

# DSC250: Advanced Data Mining

Text Embedding  
Graph Mining

**Zhiting Hu**

Lecture 11, November 1st, 2023

**UC San Diego**

**HALICIOĞLU DATA SCIENCE INSTITUTE**

# Outline

- Recurrent Networks (RNNs)
  - Long-range dependency, vanishing gradients
  - LSTM
  - RNNs in different forms
- Attention Mechanisms
  - (Query, Key, Value)
  - Attention on Text and Images
- Transformers: Multi-head Attention

# Text Embedding

# Word Embedding

- A pre-trained matrix, each row is an embedding vector of a word

	0	1	2	3	4	5	6	7	8	9	..
<b>fox</b>	-0.348680	-0.077720	0.177750	-0.094953	-0.452890	0.237790	0.209440	0.037886	0.035064	0.899010	..
<b>ham</b>	-0.773320	-0.282540	0.580760	0.841480	0.258540	0.585210	-0.021890	-0.463680	0.139070	0.658720	..
<b>brown</b>	-0.374120	-0.076264	0.109260	0.186620	0.029943	0.182700	-0.631980	0.133060	-0.128980	0.603430	..
<b>beautiful</b>	0.171200	0.534390	-0.348540	-0.097234	0.101800	-0.170860	0.295650	-0.041816	-0.516550	2.117200	..
<b>jumps</b>	-0.334840	0.215990	-0.350440	-0.260020	0.411070	0.154010	-0.386110	0.206380	0.386700	1.460500	..
<b>eggs</b>	-0.417810	-0.035192	-0.126150	-0.215930	-0.669740	0.513250	-0.797090	-0.068611	0.634660	1.256300	..
<b>beans</b>	-0.423290	-0.264500	0.200870	0.082187	0.066944	1.027600	-0.989140	-0.259950	0.145960	0.766450	..
<b>sky</b>	0.312550	-0.303080	0.019587	-0.354940	0.100180	-0.141530	-0.514270	0.886110	-0.530540	1.556600	..
<b>bacon</b>	-0.430730	-0.016025	0.484620	0.101390	-0.299200	0.761820	-0.353130	-0.325290	0.156730	0.873210	..
<b>breakfast</b>	0.073378	0.227670	0.208420	-0.456790	-0.078219	0.601960	-0.024494	-0.467980	0.054627	2.283700	..
<b>toast</b>	0.130740	-0.193730	0.253270	0.090102	-0.272580	-0.030571	0.096945	-0.115060	0.484000	0.848380	..
<b>today</b>	-0.156570	0.594890	-0.031445	-0.077586	0.278630	-0.509210	-0.066350	-0.081890	-0.047986	2.803600	..
<b>blue</b>	0.129450	0.036518	0.032298	-0.060034	0.399840	-0.103020	-0.507880	0.076630	-0.422920	0.815730	..
<b>green</b>	-0.072368	0.233200	0.137260	-0.156630	0.248440	0.349870	-0.241700	-0.091426	-0.530150	1.341300	..
<b>kings</b>	0.259230	-0.854690	0.360010	-0.642000	0.568530	-0.321420	0.173250	0.133030	-0.089720	1.528600	..
<b>dog</b>	-0.057120	0.052685	0.003026	-0.048517	0.007043	0.041856	-0.024704	-0.039783	0.009614	0.308416	..
<b>sausages</b>	-0.174290	-0.064869	-0.046976	0.287420	-0.128150	0.647630	0.056315	-0.240440	-0.025094	0.502220	..
<b>lazy</b>	-0.353320	-0.299710	-0.176230	-0.321940	-0.385640	0.586110	0.411160	-0.418680	0.073093	1.486500	..
<b>love</b>	0.139490	0.534530	-0.252470	-0.125650	0.048748	0.152440	0.199060	-0.065970	0.128830	2.055900	..
<b>quick</b>	-0.445630	0.191510	-0.249210	0.465900	0.161950	0.212780	-0.046480	0.021170	0.417660	1.686900	..

# Word Embedding

- Problem: word embeddings are applied in a context free manner

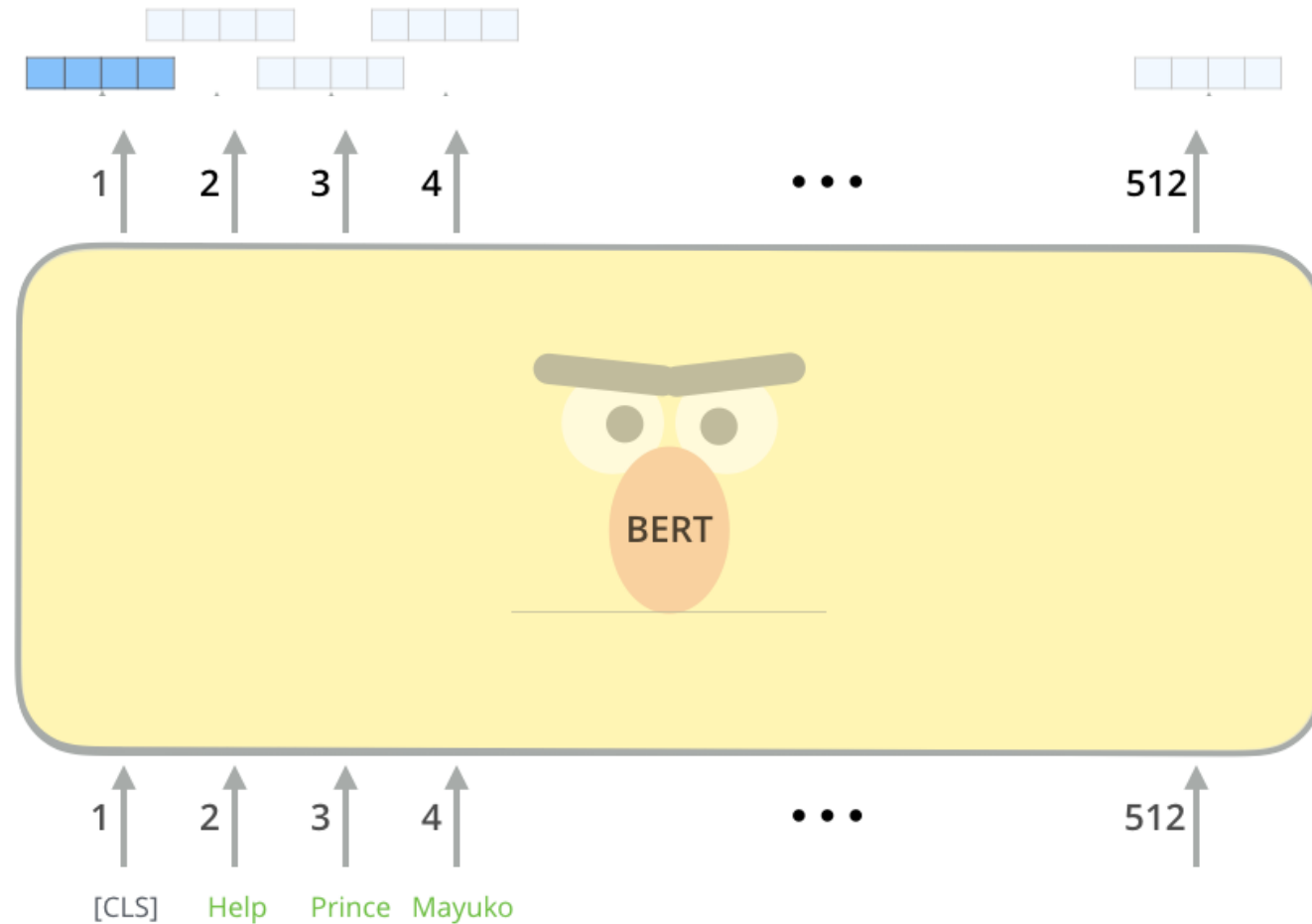
open a bank account                      on the river bank

