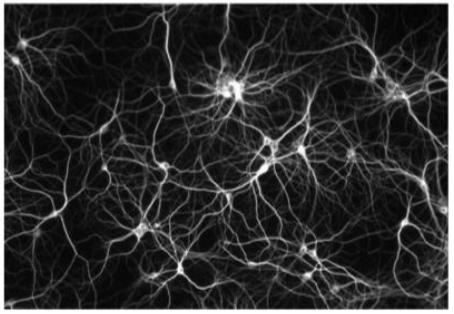


Image credit: The Conversation

Networks of Neurons

Why Graphs?

- Graphs are a general language for describing and analyzing entities with relations/interactions.

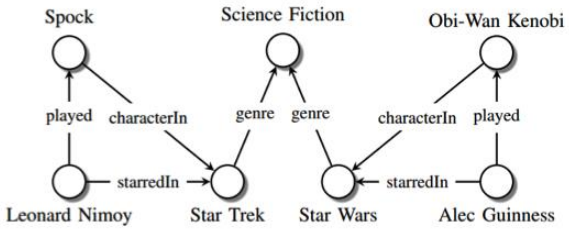


Image credit: Maximilian Nickel et al

Knowledge Graphs

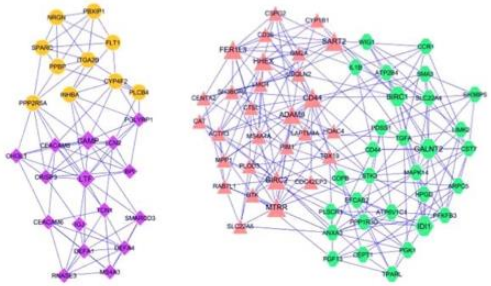


Image credit: ese.wustl.edu

Regulatory Networks

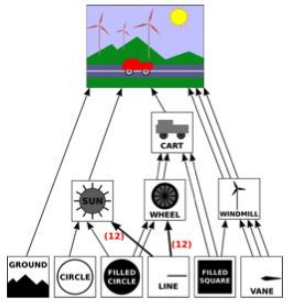


Image credit: math.hws.edu

Scene Graphs

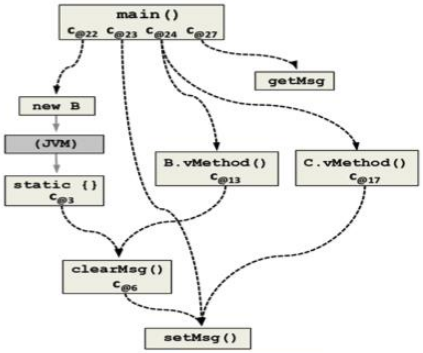


Image credit: ResearchGate

Code Graphs

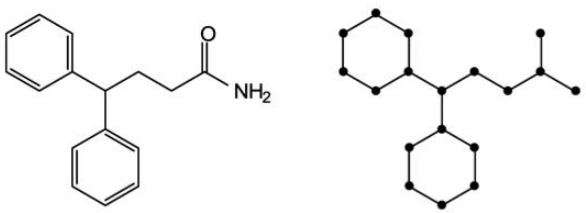


Image credit: MDPI

Molecules

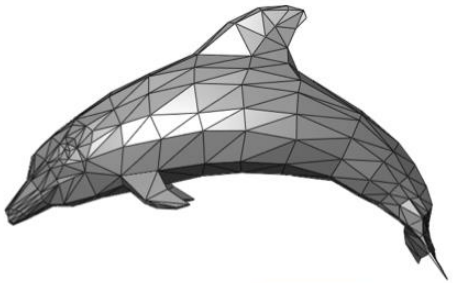


Image credit: Wikipedia

3D Shapes

Graphs: Machine Learning

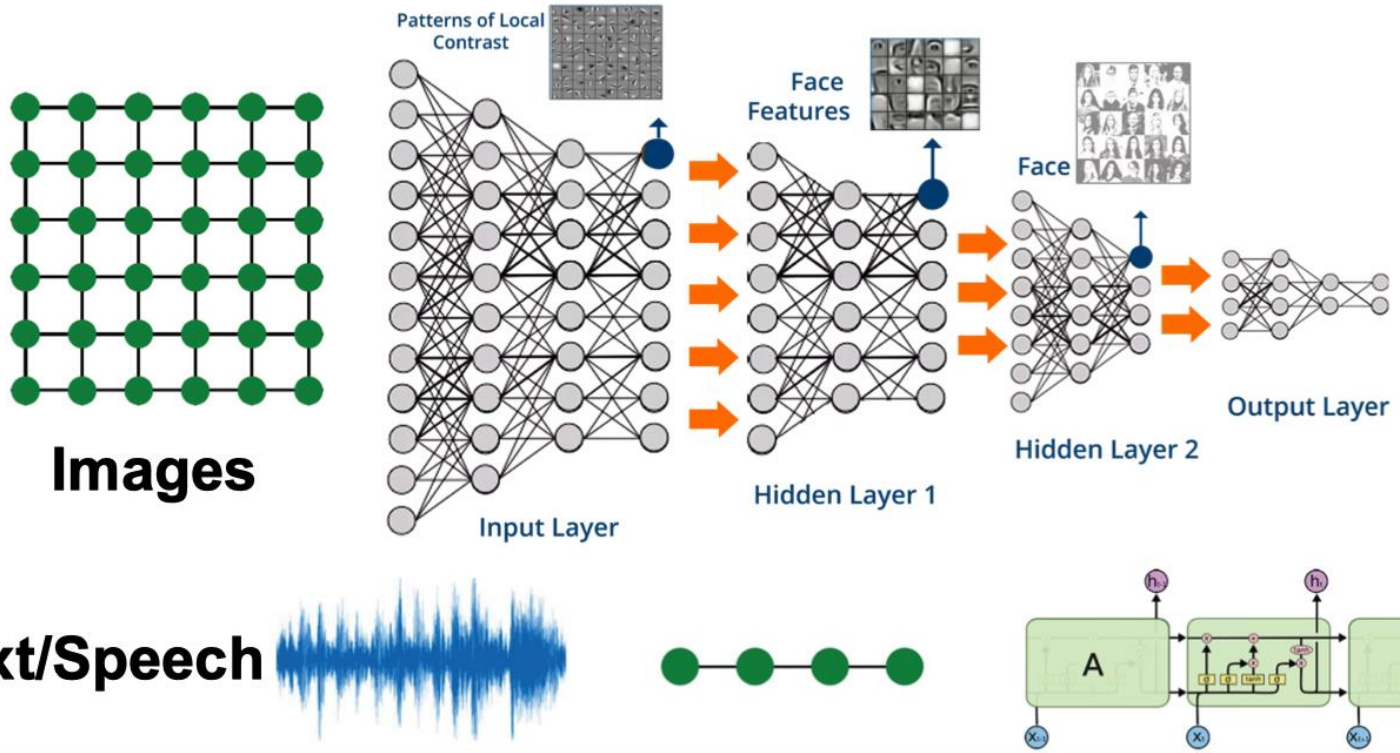
Complex domains have a rich relational structure, which can be represented as a **relational graph**

By explicitly modeling relationships we achieve better performance!

Main question:

How do we take advantage of relational structure for better prediction?

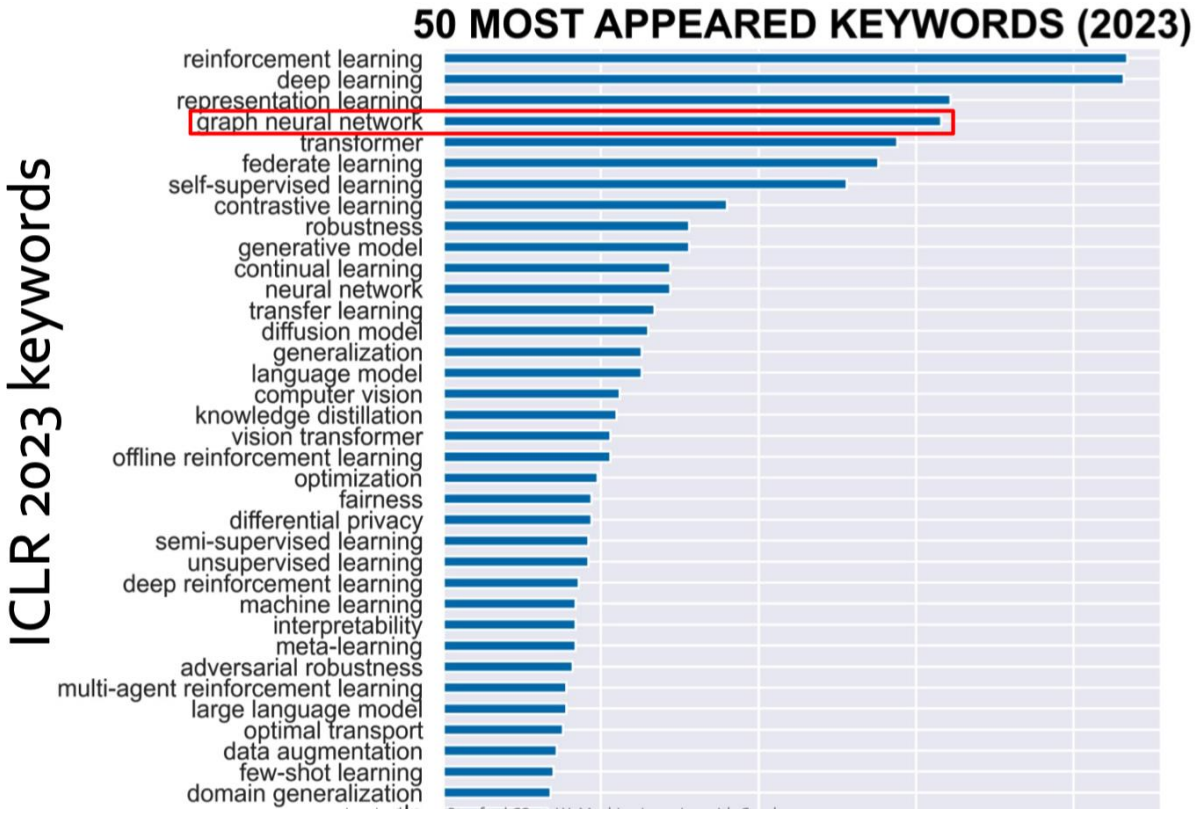
Deep Models We Learned



Modern deep learning toolbox is designed for simple sequences & grids

Graph Neural Network

- Graphs are the new frontier of deep learning



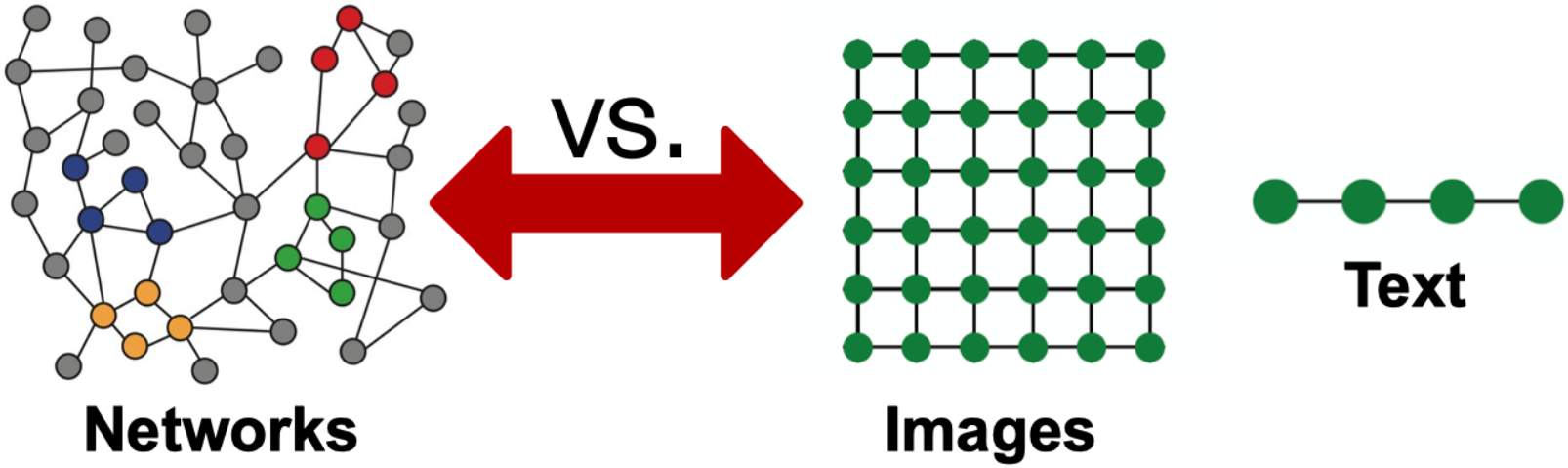
ICLR 2024 Top 50 Keywords

Keyword	Count
Large Language Models	318
Reinforcement Learning	201
Graph Neural Networks	123
Diffusion Models	112
Deep Learning	110
Representation Learning	107
Generative Models	86
Federated Learning	75
Language Models	74
Interpretability	66
Generalization	61
Transformers	56
Robustness	51
Neural Networks	47
Self-supervised Learning	45
Contrastive Learning	44
Transformer	44

Graph Deep Learning is Hard

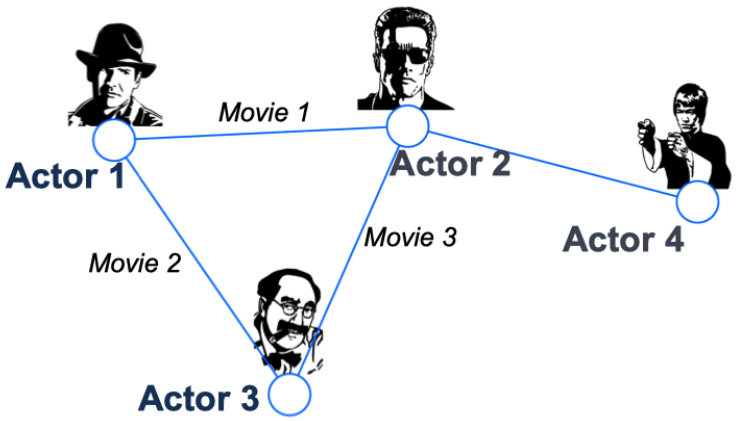
Networks are complex.