[0.3, 0.2, -0.8, ...]



# BERT

- BERT: A bidirectional model to extract contextual word embedding



# BERT: Pre-training Procedure

- Dataset:
  - Wikipedia (2.5B words) + a collection of free ebooks (800M words)



# BERT: Pre-training Procedure

- Dataset:
  - Wikipedia (2.5B words) + a collection of free ebooks (800M words)
- Training procedure
  - **masked language model** (masked LM)
    - Masks some percent of words from the input and has to reconstruct those words from context

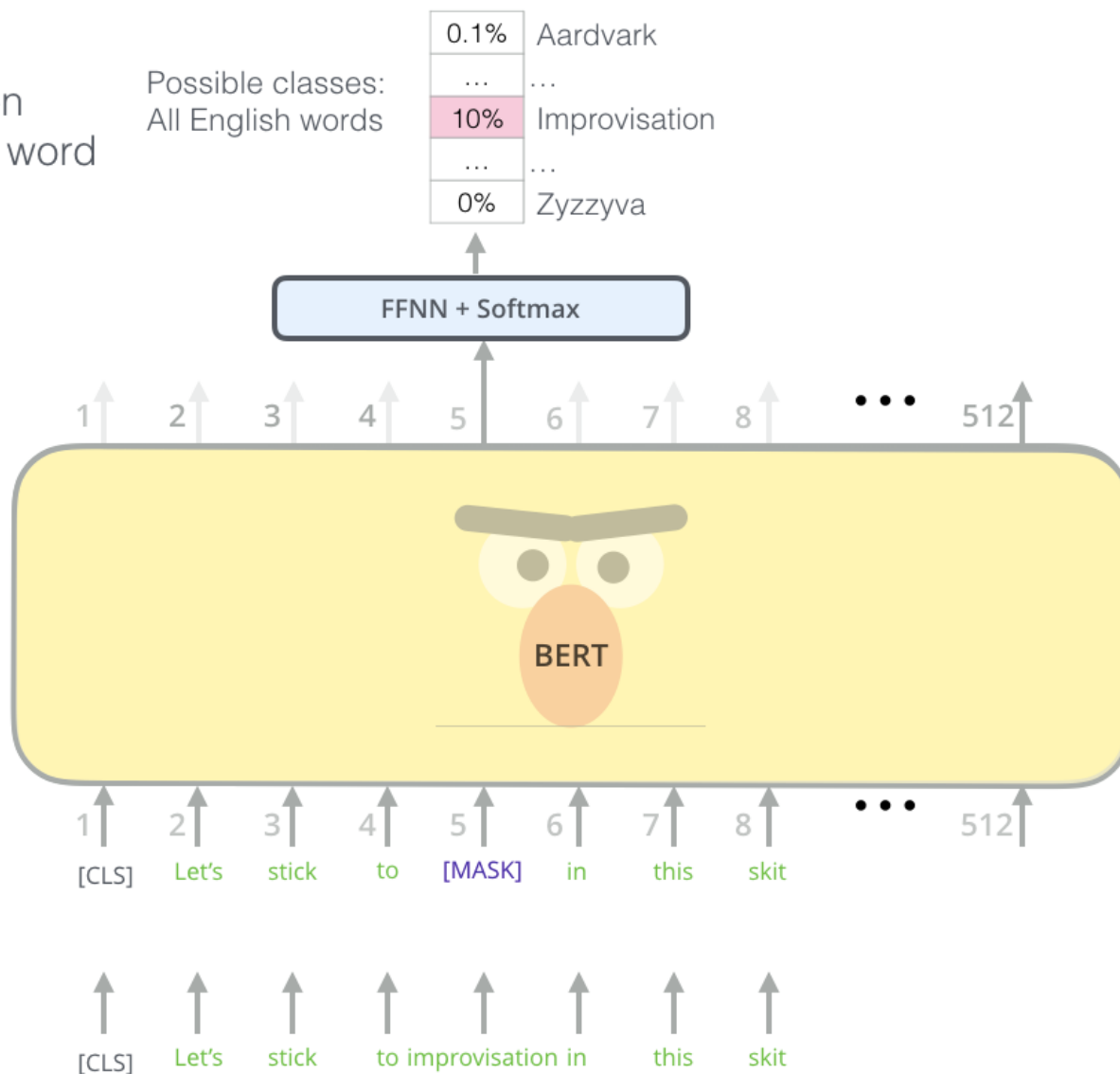
# BERT: Pre-training Procedure

- Masked LM

Use the output of the masked word's position to predict the masked word

Randomly mask 15% of tokens

Input



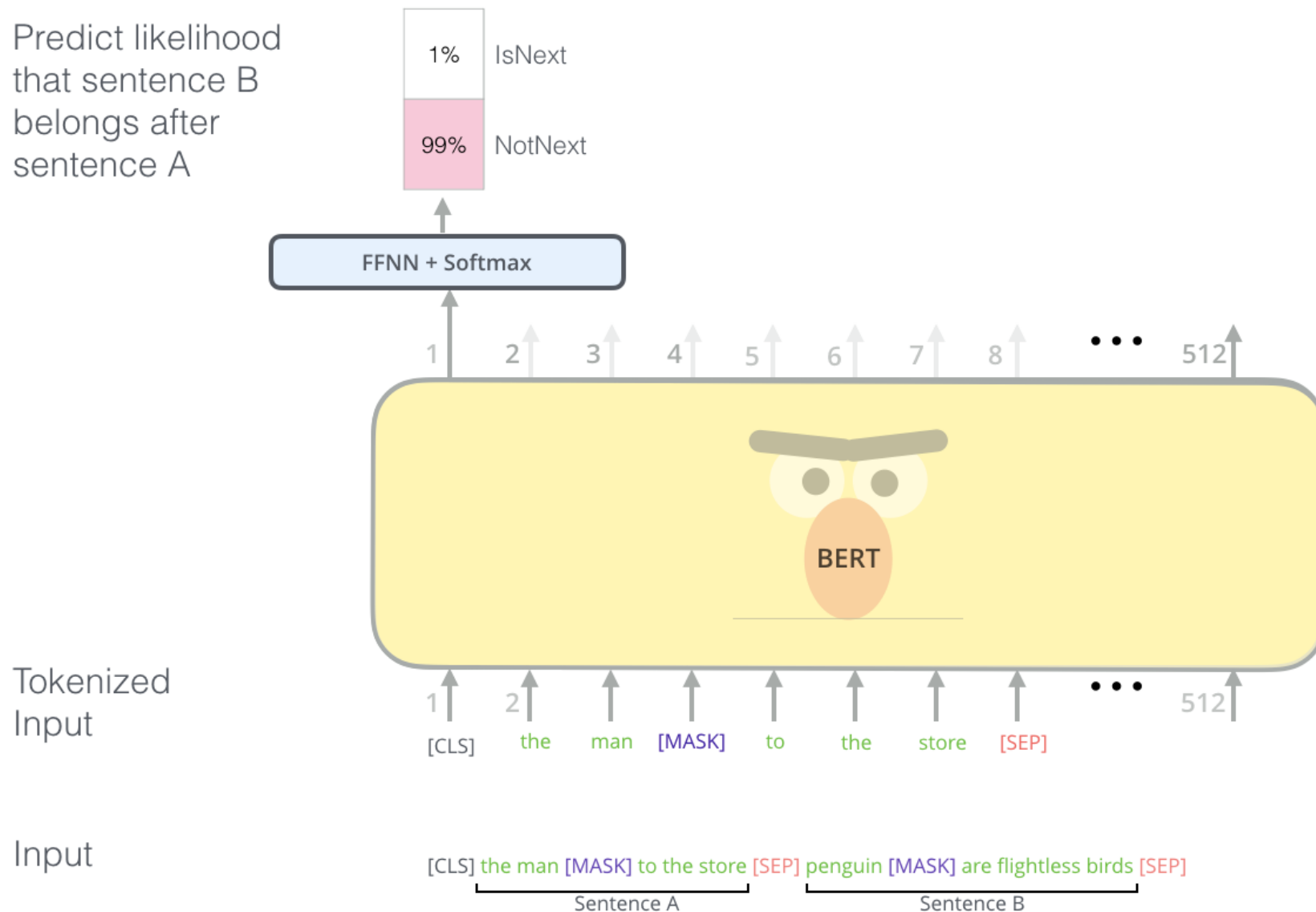
# BERT: Pre-training Procedure

- Dataset:
  - Wikipedia (2.5B words) + a collection of free ebooks (800M words)
- Training procedure
  - **masked language model** (masked LM)
    - Masks some percent of words from the input and has to reconstruct those words from context
  - **Two-sentence task**
    - To understand relationships between sentences
    - Concatenate two sentences A and B and predict whether B actually comes after A in the original text