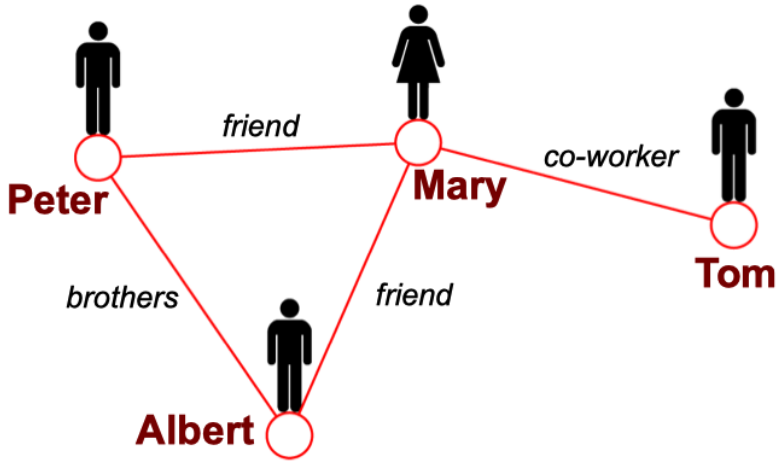
- Arbitrary size and complex topological structure (*i.e.*, no spatial locality like grids)

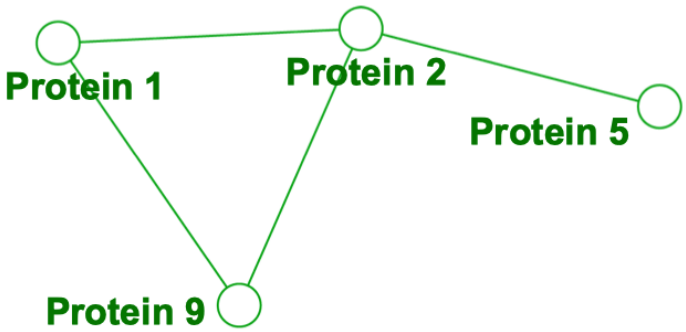


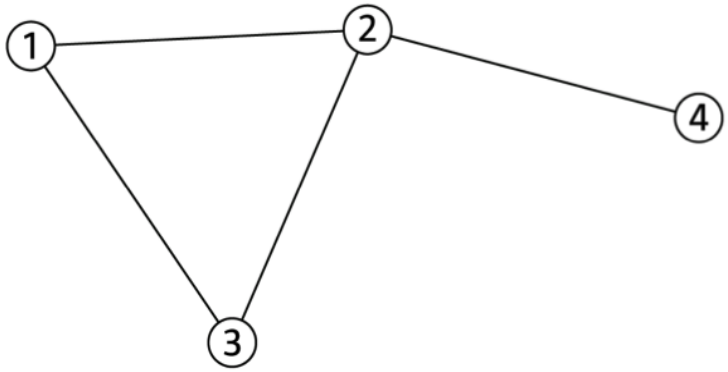
- No fixed node ordering or reference point
- Often dynamic and have multimodal features

Graph Representation





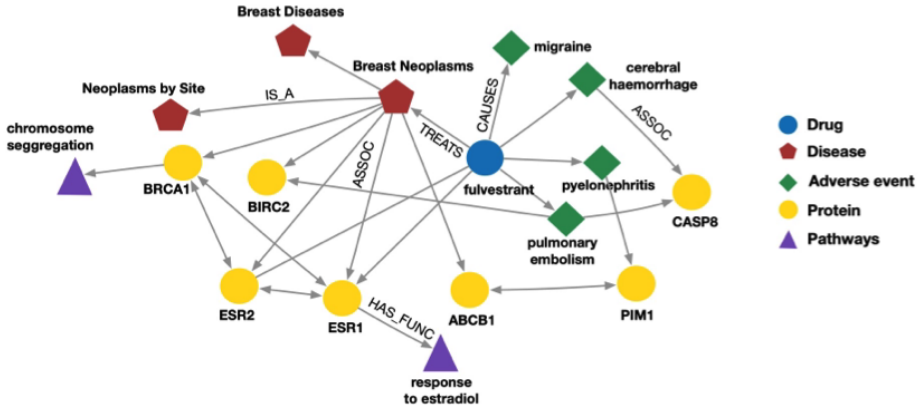


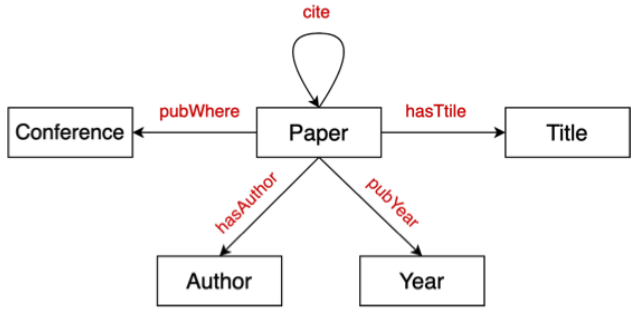


Graphs

- **A heterogeneous graph is defined as**
  $$\boldsymbol{G = (V, E, R, T)}$$
  - Nodes with node types $v_i \in V$
  - Edges with relation types $(v_i, r, v_j) \in E$
  - Node type $T(v_i)$
  - Relation type $r \in R$
  - Nodes and edges have attributes/features

Graph Examples





Biomedical Knowledge Graphs

Example node: Migraine
Example edge: (fulvestrant, Treats, Breast Neoplasms)
Example node type: Protein
Example edge type (relation): Causes

Academic Graphs

Example node: ICML
Example edge: (GraphSAGE, NeurIPS)
Example node type: Author
Example edge type (relation): pubYear

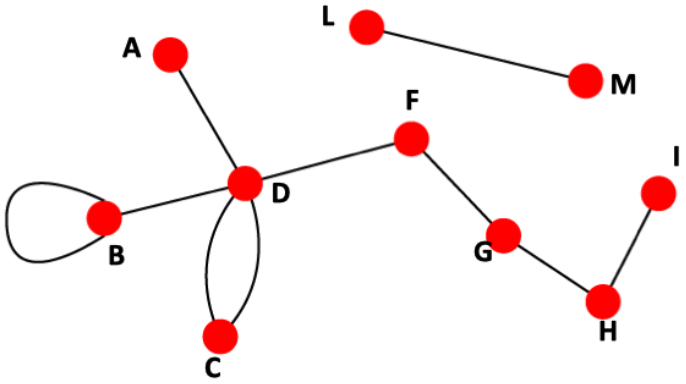
A Proper Representation

- **How to build a graph:**
  - **What are nodes?**
  - **What are edges?**
- **Choice of the proper network representation of a given domain/problem determines our ability to use networks successfully:**
  - In some cases, there is a unique, unambiguous representation
  - In other cases, the representation is by no means unique
  - The way you assign links will determine the nature of the question you can study

Directed vs. Undirected Graphs

Undirected

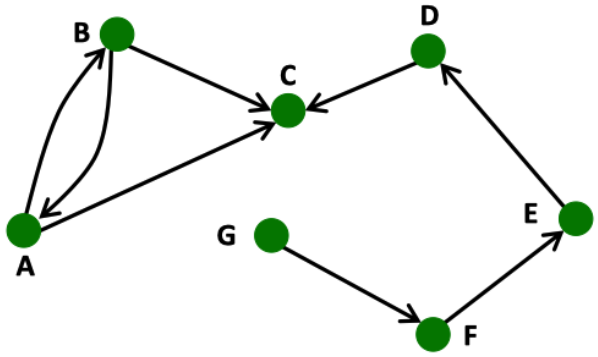
- **Links:** undirected (symmetrical, reciprocal)



- **Other considerations:**
  - Weights
  - Properties

Directed

- **Links:** directed

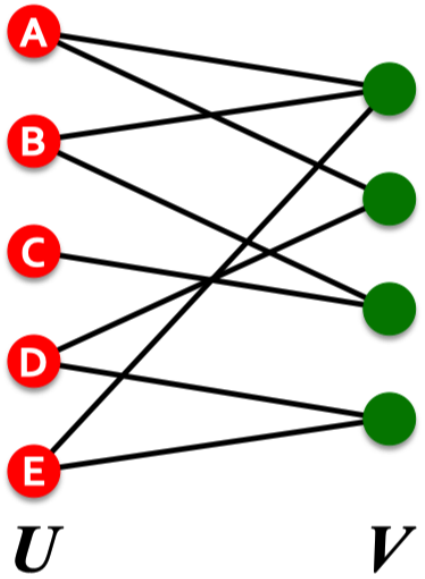


  - Types
  - Attributes

Bipartite Graph

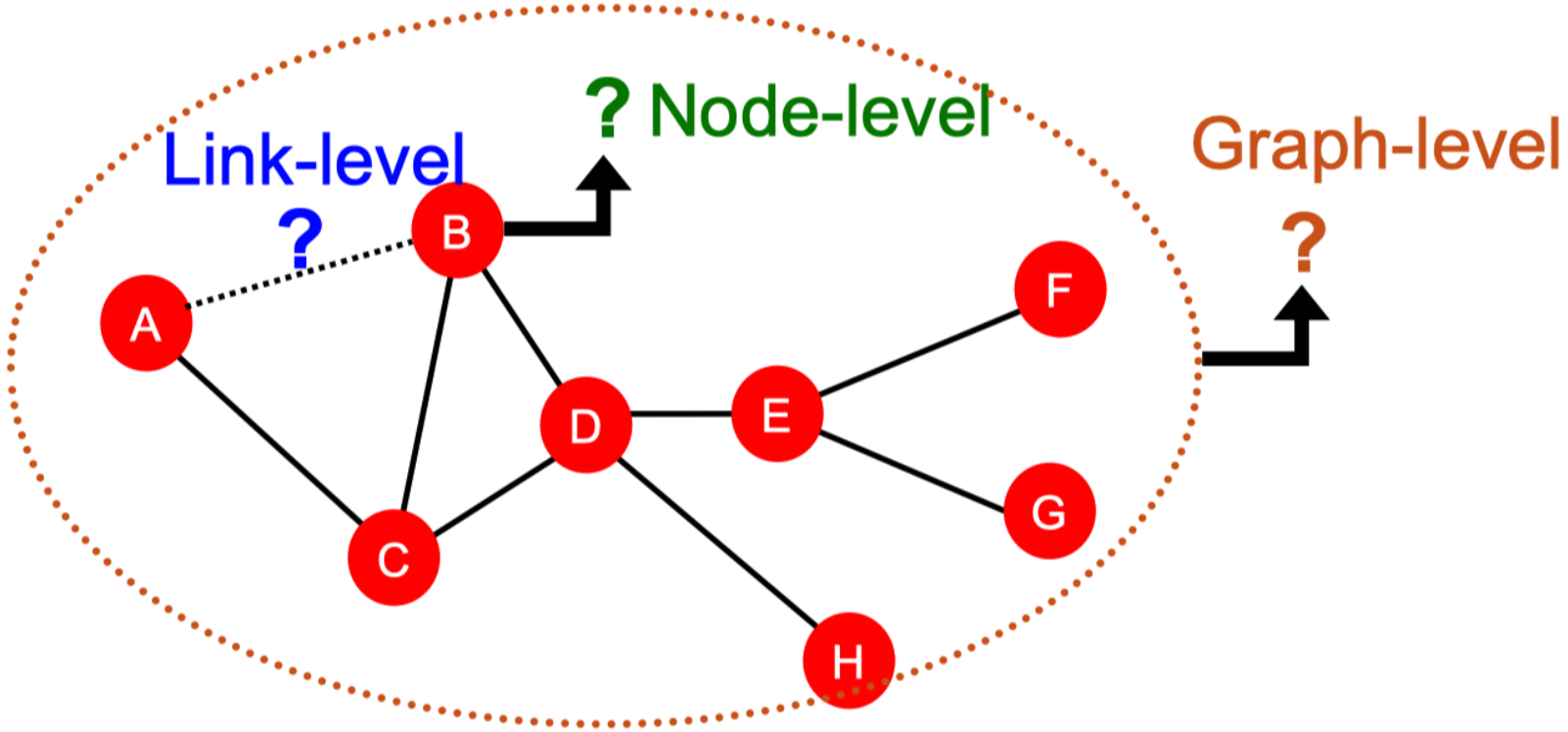
- **Bipartite graph** is a graph whose nodes can be divided into two disjoint sets ***U*** and ***V*** such that every link connects a node in ***U*** to one in ***V***; that is, ***U*** and ***V*** are **independent sets**

- **Examples:**
  - Authors-to-Papers (they authored)
  - Actors-to-Movies (they appeared in)
  - Users-to-Movies (they rated)
  - Recipes-to-Ingredients (they contain)

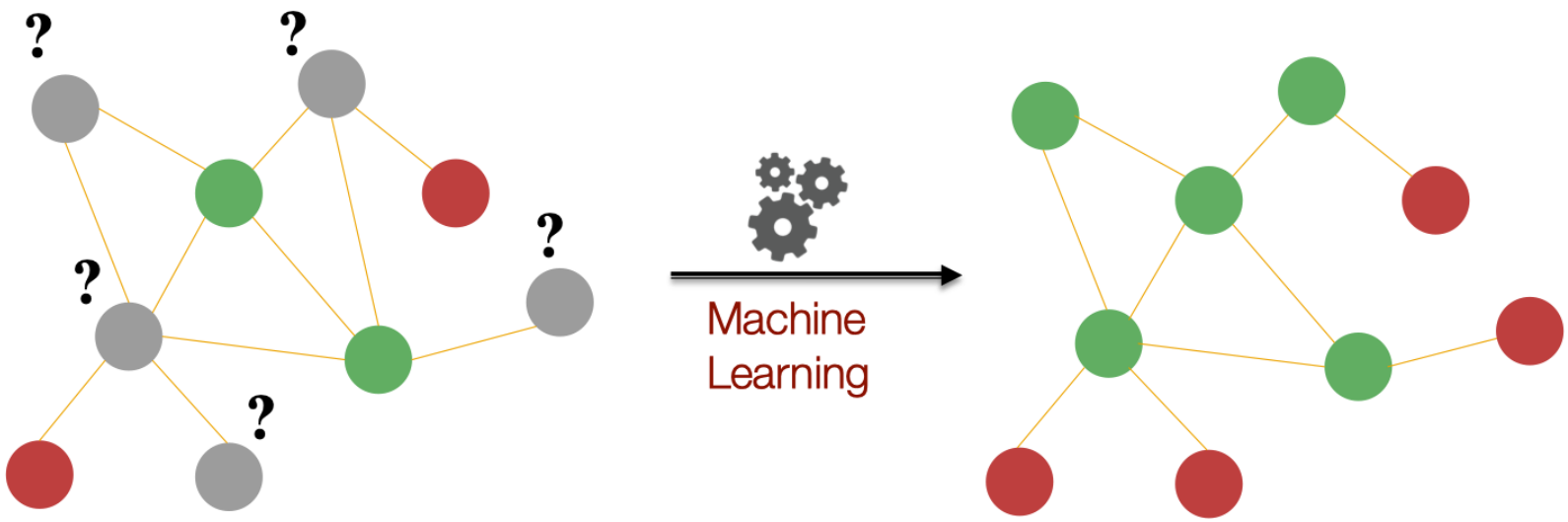


Different Types of Tasks

- Node-level prediction
- Link-level prediction
- Graph-level prediction



Node-level Tasks

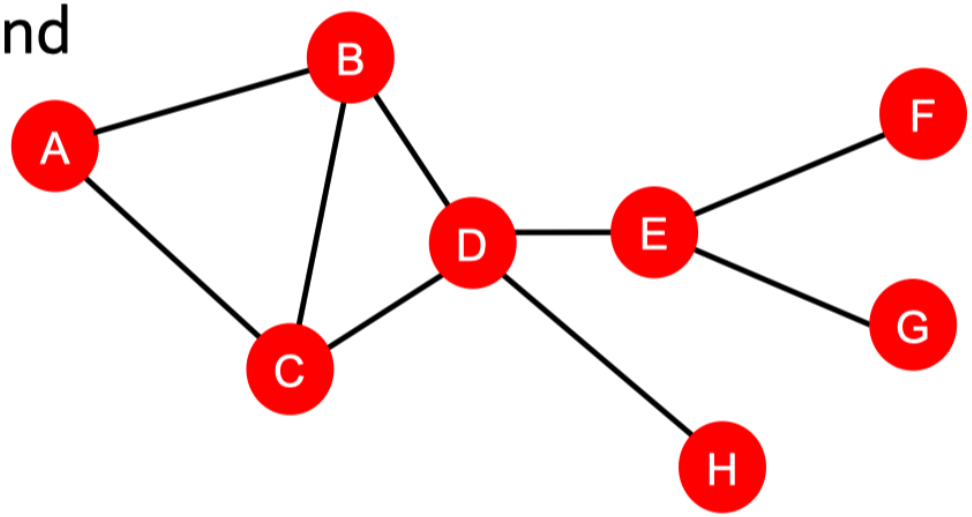


Node classification

Node-level Network Structure

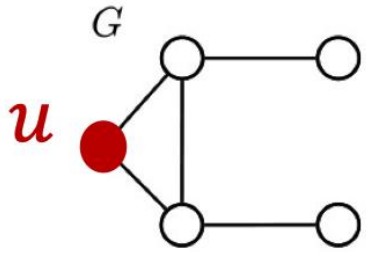
Goal: Characterize the structure and position of a node in the network:

- Node degree
- Node importance & position
  - E.g., Number of shortest paths passing through a node
  - E.g., Avg. shortest path length to other nodes
- Substructures around the node

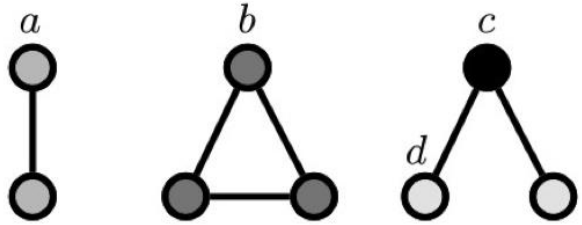


Node's Subgraphs: Graphlets

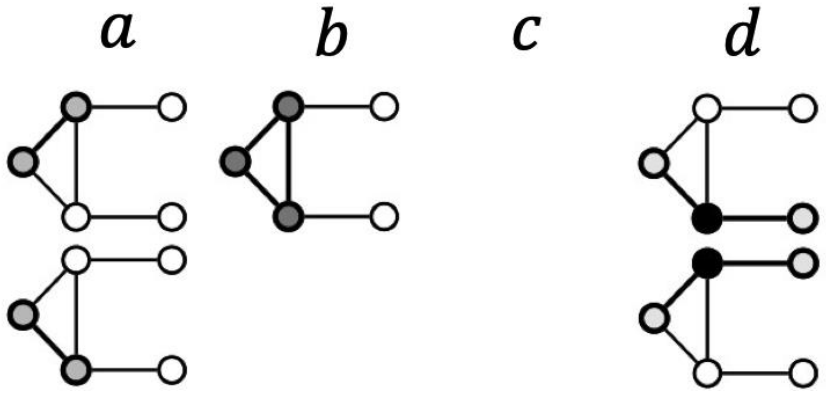
- **Graphlets:** A count vector of rooted subgraphs at a given node.
- **Example:**



All possible graphlets on up to 3 nodes



Graphlet instances of node u:



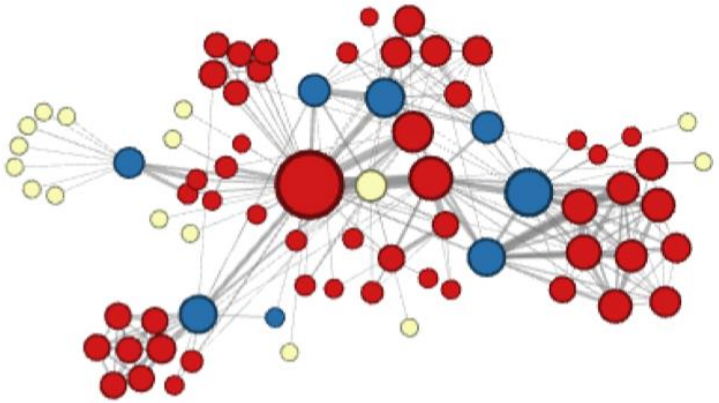
Graphlets of node u:

a, b, c, d

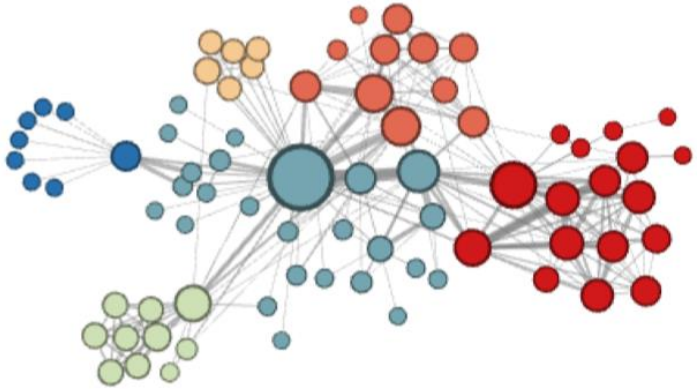
[2,1,0,2]

Node-level Tasks

Different ways to label nodes of the network:



Node features defined so far would allow to distinguish nodes in the above example

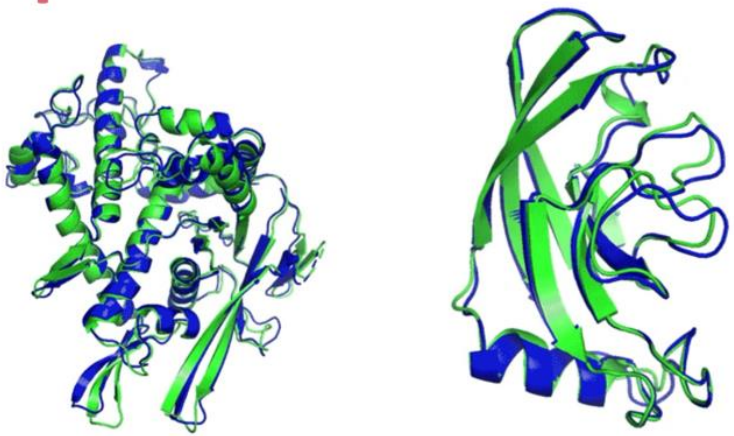


However, the features defines so far would not allow for distinguishing the above node labelling

Node-level Task Example: Protein Folding

Computationally predict a protein's 3D structure based solely on its amino acid sequence:

For each node predict its 3D coordinates



T1037 / 6vr4
90.7 GDT
(RNA polymerase domain)

T1049 / 6y4f
93.3 GDT
(adhesin tip)

Experimental result
Computational prediction

Image credit: DeepMind

Node-level Task Example: Protein Folding

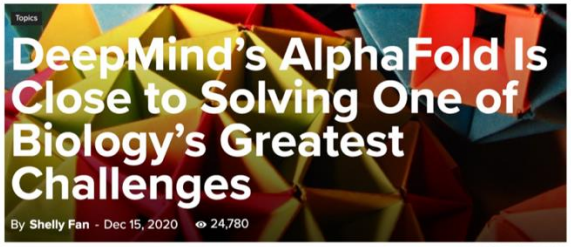


Image credit: SingularityHub

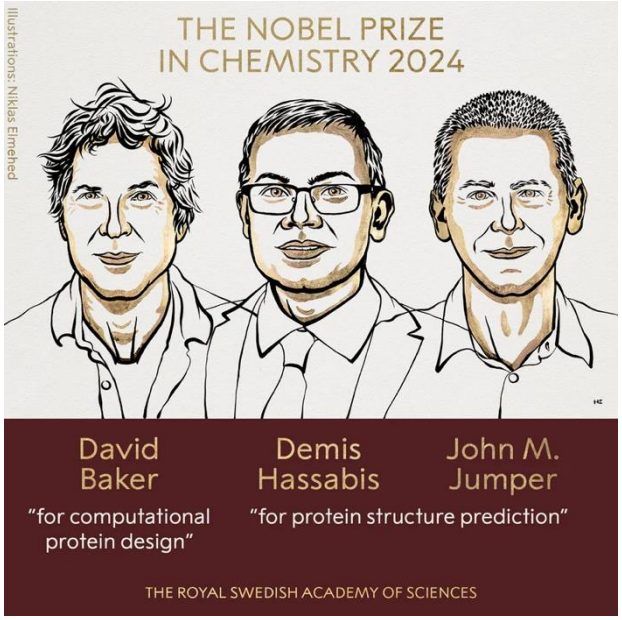
AlphaFold's AI could change the world of biological science as we know it

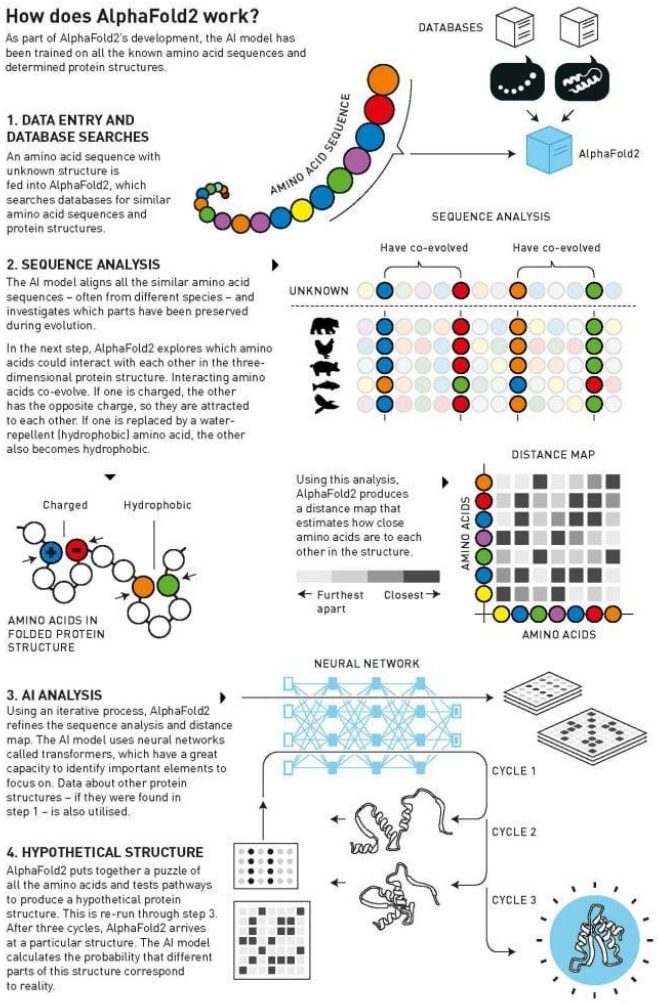
DeepMind's latest AI breakthrough can accurately predict the way proteins fold

Has Artificial Intelligence 'Solved' Biology's Protein-Folding Problem?

12-14-20

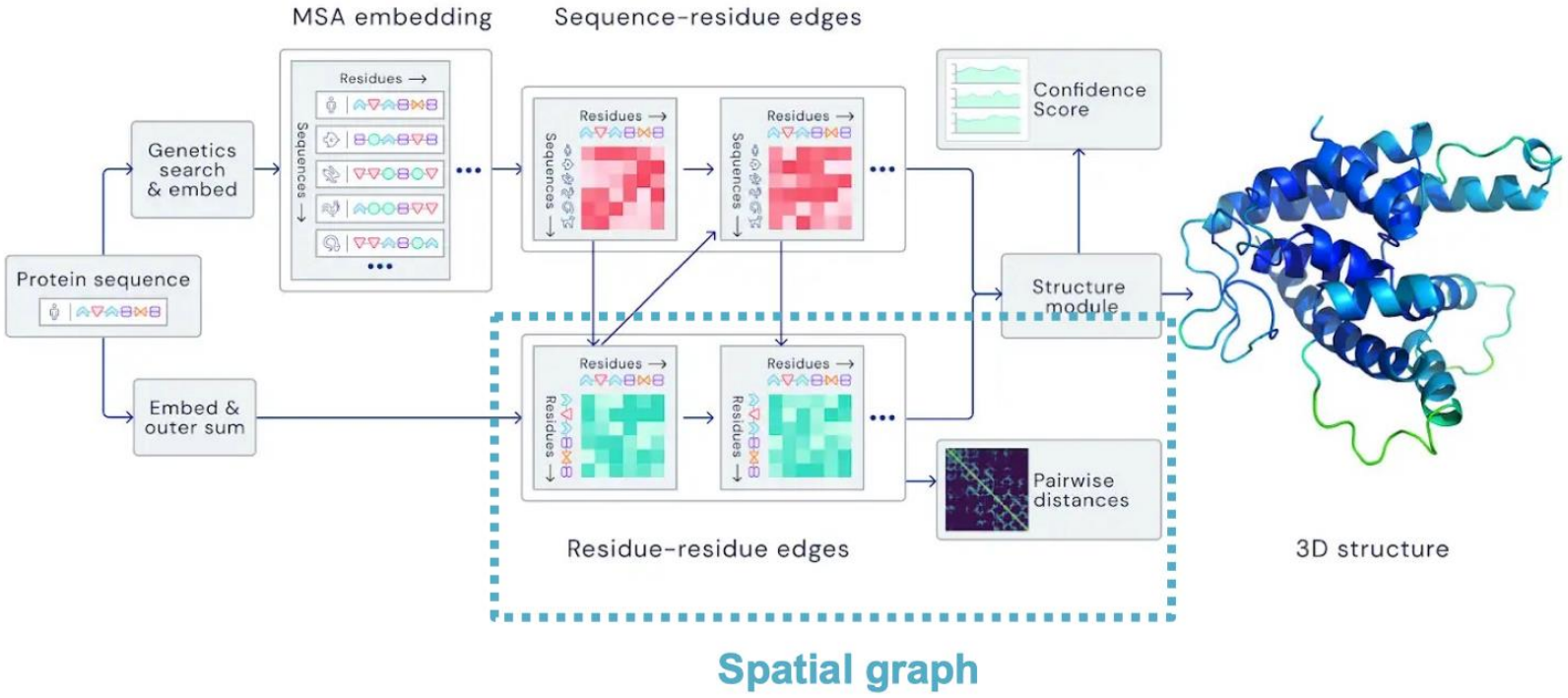
DeepMind's latest AI breakthrough could turbocharge drug discovery





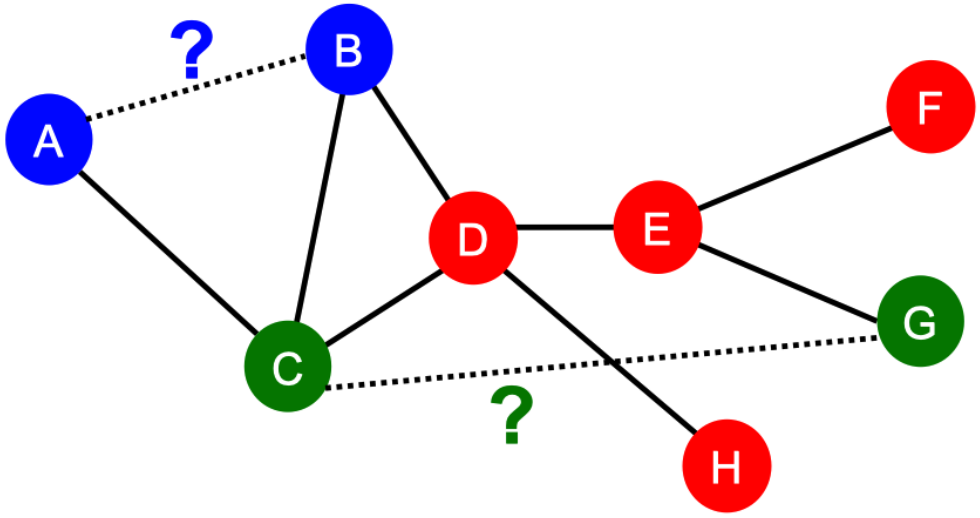
Node-level Task Example: Protein Folding

- **Key idea:** "Spatial graph"
  - **Nodes:** Amino acids in a protein sequence
  - **Edges:** Proximity between amino acids (residues)



Link-Level Prediction Task

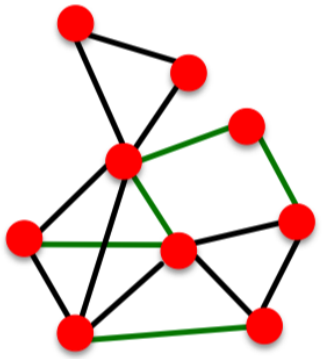
- The task is to predict **new/missing/unknown links** based on the existing links.
- At test time, node pairs (with no existing links) are ranked, and top K node pairs are predicted.
- Task: Make a prediction for **a pair of nodes**.



Link-Level Prediction Task

Two formulations of the link prediction task:

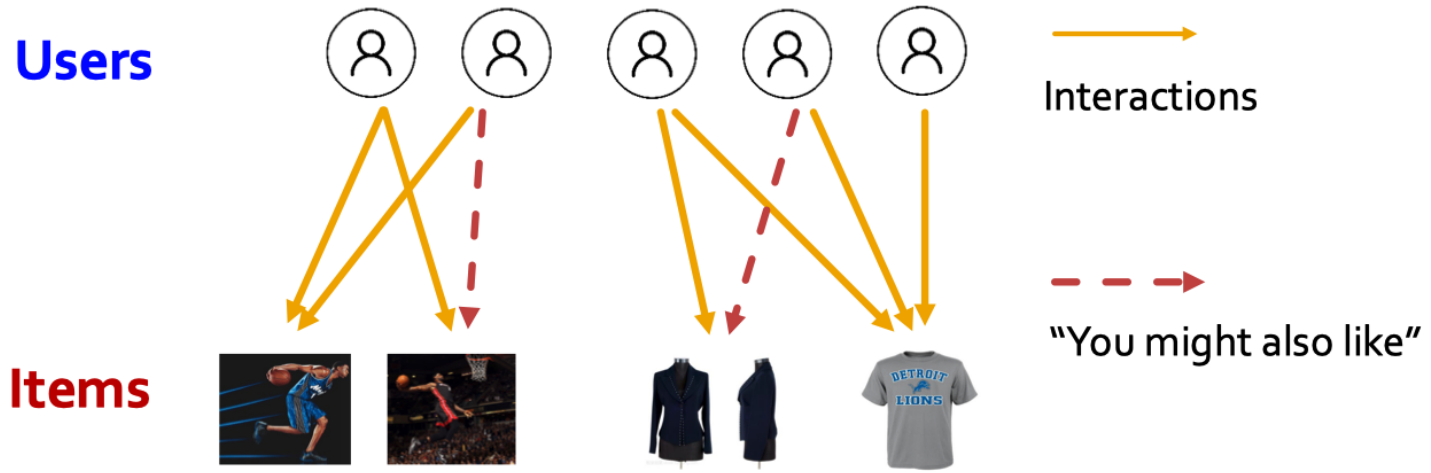
- **1) Links missing at random:**
  - Remove a random set of links and then aim to predict them
- **2) Links over time:**
  - Given $G[t_0, t_0']$ a graph defined by edges up to time t_0', **output a ranked list L** of edges (not in $G[t_0, t_0']$) that are predicted to appear in time $G[t_1, t_1']$
  - **Evaluation:**
    - $n = |E_{new}|$: # new edges that appear during the test period $[t_1, t_1']$
    - Take top n elements of L and count correct edges