# BERT: Pre-training Procedure

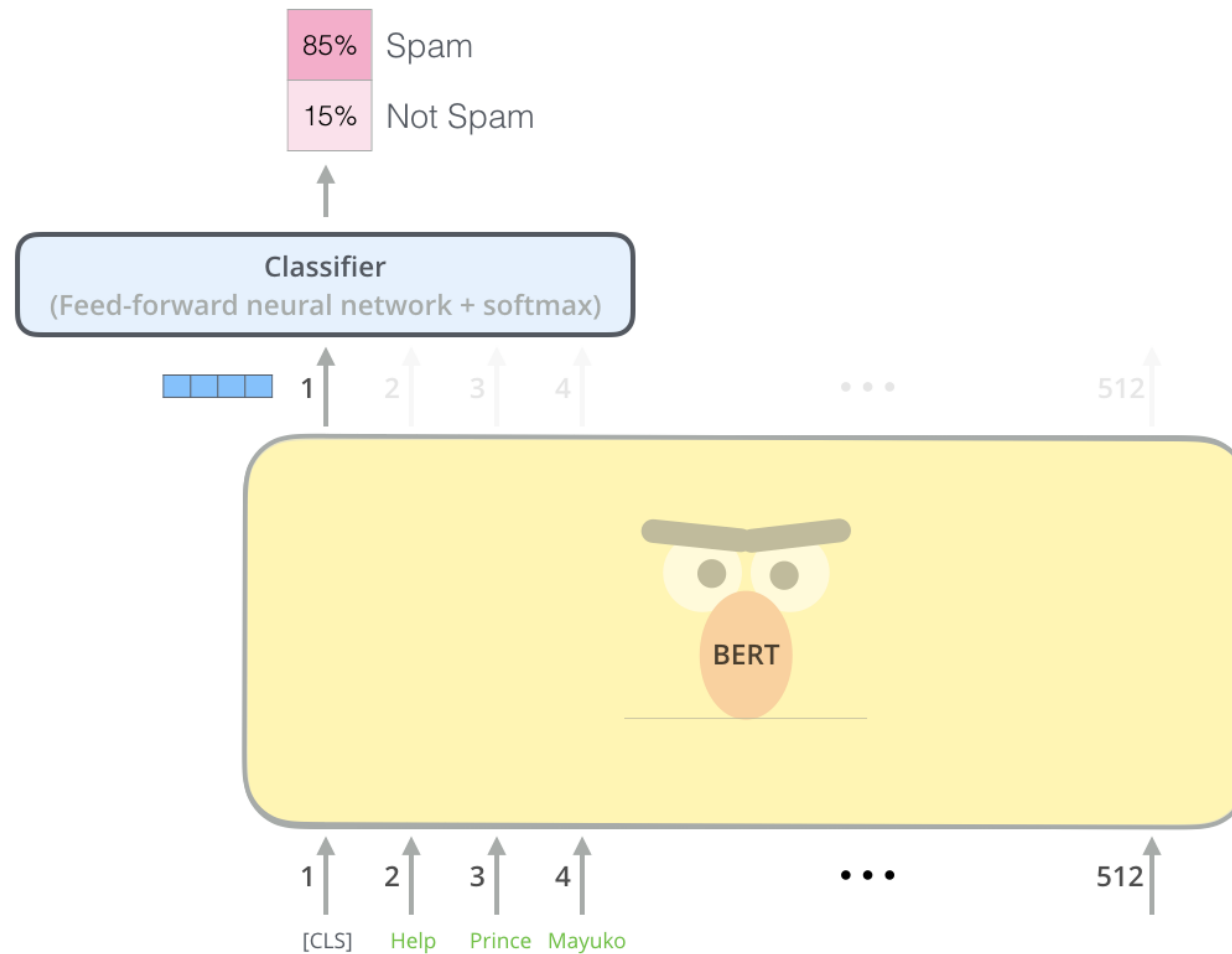
- Two sentence task

Predict likelihood that sentence B belongs after sentence A

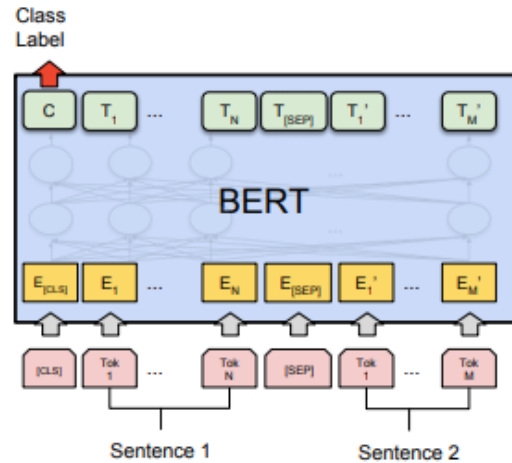


# BERT: Downstream Fine-tuning

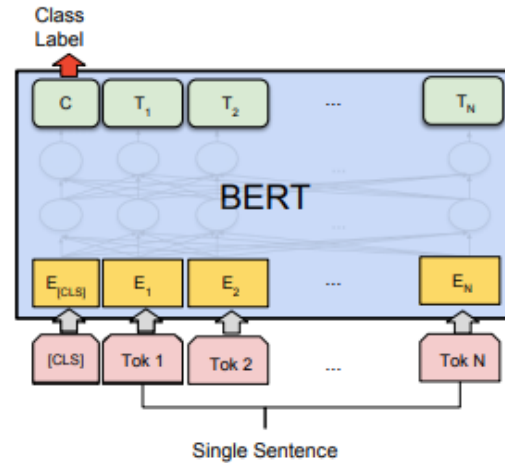
- Use BERT for sentence classification



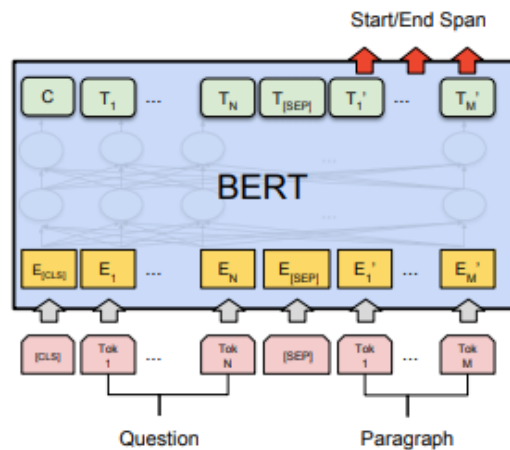
# BERT: Downstream Fine-tuning



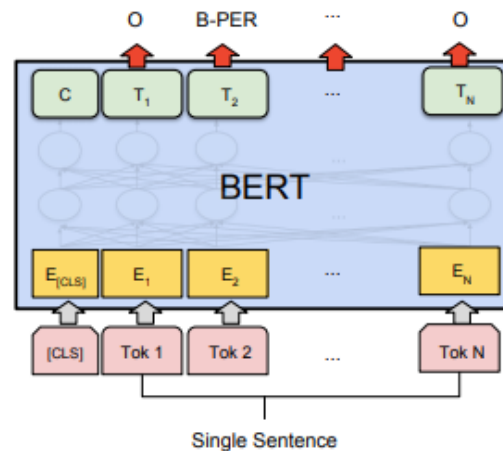
(a) Sentence Pair Classification Tasks:  
MNLI, QQP, QNLI, STS-B, MRPC,  
RTE, SWAG



(b) Single Sentence Classification Tasks:  
SST-2, CoLA



(c) Question Answering Tasks:  
SQuAD v1.1



(d) Single Sentence Tagging Tasks:  
CoNLL-2003 NER

# BERT Results

- Huge improvements over SOTA on 12 NLP task

System	MNLI-(m/mm) 392k	QQP 363k	QNLI 108k	SST-2 67k	CoLA 8.5k	STS-B 5.7k	MRPC 3.5k	RTE 2.5k	Average
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.9	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	88.1	91.3	45.4	80.0	82.3	56.0	75.2
BERT <sub>BASE</sub>	84.6/83.4	71.2	90.1	93.5	52.1	85.8	88.9	66.4	79.6
BERT <sub>LARGE</sub>	<b>86.7/85.9</b>	<b>72.1</b>	<b>91.1</b>	<b>94.9</b>	<b>60.5</b>	<b>86.5</b>	<b>89.3</b>	<b>70.1</b>	<b>81.9</b>

Table 1: GLUE Test results, scored by the GLUE evaluation server. The number below each task denotes the number of training examples. The “Average” column is slightly different than the official GLUE score, since we exclude the problematic WNLI set. OpenAI GPT = (L=12, H=768, A=12); BERT<sub>BASE</sub> = (L=12, H=768, A=12); BERT<sub>LARGE</sub> = (L=24, H=1024, A=16). BERT and OpenAI GPT are single-model, single task. All results obtained from <https://gluebenchmark.com/leaderboard> and <https://blog.openai.com/language-unsupervised/>.

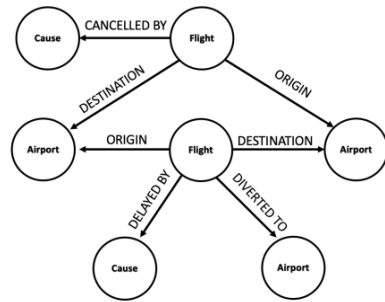
# Graph Mining

Slides adapted from:

- Jure Leskovec, Stanford CS224W: Machine Learning with Graphs



# Graph is everywhere

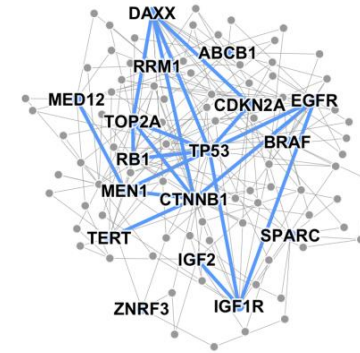


**Event Graphs**



Image credit: [SalientNetworks](#)

**Computer Networks**



**Disease Pathways**

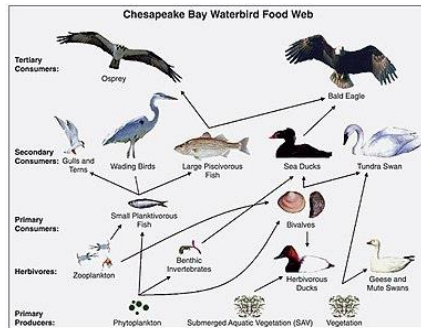


Image credit: [Wikipedia](#)