$G[t_0, t_0']$
$G[t_1, t_1']$

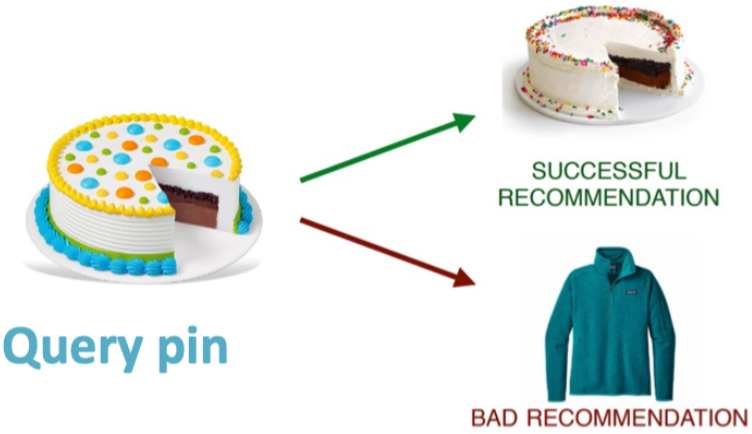
Link-Level Task Example: Recommender System

- **Users interacts with items**
  - Watch movies, buy merchandise, listen to music
  - **Nodes:** Users and items
  - **Edges:** User-item interactions
- **Goal: Recommend items users might like**



Link-Level Task Example: Recommender System

Task: Recommend related pins to users



Task: Learn node embeddings z_i such that

$$d(z_{cake1}, z_{cake2}) < d(z_{cake1}, z_{sweater})$$

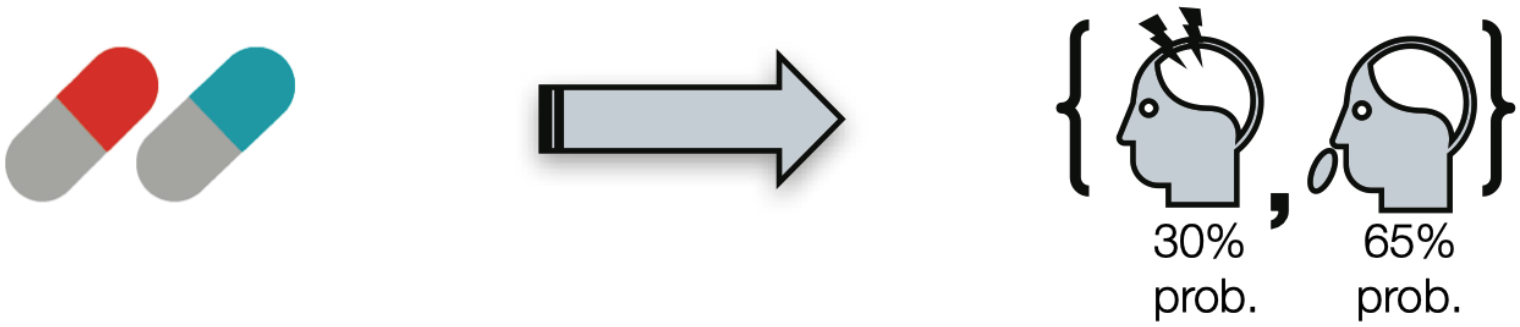
Predict whether two nodes in a graph are related

Link-Level Task Example: Drug Side Effect

Many patients take multiple drugs to treat complex or co-existing diseases:

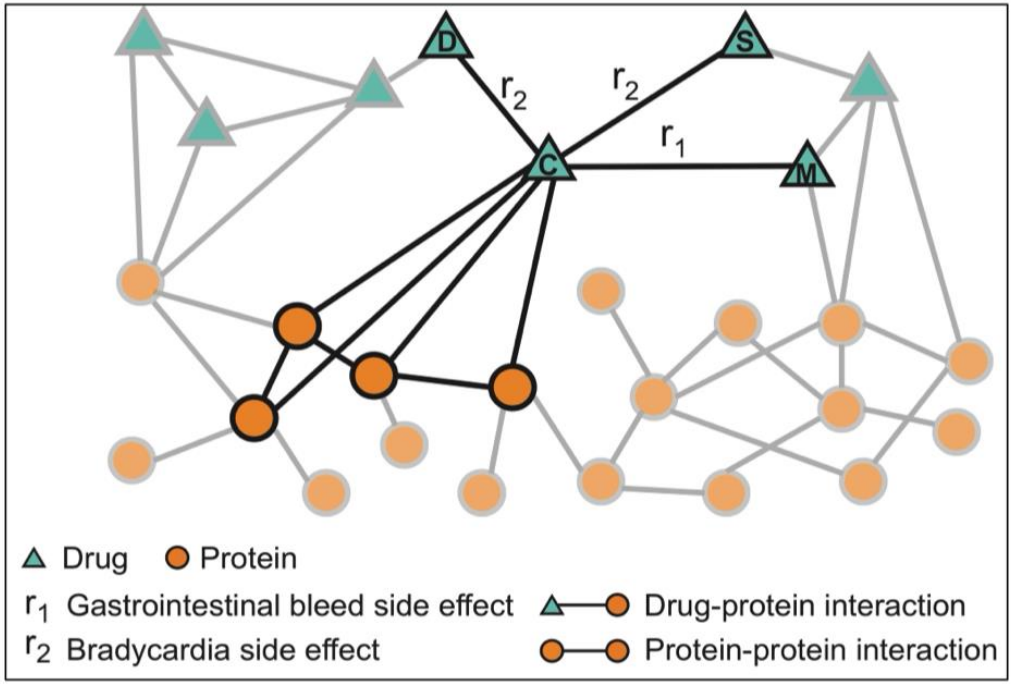
- 46% of people ages 70-79 take more than 5 drugs
- Many patients take more than 20 drugs to treat heart disease, depression, insomnia, etc.

Task: Given a pair of drugs predict adverse side effects

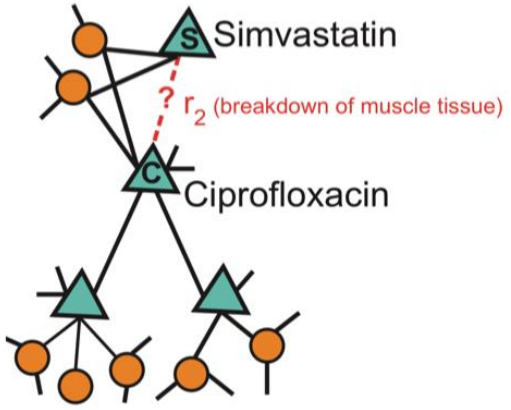


Link-Level Task Example: Drug Side Effect

- **Nodes**: Drugs & Proteins
- **Edges**: Interactions



Query: How likely will Simvastatin and Ciprofloxacin, when taken together, break down muscle tissue?



Link-Level Task Example: Drug Side Effect

Rank	Drug c	Drug d	Side effect r	Evidence found
1	Pyrimethamine	Aliskiren	Sarcoma	Stage *et al.* 2015
2	Tigecycline	Bimatoprost	Autonomic neuropathy	
3	Omeprazole	Dacarbazine	Telangiectases	
4	Tolcapone	Pyrimethamine	Breast disorder	Bicker *et al.* 2017
5	Minoxidil	Paricalcitol	Cluster headache	
6	Omeprazole	Amoxicillin	Renal tubular acidosis	Russo *et al.* 2016
7	Anagrelide	Azelaic acid	Cerebral thrombosis	
8	Atorvastatin	Amlodipine	Muscle inflammation	Banakh *et al.* 2017
9	Aliskiren	Tioconazole	Breast inflammation	Parving *et al.* 2012
10	Estradiol	Nadolol	Endometriosis	

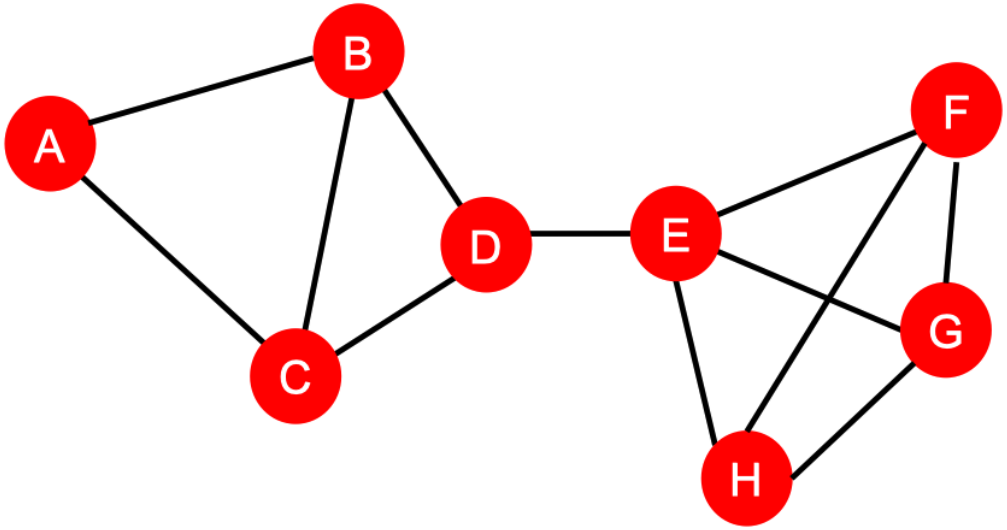
Case Report

Severe Rhabdomyolysis due to Presumed Drug Interactions between Atorvastatin with Amlodipine and Ticagrelor

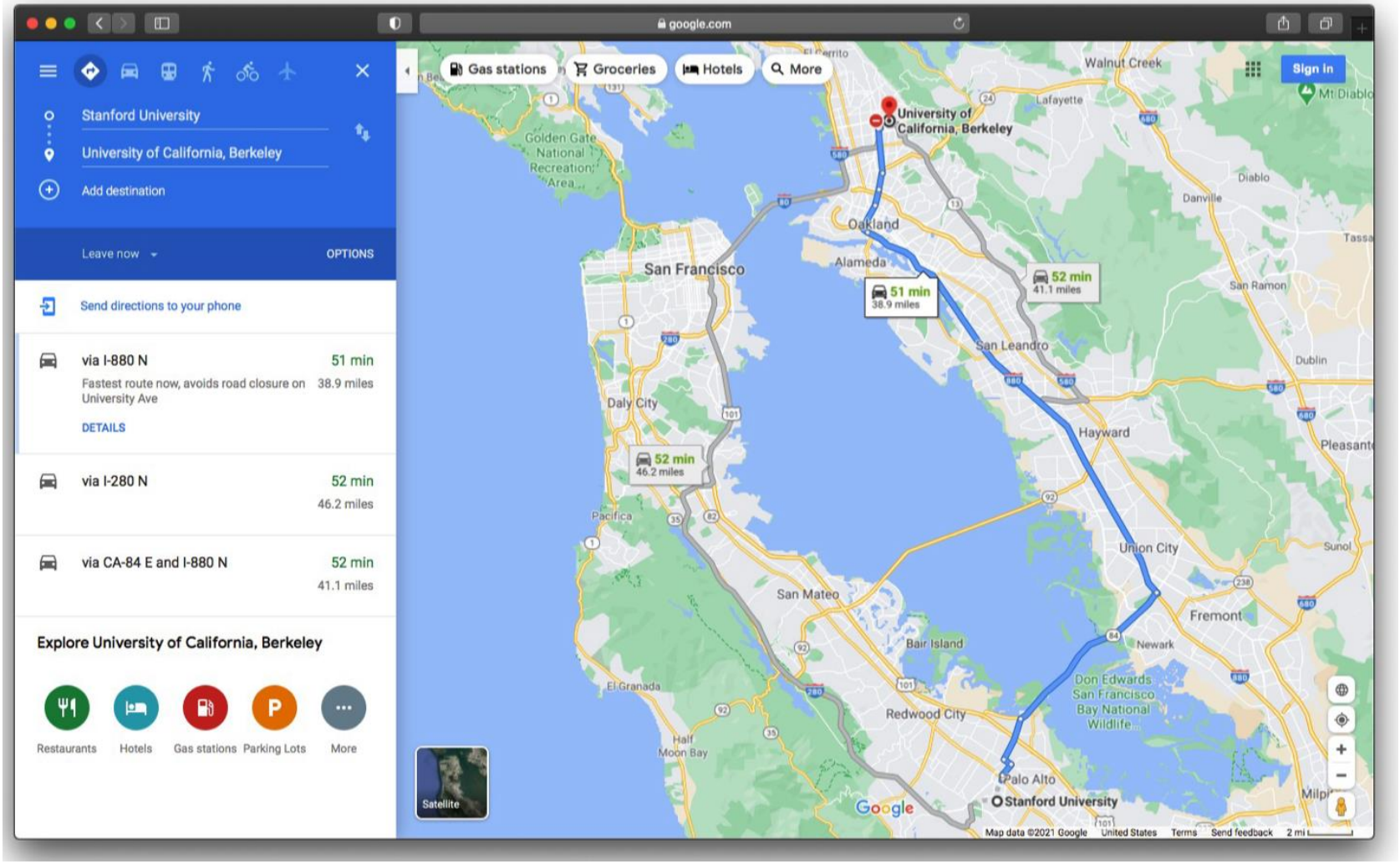
Graph-Level Tasks

- **Goal:** We want make a prediction for an entire graph or a subgraph of the graph.

- **For example:**



Graph-Level Task Example: Traffic Prediction



Graph-Level Task Example: Traffic Prediction

- **Nodes:** Road segments
- **Edges:** Connectivity between road segments
- **Prediction:** Time of Arrival (ETA)