**Food Webs**



Image credit: [Pinterest](#)

**Particle Networks**

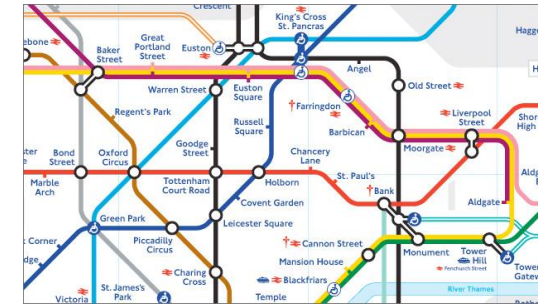


Image credit: [visitlondon.com](#)

**Underground Networks**

# Graph is everywhere



Image credit: [Medium](#)

## Social Networks

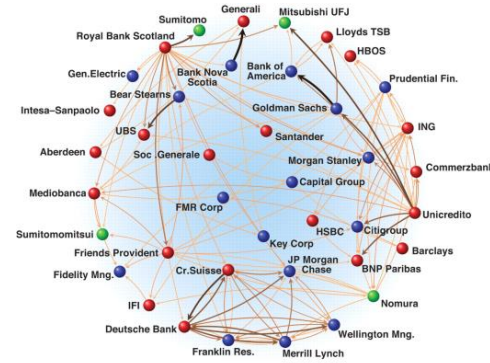


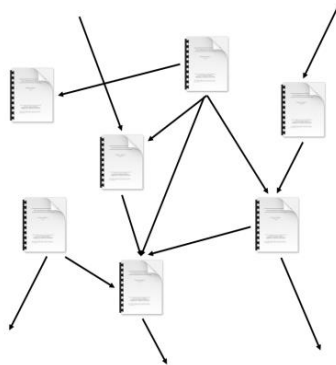
Image credit: [Science](#)

## Economic Networks



Image credit: [Lumen Learning](#)

## Communication Networks



## Citation Networks



Image credit: [Missoula Current News](#)

## Internet

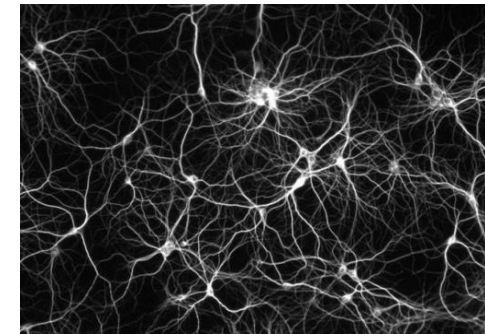


Image credit: [The Conversation](#)

## Networks of Neurons

# Graph is everywhere

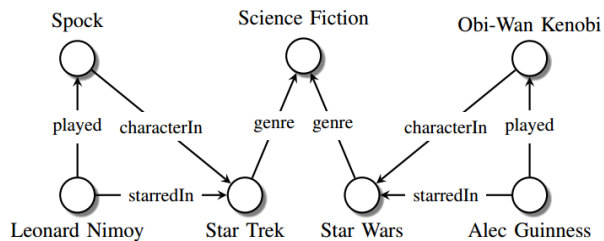


Image credit: [Maximilian Nickel et al](#)

## Knowledge Graphs

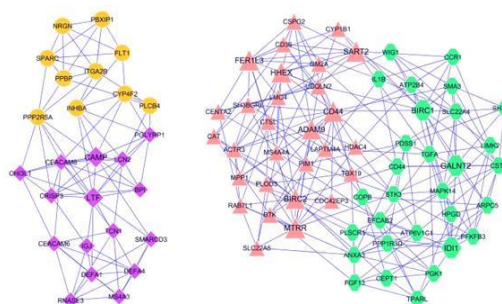


Image credit: [ese.wustl.edu](#)

## Regulatory Networks

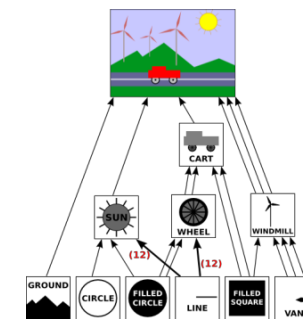


Image credit: [math.hws.edu](#)

## Scene Graphs

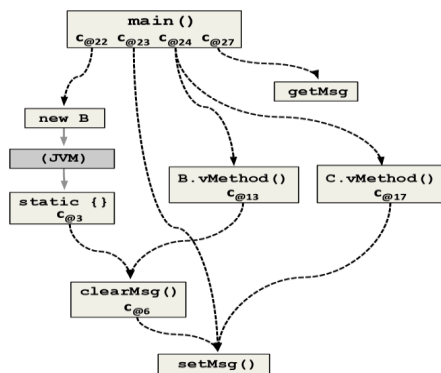


Image credit: [ResearchGate](#)

## Code Graphs

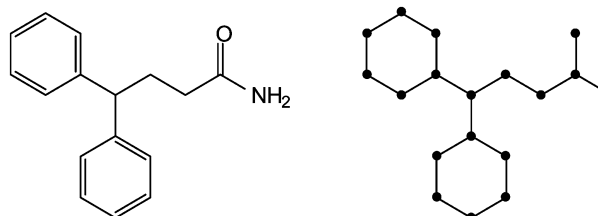


Image credit: [MDPI](#)

## Molecules

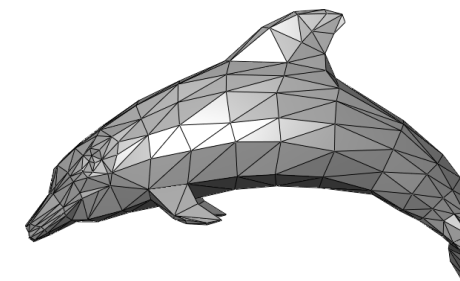
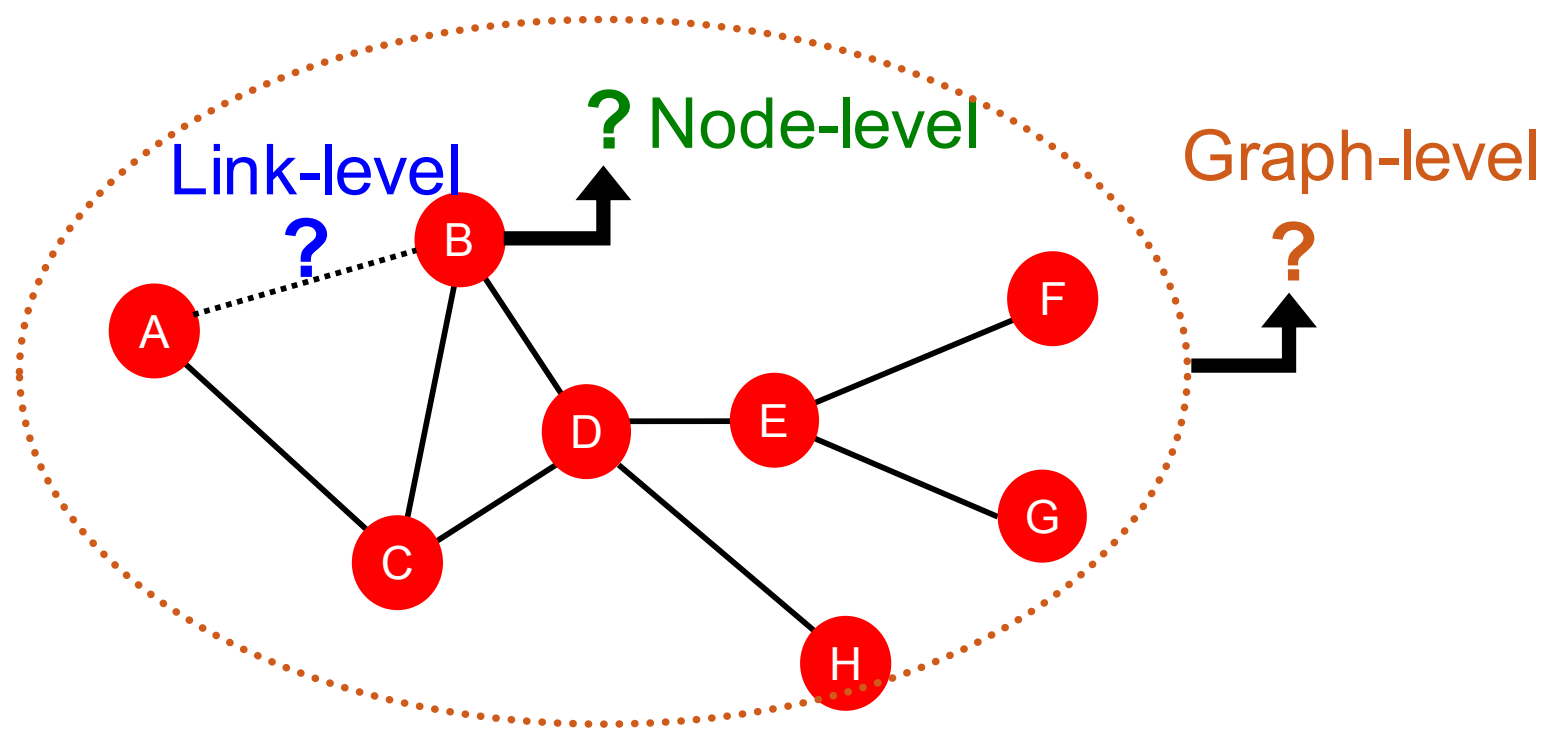


Image credit: [Wikipedia](#)