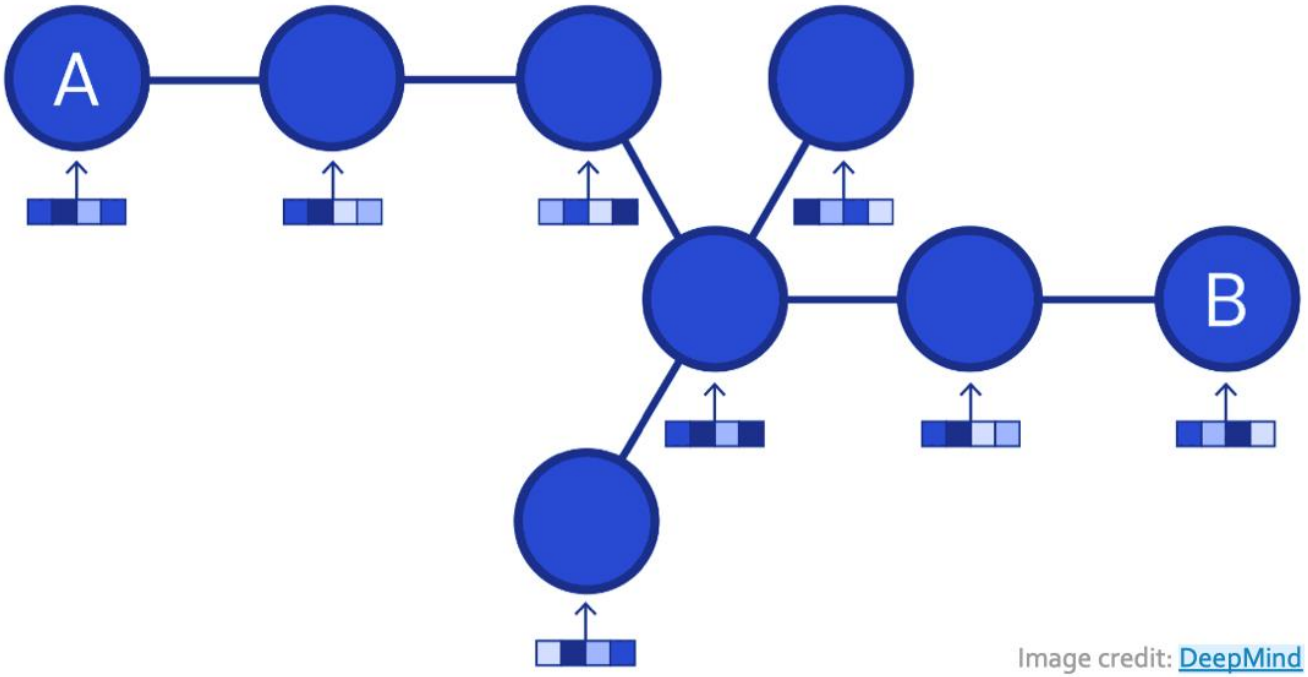
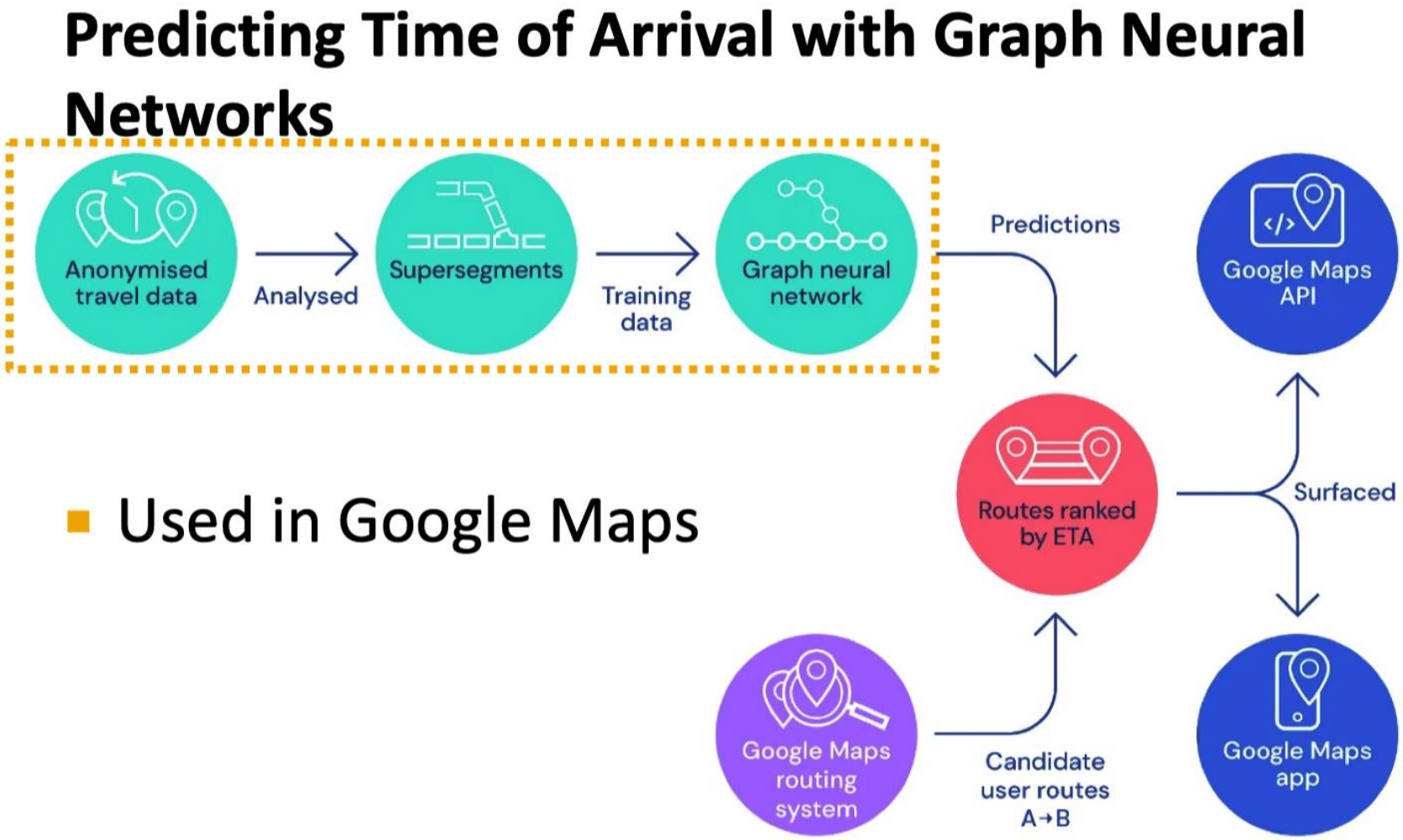


Image credit: DeepMind

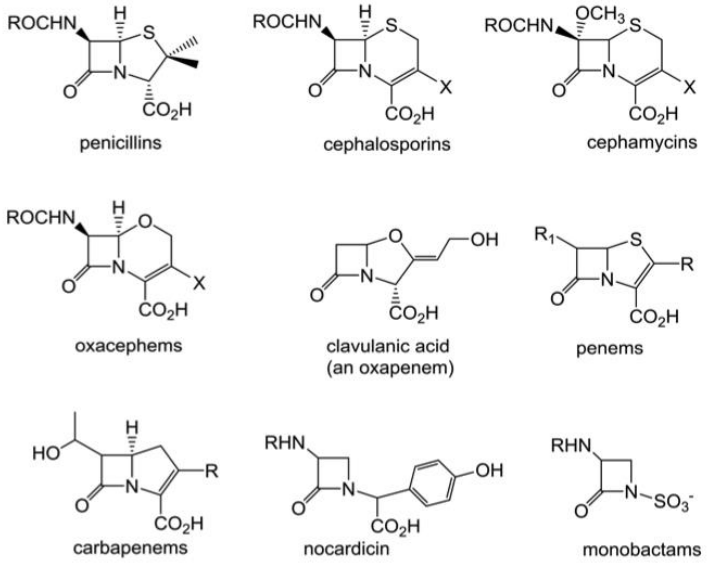
Graph-Level Task Example: Traffic Prediction



- Used in Google Maps

Graph-Level Task Example: Drug Discovery

- **Antibiotics are small molecular graphs**
  - **Nodes:** Atoms
  - **Edges:** Chemical bonds



Konaklieva, Monika I. "Molecular targets of β-lactam-based antimicrobials: beyond the usual suspects." Antibiotics 3.2 (2014): 128-142.

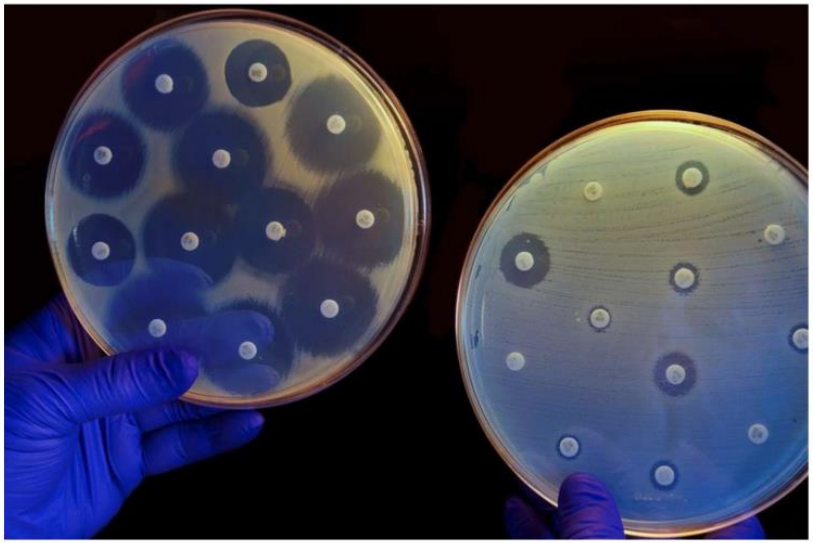
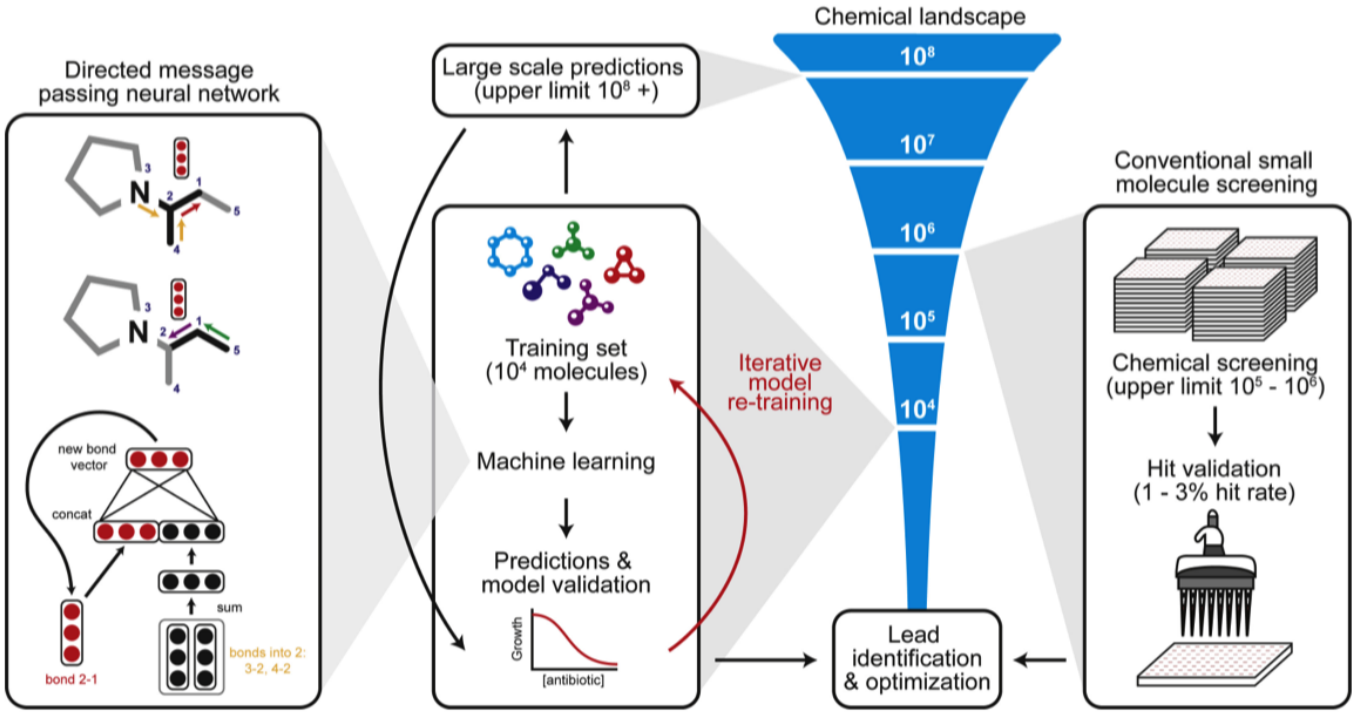


Image credit: CNN

Graph-Level Task Example: Drug Discovery

- A Graph Neural Network **graph classification model**
- Predict promising molecules from a pool of candidates



Stokes, Jonathan M., et al. "A deep learning approach to antibiotic discovery." Cell 180.4 (2020): 688-702.

Graph-Level Task Example: Physics Simulation

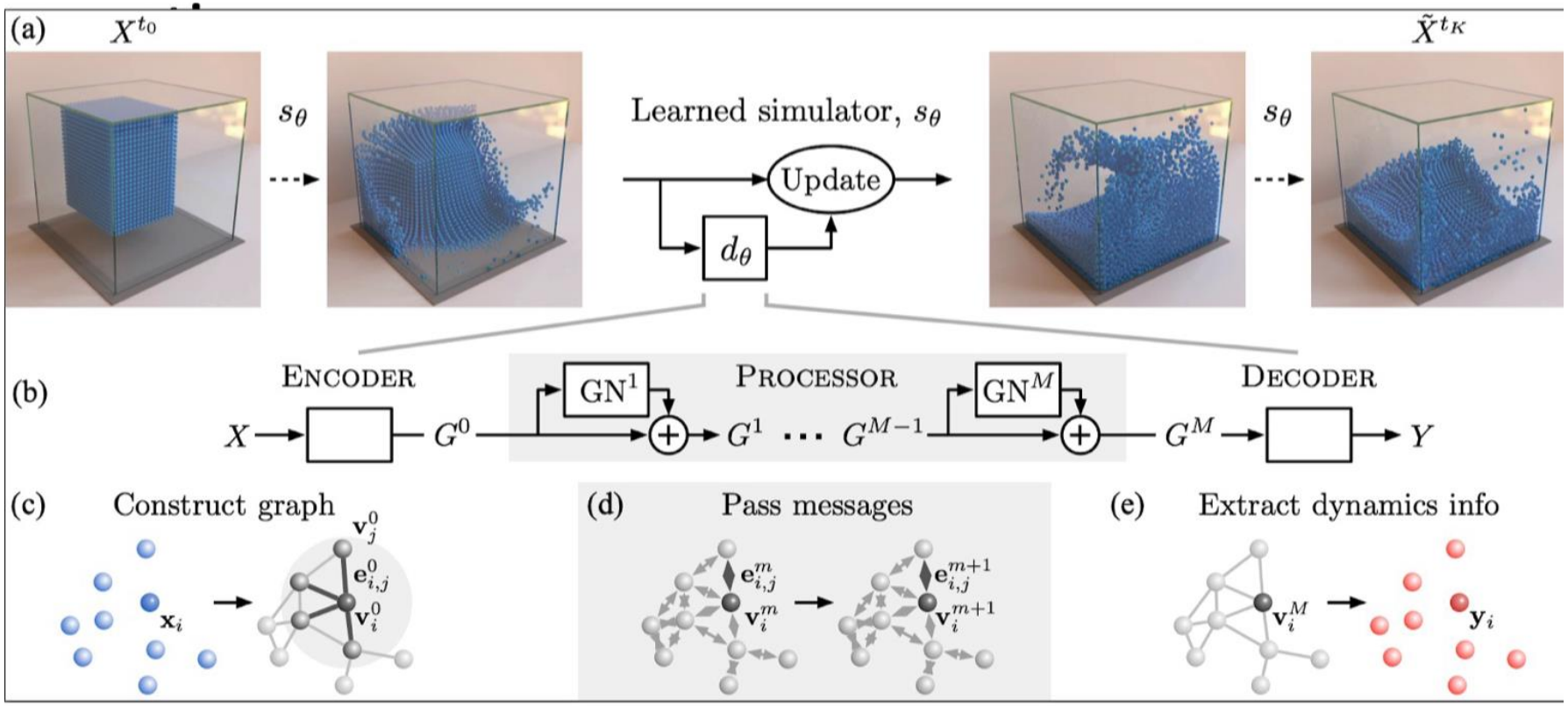
Physical simulation as a graph:

- **Nodes**: Particles
- **Edges**: Interaction between particles

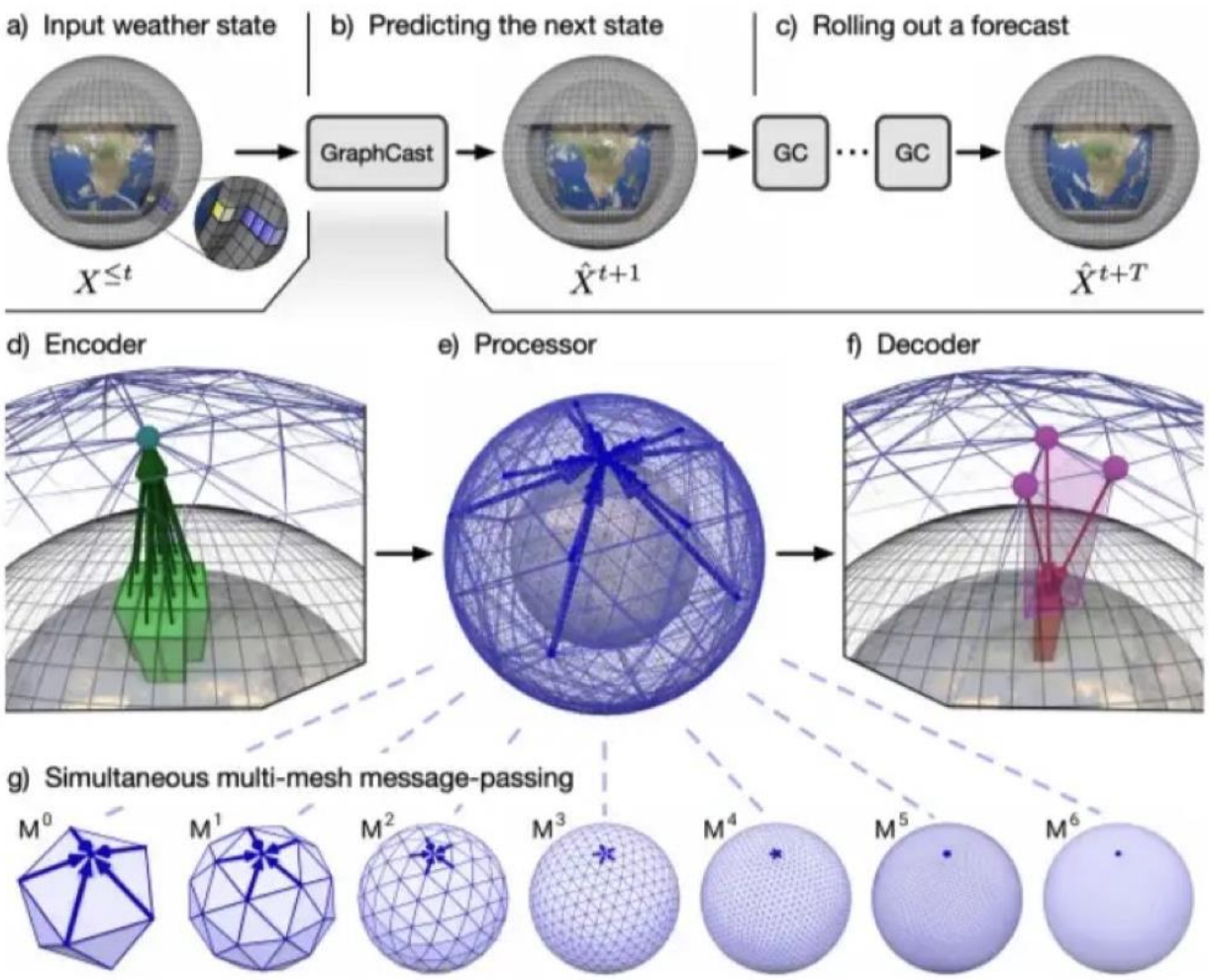
Graph-Level Task Example: Physics Simulation