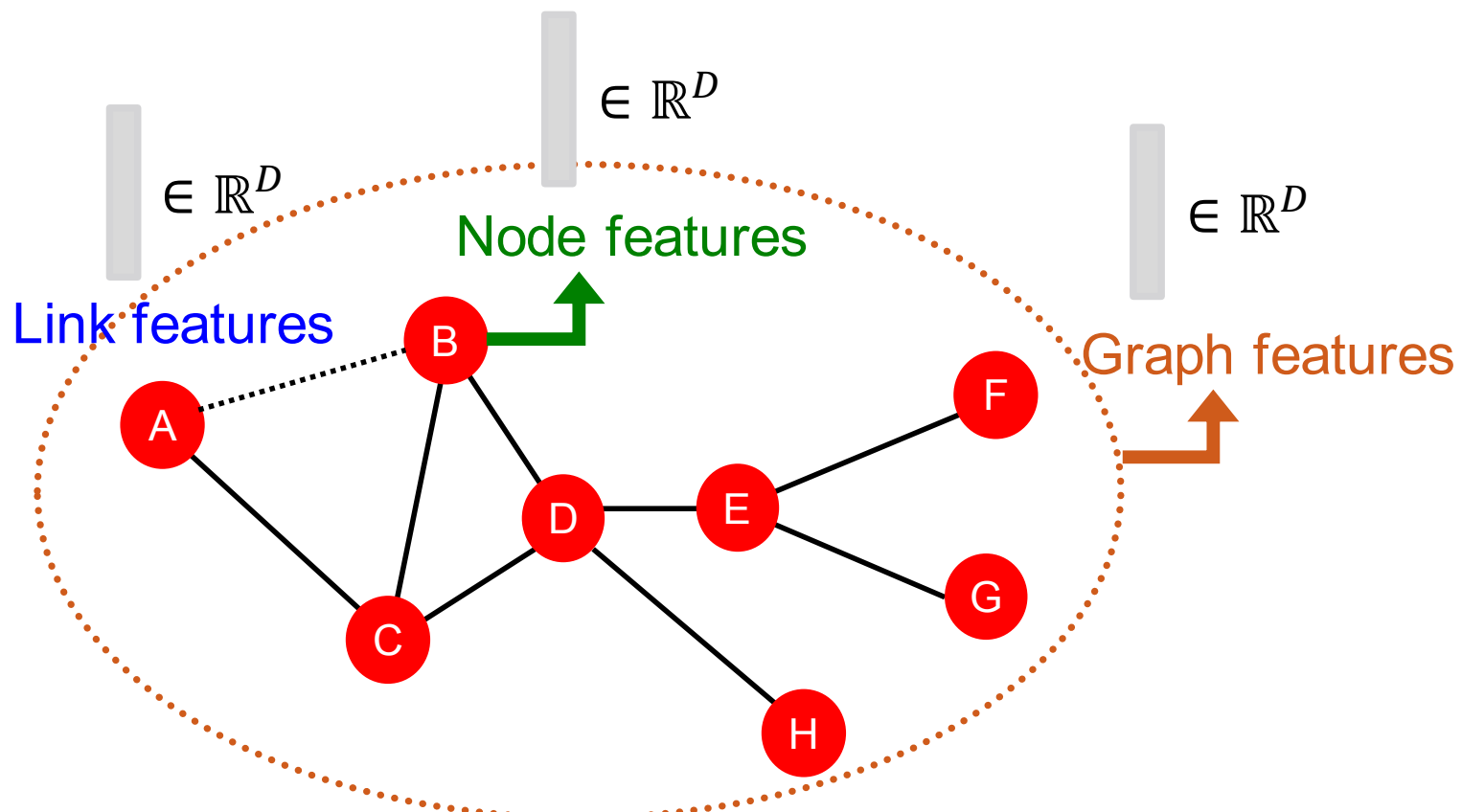
## 3D Shapes

# Tasks on Graph

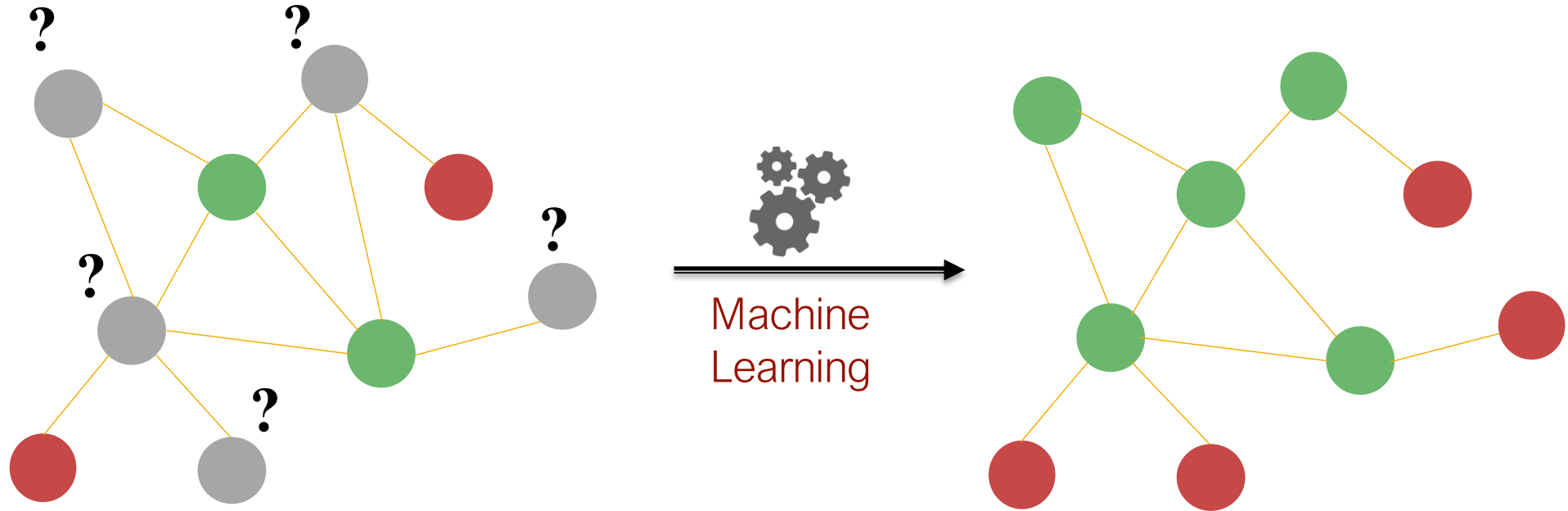
- Node-level prediction
- Link-level prediction
- Graph-level prediction



# Getting Features for Nodes/Links/Graphs