A graph evolution task:

- **Goal**: Predict how a graph will evolve over



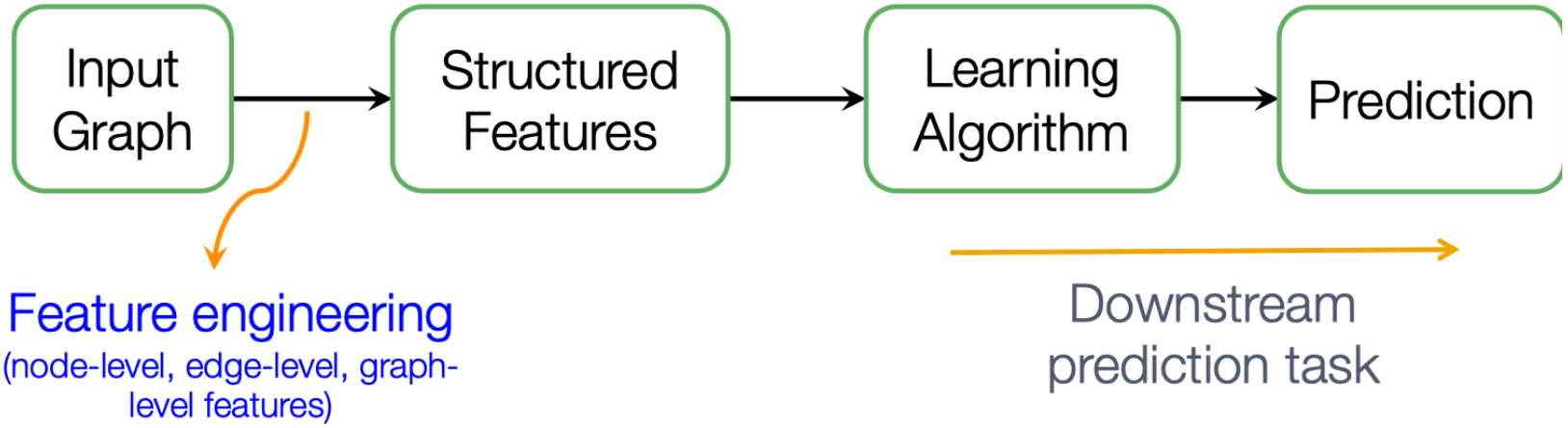
Graph-Level Task Example: Weather Forecasting



https://medium.com/syncedreview/deepmind-googles-ml-based-graphcast-outperforms-the-world-s-best-medium-range-weather-9d114460aa0c

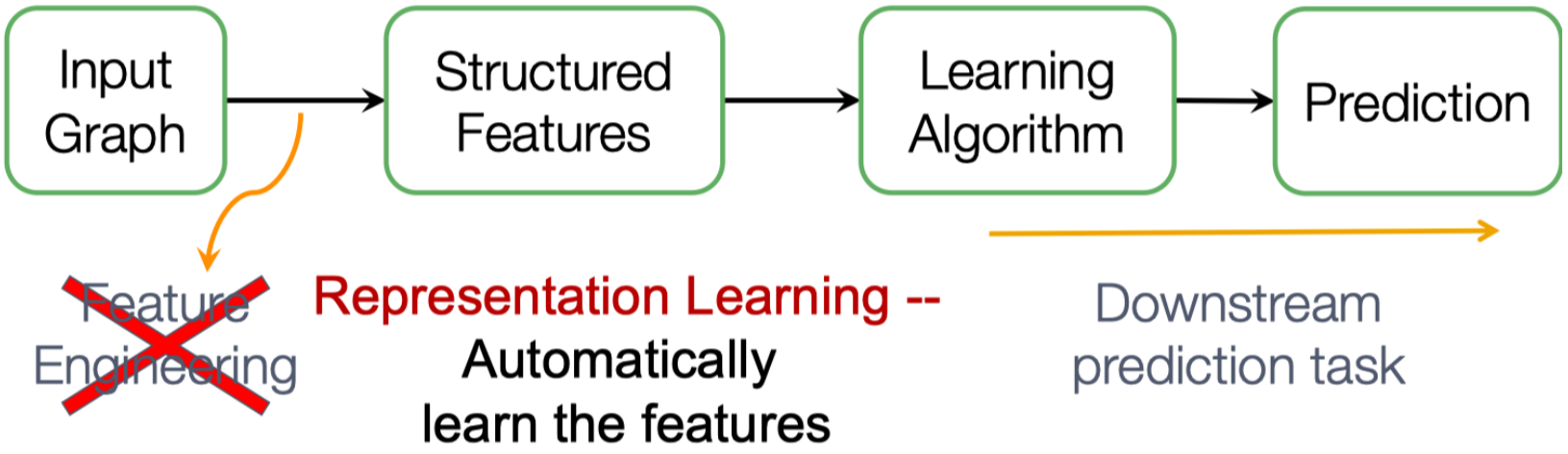
Traditional ML for Graphs

Given an input graph, extract node, link and graph-level features, then learn a model (SVM, neural network, etc.) that maps features to labels.



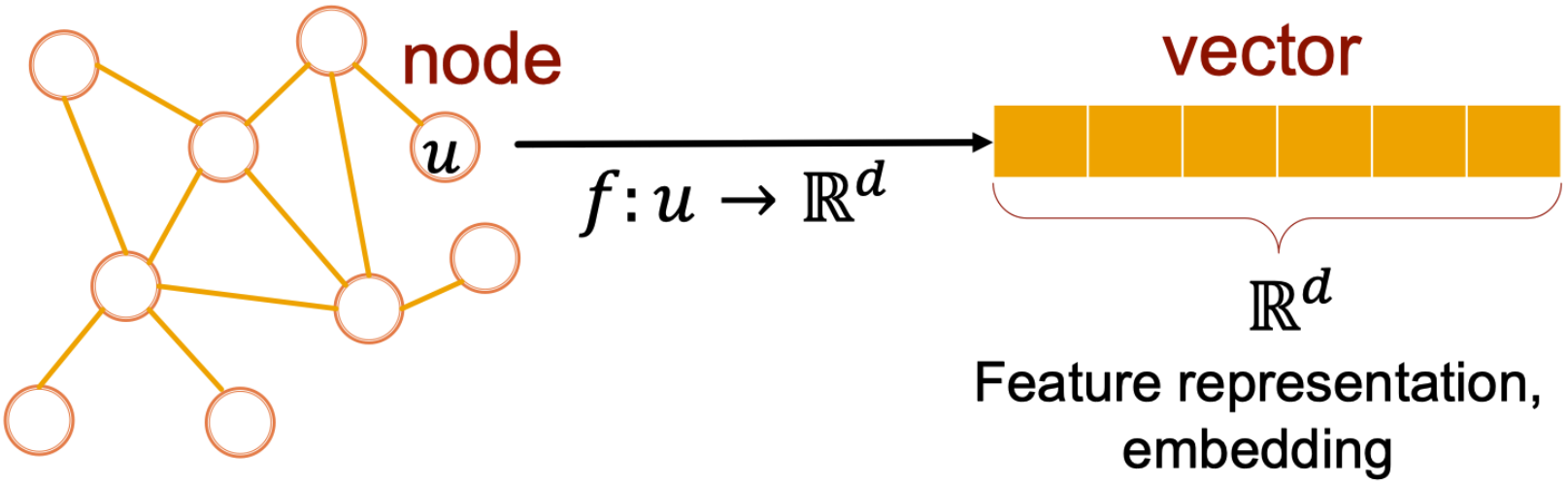
Graph Representation Learning

Graph Representation Learning alleviates the need to do feature engineering every single time.



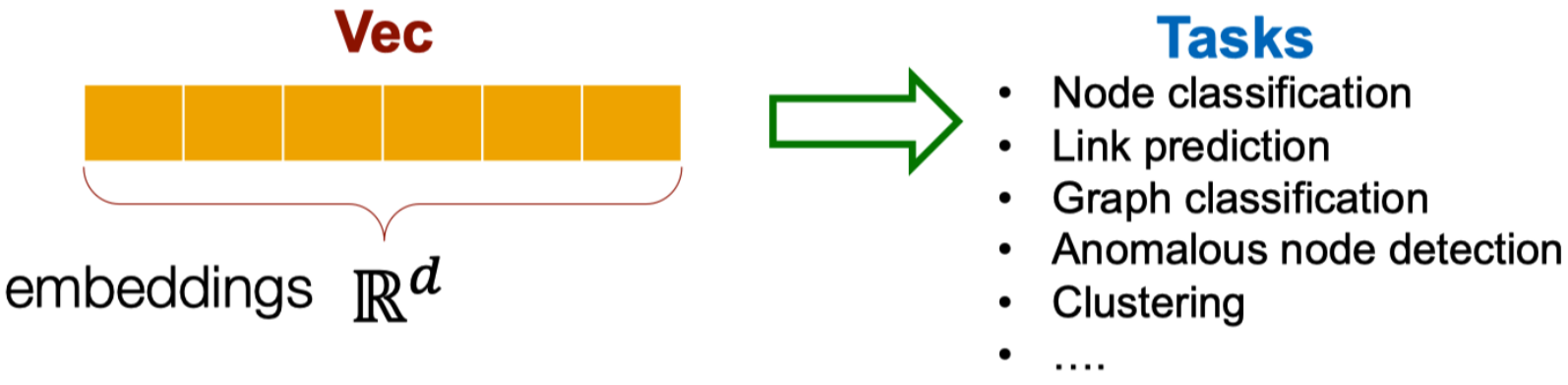
Graph Representation Learning

Goal: Efficient task-independent feature learning for machine learning with graphs!



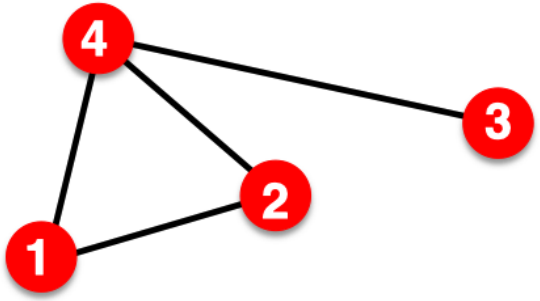
Why Embedding

- **Task: Map nodes into an embedding space**
  - Similarity of embeddings between nodes indicates their similarity in the network. For example:
    - Both nodes are close to each other (connected by an edge)
  - Encode network information
  - Potentially used for many downstream predictions



Setup

- **Assume we have an (undirected) graph G:**
  - V is the vertex set.
  - A is the adjacency matrix (assume binary).
  - **For simplicity: No node features or extra information is used**

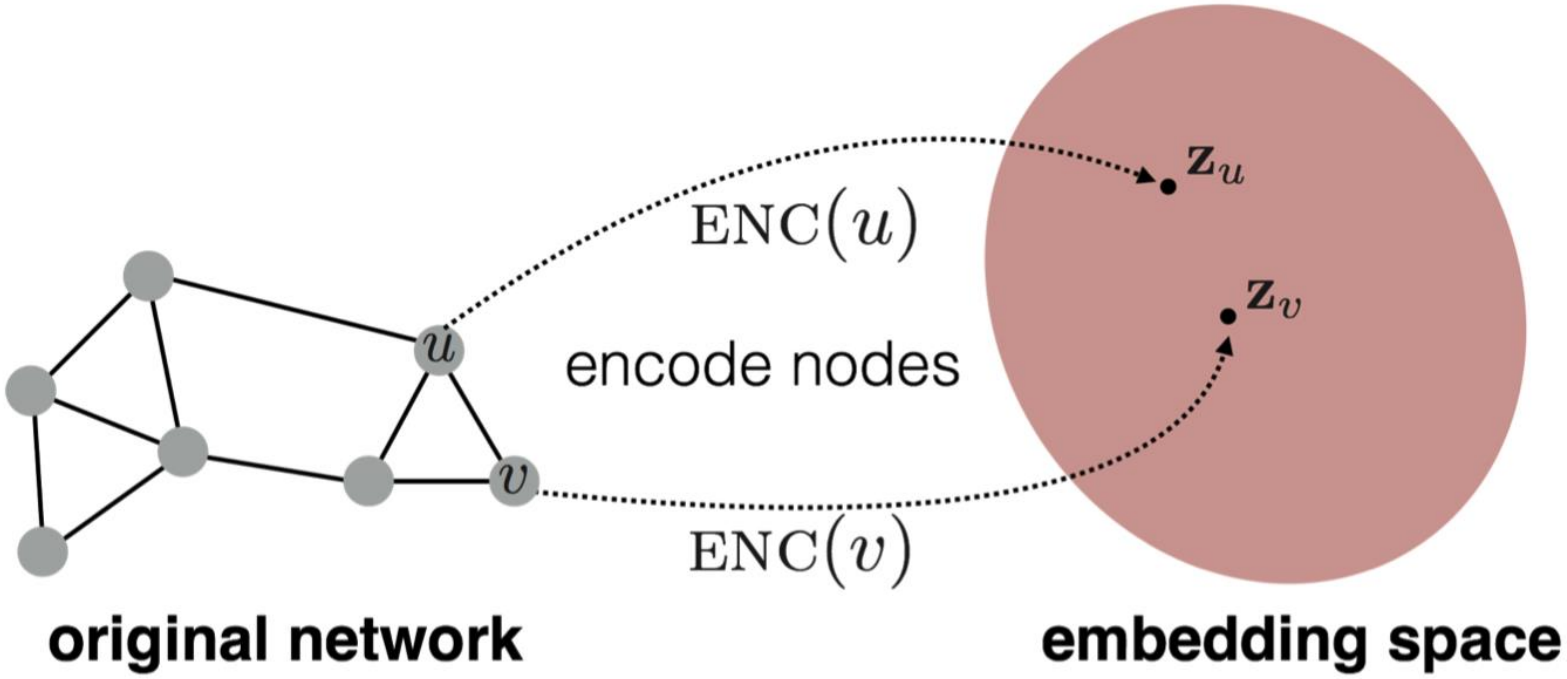


V: {1, 2, 3, 4}

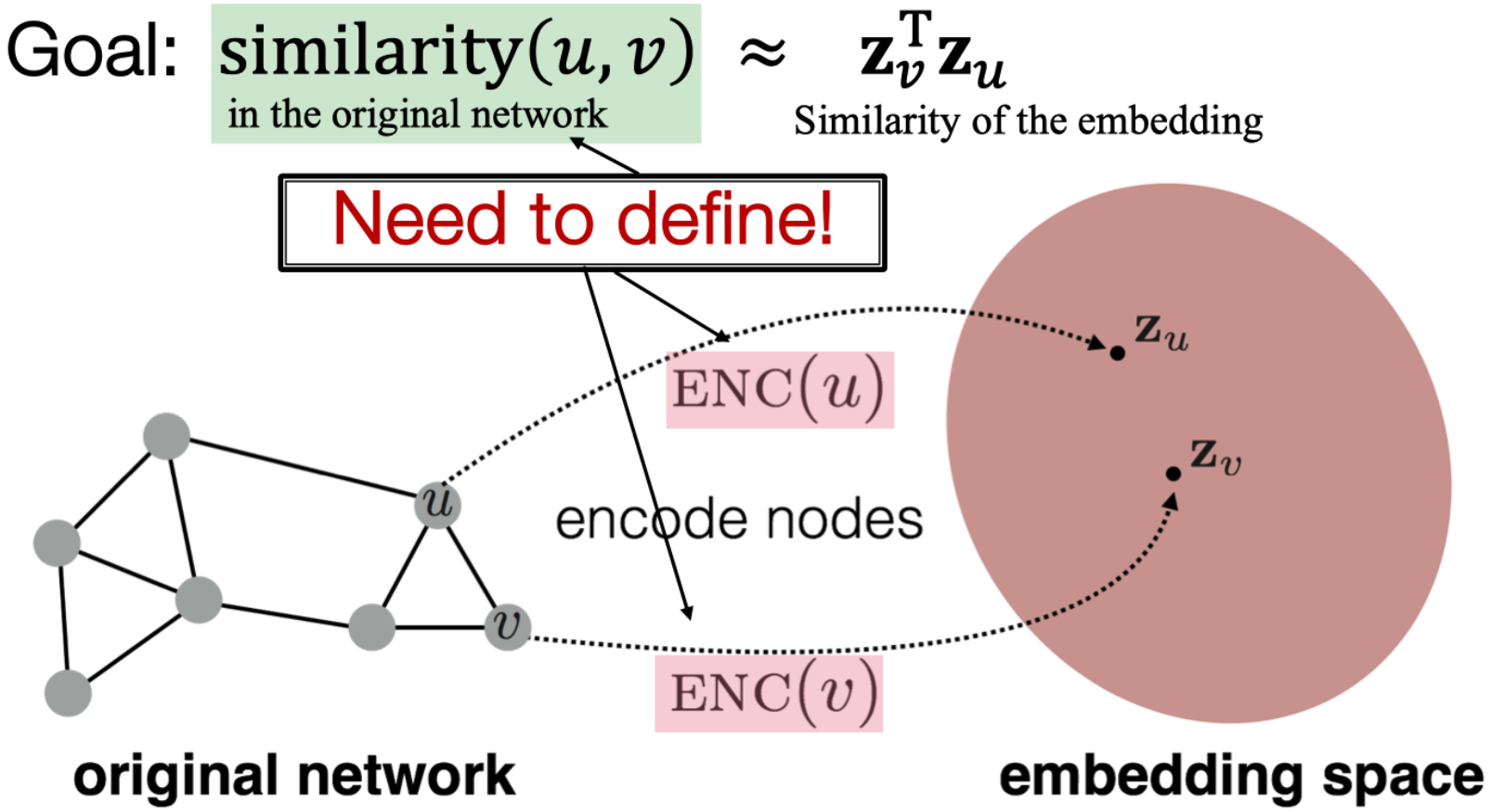
$$A = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

Embedding Notes

- Goal is to encode nodes so that similarity in the embedding space (e.g., dot product) approximates similarity in the graph



Embedding Notes



Learning Note Embeddings

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

$\mathrm{DEC}(\mathbf{z}_v^{\mathrm{T}}\mathbf{z}_u)$

$$\underbrace{\text{similarity}(u, v)}_{\text{in the original network}} \approx \underbrace{\mathbf{z}_v^{\mathrm{T}}\mathbf{z}_u}_{\text{Similarity of the embedding}}$$

Components

- **Encoder:** maps each node to a low-dimensional vector

$$\text{ENC}(v) = \mathbf{z}_v$$

d-dimensional embedding

node in the input graph

- **Similarity function:** specifies how the relationships in vector space map to the relationships in the original network

$$\text{similarity}(u, v) \approx \mathbf{z}_v^{\mathrm{T}} \mathbf{z}_u$$

Decoder

Similarity of u and v in the original network

dot product between node embeddings

Shallow Encoding

Simplest encoding approach: **Encoder is just an embedding-lookup**

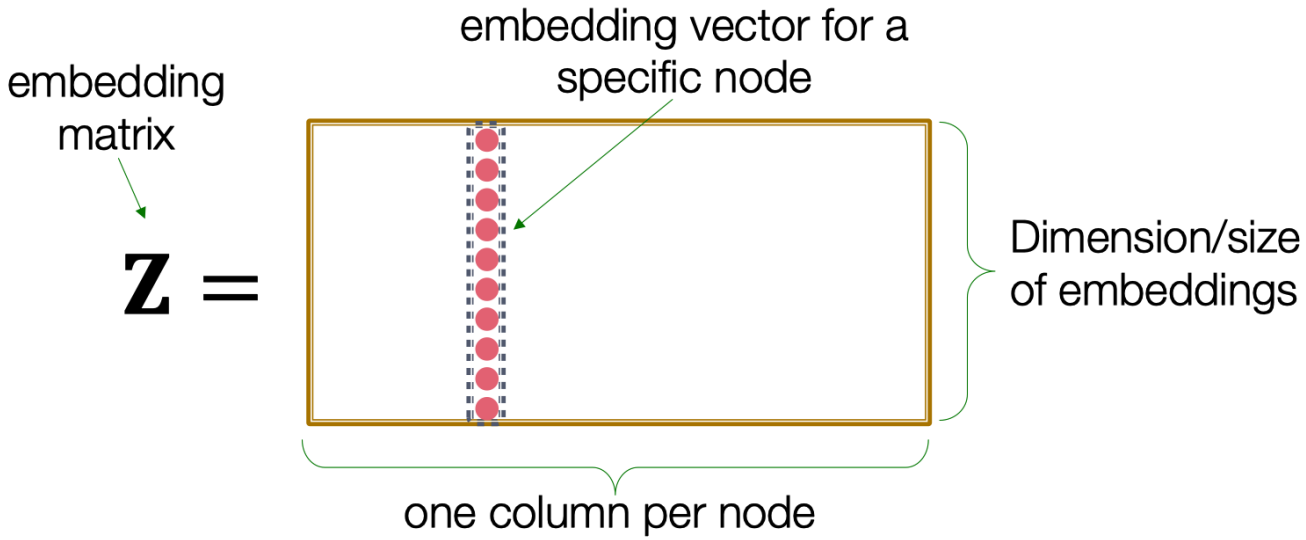
$$\text{ENC}(v) = \mathbf{z}_v = \mathbf{Z} \cdot v$$

$\mathbf{Z} \in \mathbb{R}^{d \times |\mathcal{V}|}$ matrix, each column is a node embedding [what we learn / optimize]

$v \in \mathbb{I}^{|\mathcal{V}|}$ indicator vector, all zeroes except a one in column indicating node *v*

Shallow Encoding

Simplest encoding approach: **encoder is just an embedding-lookup**



Each node is assigned a unique embedding vector

(i.e., we directly optimize the embedding of each node)

Framework Summary

- **Encoder + Decoder Framework**
  - Shallow encoder: Embedding lookup
  - Parameters to optimize: $\mathbf{Z}$ which contains node embeddings $\mathbf{z}_u$ for all nodes $u \in V$
  - We will cover deep encoders in the GNNs

  - **Decoder:** based on node similarity.
  - **Objective:** maximize $\mathbf{z}_v^{\mathrm{T}} \mathbf{z}_u$ for node pairs (u, v) that are **similar**

How to Define Node Similarity

- Key choice of methods is **how they define node similarity.**
- Should two nodes have a similar embedding if they…
  - are linked?
  - share neighbors?
  - have similar "structural roles"?
- We will now learn node similarity definition that uses **random walks**, and how to optimize embeddings for such a similarity measure.

Unsupervised Node Embedding Learning

- This is **unsupervised/self-supervised** way of learning node embeddings.
  - We are **not** utilizing node labels
  - We are **not** utilizing node features
  - The goal is to directly estimate a set of coordinates (i.e., the embedding) of a node so that some aspect of the network structure (captured by DEC) is preserved.
- These embeddings are **task independent:**
  - They are not trained for a specific task but can be used for any task.

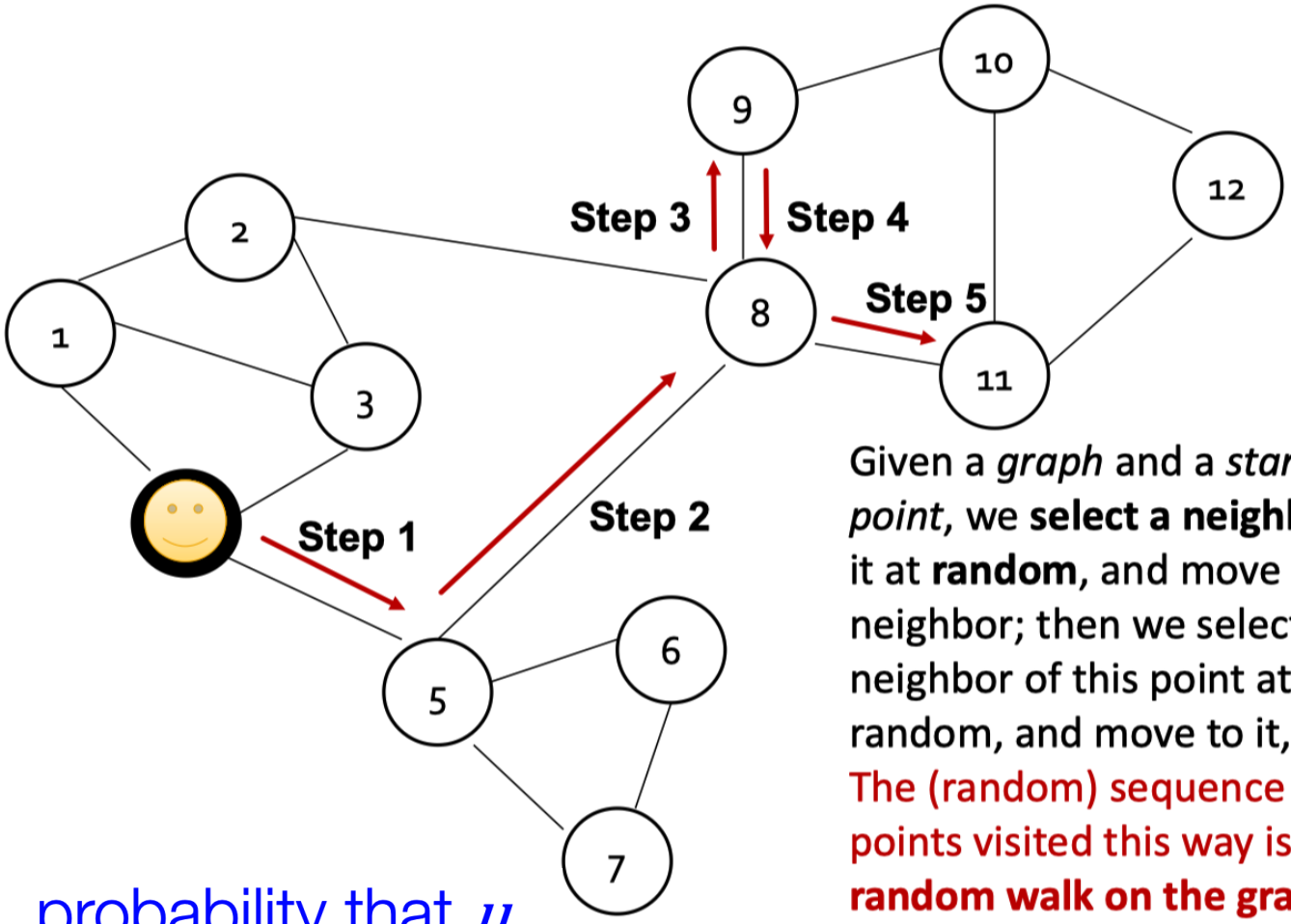
Random Walk

- **Vector $\mathbf{z}_u$:**
  - The embedding of node u (what we aim to find).
- **Probability $P(v \mid \mathbf{z}_u)$:** ⟸ Our model prediction based on $\mathbf{z}_u$
  - The **(predicted) probability** of visiting node v on random walks starting from node u.

Non-linear functions used to produce predicted **probabilities**

- **Softmax** function:
  - Turns vector of K real values (model predictions) into K probabilities that sum to 1: $S(\boldsymbol{z})[i] = \frac{e^{\boldsymbol{z}[i]}}{\sum_{j=1}^{K} e^{\boldsymbol{z}[j]}}$
- **Sigmoid** function:
  - S-shaped function that turns real values into the range of (0, 1). Written as $\sigma(x) = \frac{1}{1+e^{-x}}$.

Random Walk

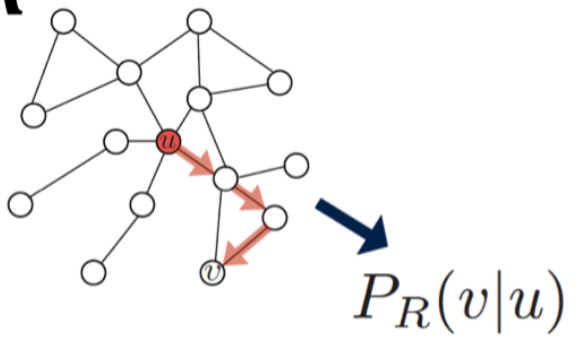


Given a *graph* and a *starting point*, we **select a neighbor** of it at **random**, and move to this neighbor; then we select a neighbor of this point at random, and move to it, etc. The (random) sequence of points visited this way is a **random walk on the graph**.

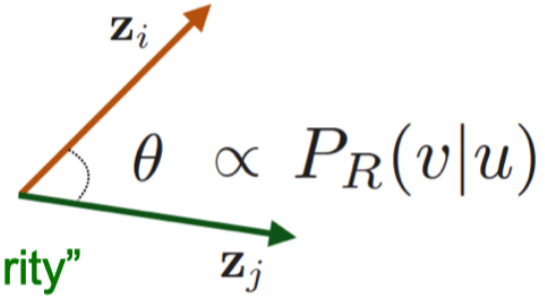
$\mathbf{z}_u^{\mathrm{T}}\mathbf{z}_v \approx$ probability that u and v co-occur on a random walk over the graph

Random Walk Embeddings

1. **Estimate probability of visiting node v on a random walk starting from node u using some random walk strategy R**



2. **Optimize embeddings to encode these random walk statistics:**



Similarity in embedding space (Here: dot product=$\cos(\theta)$) encodes random walk "similarity"

Why Random Walks

1. **Expressivity:** Flexible stochastic definition of node similarity that incorporates both local and higher-order neighborhood information
   Idea: if random walk starting from node u visits v with high probability, u and v are similar (high-order multi-hop information)

2. **Efficiency:** Do not need to consider all node pairs when training; only need to consider pairs that co-occur on random walks

Unsupervised Feature Learning

- Intuition: Find embedding of nodes in d-dimensional space that preserves similarity
- Idea: Learn node embedding such that nearby nodes are close together in the network
- Given a node u, how do we define nearby nodes?
  - $N_R(u)$ … neighbourhood of u obtained by some random walk strategy R

Feature Learning as Optimization

- Given $G = (V, E)$,
- Our goal is to learn a mapping $f: u \rightarrow \mathbb{R}^d$:
  $f(u) = \mathbf{z}_u$

- Log-likelihood objective:

$$\arg\max_{z} \sum_{u \in V} \log \mathrm{P}(N_{\mathrm{R}}(u) | \mathbf{z}_u)$$

  - $N_R(u)$ is the neighborhood of node u by strategy R

- Given node u, we want to learn feature representations that are predictive of the nodes in its random walk neighborhood $N_R(u)$.

Random Walk Optimization

1. Run **short fixed-length random walks** starting from each node u in the graph using some random walk strategy R.
2. For each node u collect $N_R(u)$, the multiset* of nodes visited on random walks starting from u.
3. Optimize embeddings according to: Given node u, predict its neighbors $N_R(u)$.

$$\arg\max_{z} \sum_{u \in V} \log \mathrm{P}(N_{\mathrm{R}}(u) | \mathbf{z}_u)$$ ⟹ Maximum likelihood objective

Random Walk Optimization

Equivalently,

$$\arg\min_{z} \mathcal{L} = \sum_{u \in V} \sum_{v \in N_R(u)} -\log(P(v|\mathbf{z}_u))$$

- **Intuition:** Optimize embeddings $\mathbf{z}_u$ to minimize the negative log-likelihood of random walk neighborhoods $N(u)$.
- **Parameterize $P(v|\mathbf{z}_u)$ using softmax:**

$$P(v|\mathbf{z}_u) = \frac{\exp(\mathbf{z}_u^{\mathrm{T}}\mathbf{z}_v)}{\sum_{n \in V} \exp(\mathbf{z}_u^{\mathrm{T}}\mathbf{z}_n)}$$

Why softmax?
We want node v to be most similar to node u (out of all nodes n).
Intuition: $\sum_i \exp(x_i) \approx \max_i \exp(x_i)$

Random Walk Optimization

Putting it all together:

$$\mathcal{L} = \sum_{u \in V} \sum_{v \in N_R(u)} -\log\left(\frac{\exp(\mathbf{z}_u^T \mathbf{z}_v)}{\sum_{n \in V} \exp(\mathbf{z}_u^T \mathbf{z}_n)}\right)$$

sum over all nodes u

sum over nodes v seen on random walks starting from u

predicted probability of u and v co-occuring on random walk

Optimizing random walk embeddings =

Finding embeddings $\mathbf{z}_u$ that minimize $\mathcal{L}$

Random Walk Optimization

But doing this naively is too expensive!

$$\mathcal{L} = \sum_{u \in V} \sum_{v \in N_R(u)} -\log\left(\frac{\exp(\mathbf{z}_u^{\mathrm{T}} \mathbf{z}_v)}{\sum_{n \in V} \exp(\mathbf{z}_u^{\mathrm{T}} \mathbf{z}_n)}\right)$$

Nested sum over nodes gives $O(|V|^2)$ complexity!

Random Walk Optimization

But doing this naively is too expensive!

$$\mathcal{L} = \sum_{u \in V} \sum_{v \in N_R(u)} -\log\left(\frac{\exp(\mathbf{z}_u^{\mathrm{T}} \mathbf{z}_v)}{\sum_{n \in V} \exp(\mathbf{z}_u^{\mathrm{T}} \mathbf{z}_n)}\right)$$

The normalization term from the softmax is the culprit… can we approximate it?

Random Walk Optimization

- **Solution:** Negative sampling

$$-\log\left(\frac{\exp\left(\mathbf{z}_u^{\mathrm{T}}\mathbf{z}_v\right)}{\sum_{n\in V}\exp\left(\mathbf{z}_u^{\mathrm{T}}\mathbf{z}_n\right)}\right)$$

Why is the approximation valid?
Technically, this is a different objective. But Negative Sampling is a form of Noise Contrastive Estimation (NCE) which approx. maximizes the log probability of softmax.

New formulation corresponds to using a logistic regression (sigmoid func.) to distinguish the target node v from nodes n_i sampled from background distribution P_v.

More at https://arxiv.org/pdf/1402.3722.pdf

$$\approx \log\left(\sigma\left(\mathbf{z}_u^{\mathrm{T}}\mathbf{z}_v\right)\right) + \sum_{i=1}^{k}\log\left(\sigma\left(-\mathbf{z}_u^{\mathrm{T}}\mathbf{z}_{n_i}\right)\right),\ n_i \sim P_V$$

sigmoid function
(makes each term a "probability" between 0 and 1)

random distribution over nodes

Instead of normalizing w.r.t. all nodes, just normalize against k random "**negative samples**" n_i

- Negative sampling allows for quick likelihood calculation.

Random Walk Optimization

$$\log\left(\frac{\exp\left(\mathbf{z}_u^{\mathrm{T}}\mathbf{z}_v\right)}{\sum_{n\in V}\exp\left(\mathbf{z}_u^{\mathrm{T}}\mathbf{z}_n\right)}\right)$$

random distribution over nodes

$$\approx \log\left(\sigma\left(\mathbf{z}_u^{\mathrm{T}}\mathbf{z}_v\right)\right) + \sum_{i=1}^{k}\log\left(\sigma\left(-\mathbf{z}_u^{\mathrm{T}}\mathbf{z}_{n_i}\right)\right),\ n_i \sim P_V$$

- Sample k negative nodes n_i each with prob. proportional to its degree.
- Two considerations for k (# negative samples):
  1. Higher k gives more robust estimates
  2. Higher k corresponds to higher bias on negative events

  In practice k =5-20.

Can negative sample be any node or only the nodes not on the walk? People often sample any node (for efficiency).

Random Walk Optimization

- After we obtained the objective function, how do we optimize (minimize) it?

$$\mathcal{L} = \sum_{u \in V} \sum_{v \in N_R(u)} -\log(P(v|\mathbf{z}_u))$$

- **Stochastic Gradient Descent**: Instead of evaluating gradients over all examples, evaluate it for each **individual** training example.
  - Initialize z_u at some randomized value for all nodes u.
  - Iterate until convergence: $\mathcal{L}^{(u)} = \sum_{v \in N_R(u)} -\log(P(v|\mathbf{z}_u))$
    - Sample a node u, for all v calculate the gradient $\frac{\partial \mathcal{L}^{(u)}}{\partial z_v}$.
    - For all v, update: $z_v \leftarrow z_v - \eta \frac{\partial \mathcal{L}^{(u)}}{\partial z_v}$.

Random Walk Summary

1. Run **short fixed-length** random walks starting from each node on the graph
2. For each node u collect $N_R(u)$, the multiset of nodes visited on random walks starting from u.
3. Optimize embeddings Z using Stochastic Gradient Descent:

$$\mathcal{L} = \sum_{u \in V} \sum_{v \in N_R(u)} -\log(P(v|\mathbf{z}_u))$$

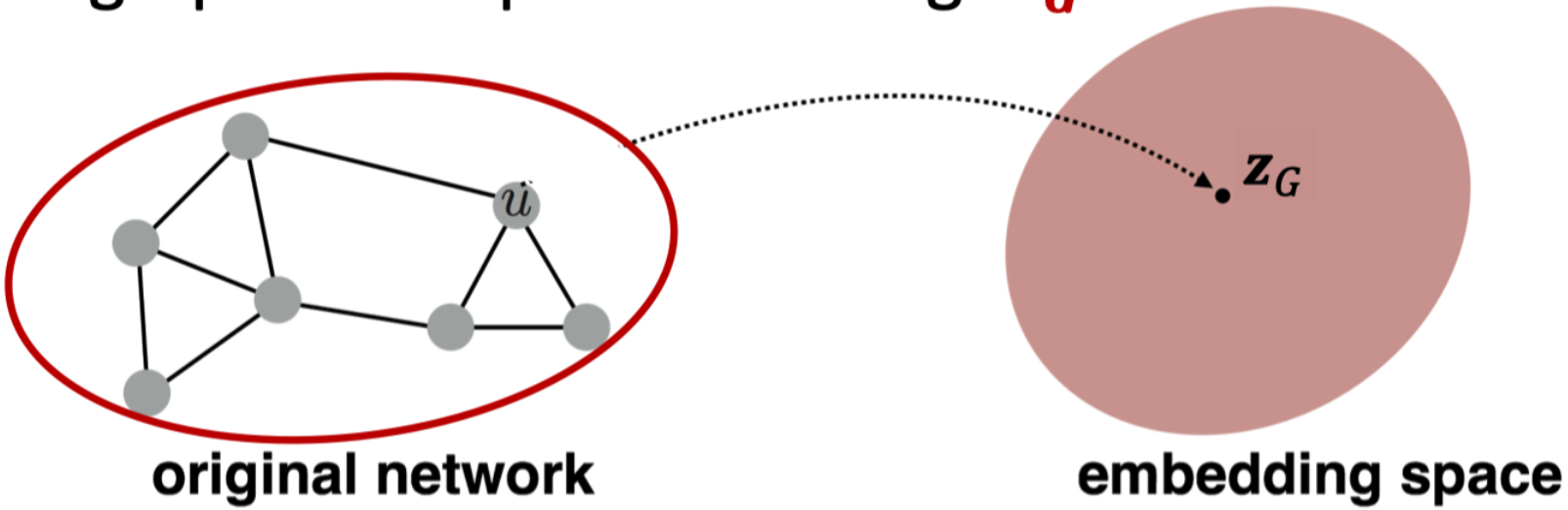
We can efficiently approximate this using negative sampling!

How Should We Randomly Walk

- So far we have described how to optimize embeddings given a random walk strategy R
- **What strategies should we use to run these random walks?**
  - Simplest idea: **Just run fixed-length, unbiased random walks starting from each node** (i.e., DeepWalk from Perozzi et al., 2013)
    - The issue is that such notion of similarity is too constrained
- **Different kinds of biased random walks:**
  - Based on node attributes (Dong et al., 2017).
  - Based on learned weights (Abu-El-Haija et al., 2017)
- **Alternative optimization schemes:**
  - Directly optimize based on 1-hop and 2-hop random walk probabilities (as in LINE from Tang et al. 2015).

Embedding Entire Graphs

- **Goal:** Want to embed a subgraph or an entire graph G. Graph embedding: $\mathbf{z}_G$.



- **Tasks:**
  - Classifying toxic vs. non-toxic molecules
  - Identifying anomalous graphs

Approach 1

Simple (but effective) approach 1:

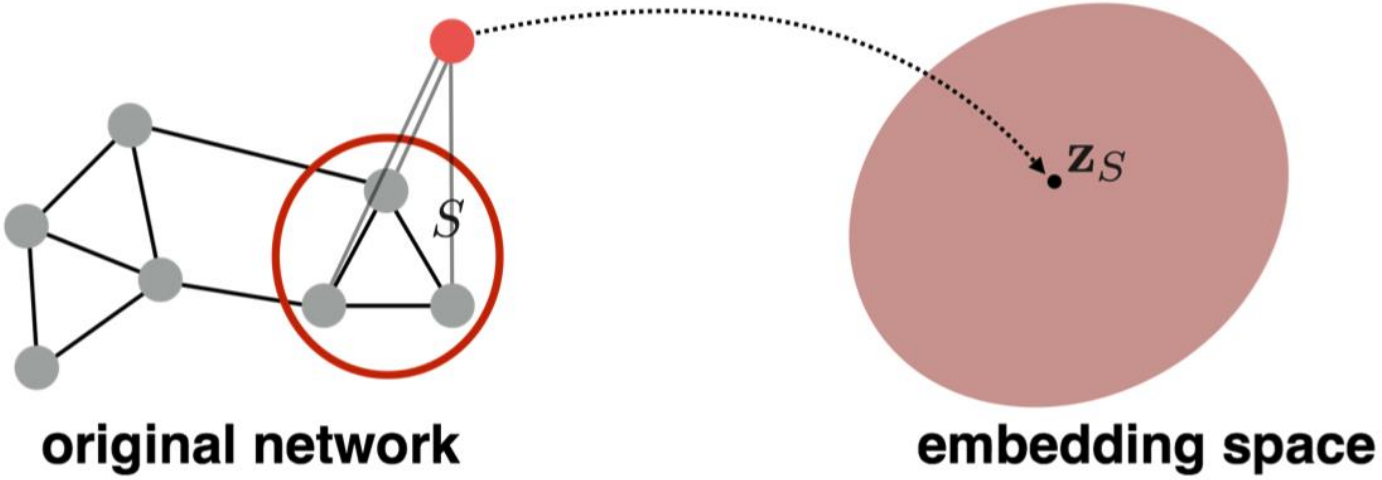
- Run a standard graph embedding technique *on* the (sub)graph G.
- Then just sum (or average) the node embeddings in the (sub)graph G.

$$\mathbf{z}_G = \sum_{v \in G} \mathbf{z}_v$$

- Used by Duvenaud et al., 2016 to classify molecules based on their graph structure

Approach 2

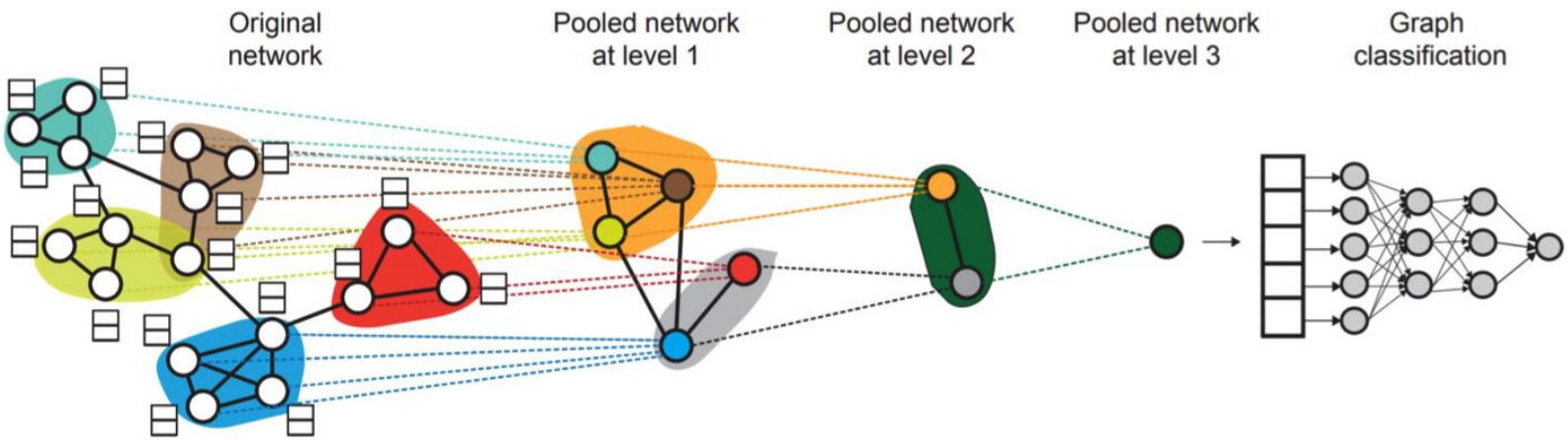
- **Approach 2:** Introduce a **"virtual node"** to represent the (sub)graph and run a standard graph embedding technique



- Proposed by Li et al., 2016 as a general technique for subgraph embedding

Approach 3

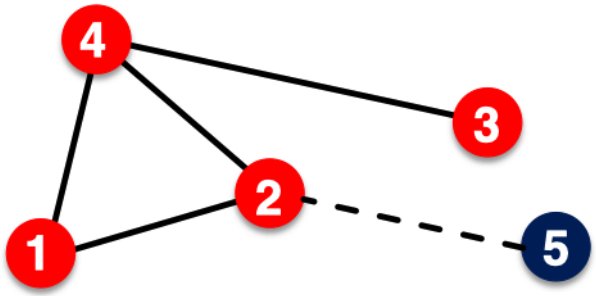
- **DiffPool:** We can also **hierarchically** cluster nodes in graphs, and **sum/avg** the node embeddings according to these clusters.



Limitation 1

Limitations of node embeddings via random walks

- **Transductive (not inductive) method:** Cannot obtain embeddings for nodes not in the training set. Cannot apply to new graphs, evolving graphs.



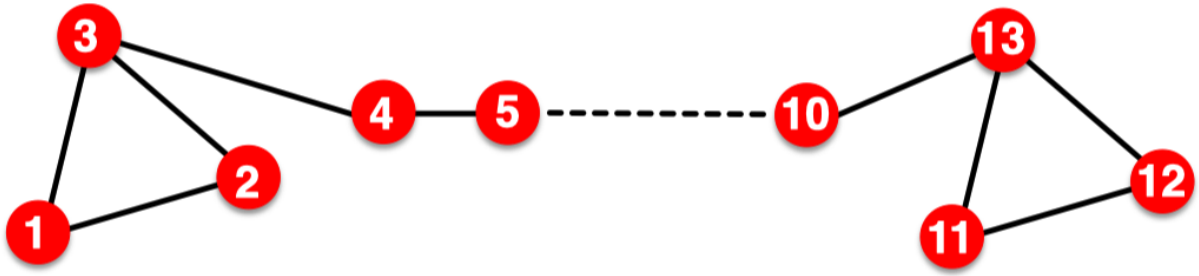
Training set

A newly added node 5 at test time (e.g., new user in a social network)

Cannot compute its embedding with DeepWalk. Need to recompute all node embeddings.

Limitation 2

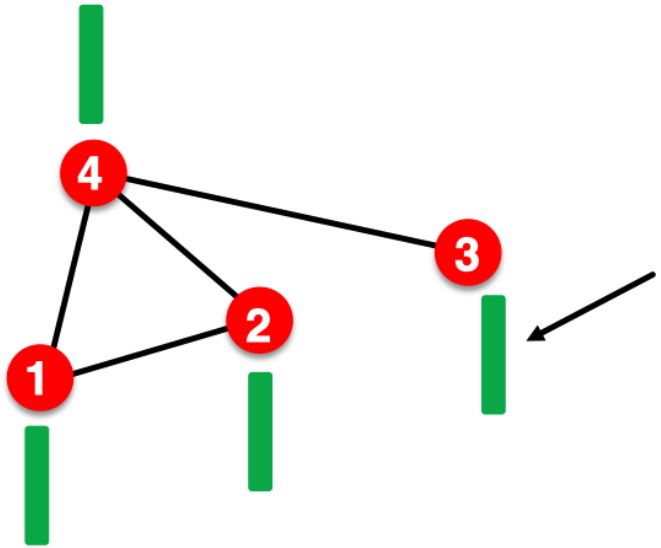
- Cannot capture **structural similarity**:



  - Node 1 and 11 are **structurally similar** – part of one triangle, degree 2, …
  - However, they have very **different** embeddings.
    - It's unlikely that a random walk will reach node 11 from node 1.

Limitation 3

- Cannot utilize node, edge and graph features



Feature vector
(e.g. protein properties in a protein-protein interaction graph)

Solution to these limitations: Deep Representation Learning and Graph Neural Networks

Questions?

UNIVERSITY OF NORTH CAROLINA CHARLOTTE