# Node-level Tasks

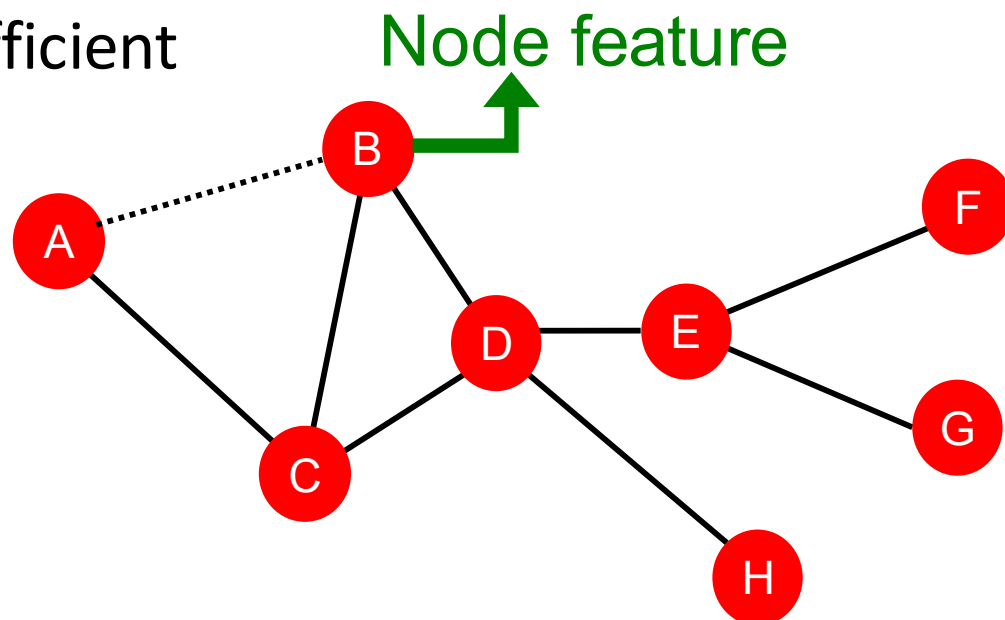


Node classification

# Node-level Features

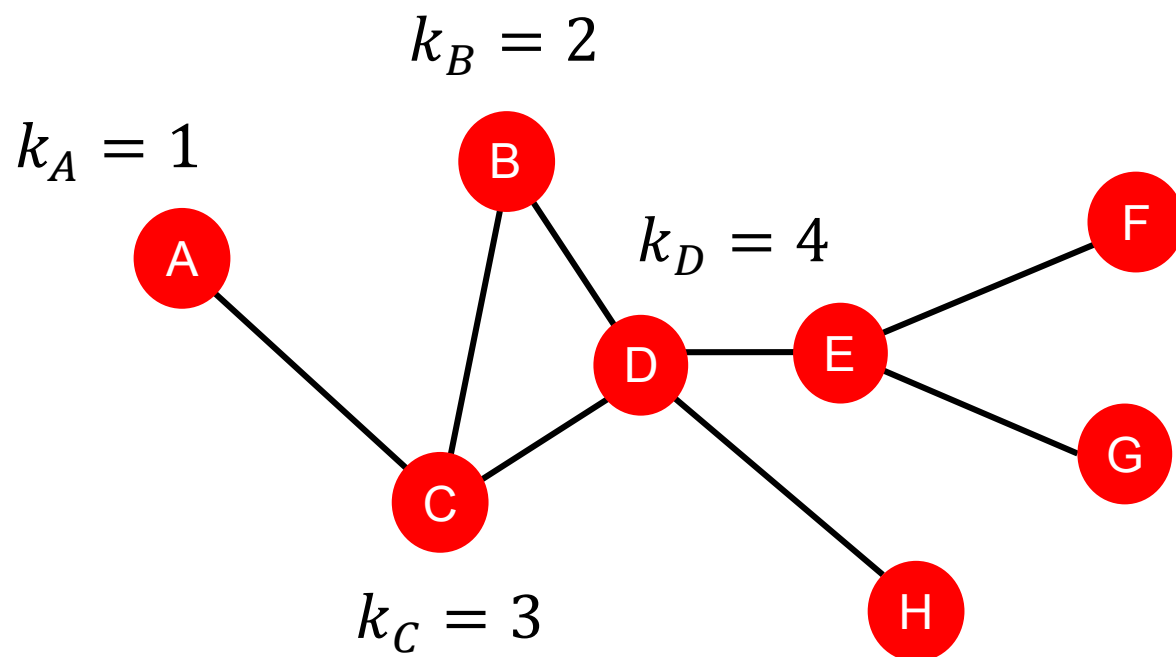
**Goal:** Characterize the structure and position of a node in the network:

- Node degree
- Node centrality
- Clustering coefficient
- Graphlets



# Node-level Features (1): Node Degree

- The degree  $k_v$  of node  $v$  is the number of edges (neighboring nodes) the node has.
- Treats all neighboring nodes equally.





## Node-level Features (2): Node Centrality

- Node degree counts the neighboring nodes **without capturing their importance.**
- **Node centrality**  $c_v$  takes the **node importance in a graph** into account
- **Different ways to model importance:**
  - Eigenvector centrality
  - Betweenness centrality
  - Closeness centrality
  - and many others...

## Node-level Features (2): Node Centrality

- **Eigenvector centrality:**

- A node  $v$  is important if **surrounded by important neighboring nodes**  $u \in N(v)$ .
- We model the centrality of node  $v$  as **the sum of the centrality of neighboring nodes:**

$$c_v = \frac{1}{\lambda} \sum_{u \in N(v)} c_u$$

$\lambda$  is normalization constant (it will turn out to be the largest eigenvalue of A)

## Node-level Features (2): Node Centrality

- **Eigenvector centrality:**

- A node  $v$  is important if **surrounded by important neighboring nodes**  $u \in N(v)$ .
- We model the centrality of node  $v$  as **the sum of the centrality of neighboring nodes:**

$$c_v = \frac{1}{\lambda} \sum_{u \in N(v)} c_u$$

$\lambda$  is normalization constant (it will turn out to be the largest eigenvalue of A)

- Notice that the above equation models centrality in a **recursive manner**. **How do we solve it?**

## Node-level Features (2): Node Centrality

### ■ Eigenvector centrality:

- Rewrite the recursive equation in the matrix form.

$$c_v = \frac{1}{\lambda} \sum_{u \in N(v)} c_u \quad \longleftrightarrow \quad \lambda \mathbf{c} = \mathbf{A} \mathbf{c}$$

$\lambda$  is normalization const  
(largest eigenvalue of  $A$ )

- $A$ : Adjacency matrix  
 $A_{uv} = 1$  if  $u \in N(v)$
- $\mathbf{c}$ : Centrality vector
- $\lambda$ : Eigenvalue

- We see that centrality  $\mathbf{c}$  is the **eigenvector of  $A$ !**
- The largest eigenvalue  $\lambda_{max}$  is always positive and unique (by Perron-Frobenius Theorem).
- The eigenvector  $\mathbf{c}_{max}$  corresponding to  $\lambda_{max}$  is used for centrality.

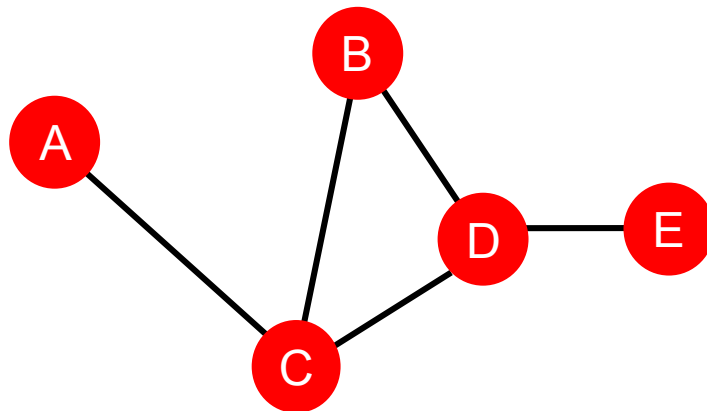
## Node-level Features (2): Node Centrality

### ■ Betweenness centrality:

- A node is important if it lies on many shortest paths between other nodes.

$$c_v = \sum_{s \neq v \neq t} \frac{\#(\text{shortest paths between } s \text{ and } t \text{ that contain } v)}{\#(\text{shortest paths between } s \text{ and } t)}$$

### ■ Example:



$$c_A = c_B = c_E = 0$$
$$c_C = 3$$

(A-C-B, A-C-D, A-C-D-E)

$$c_D = 3$$

(A-C-D-E, B-D-E, C-D-E)

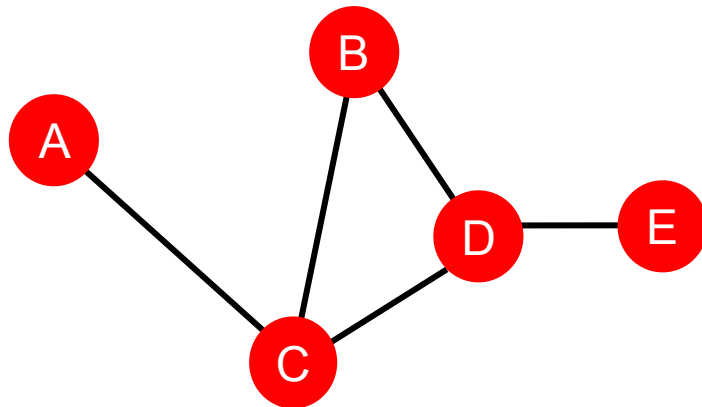
## Node-level Features (2): Node Centrality

- **Closeness centrality:**

- A node is important if it has small shortest path lengths to all other nodes.

$$c_v = \frac{1}{\sum_{u \neq v} \text{shortest path length between } u \text{ and } v}$$

- **Example:**



$$c_A = 1/(2 + 1 + 2 + 3) = 1/8$$

(A-C-B, A-C, A-C-D, A-C-D-E)

$$c_D = 1/(2 + 1 + 1 + 1) = 1/5$$

(D-C-A, D-B, D-C, D-E)

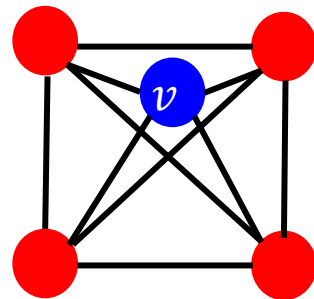
# Node-level Features (3): Clustering Coefficient

- Measures how connected  $v$ 's neighboring nodes are:

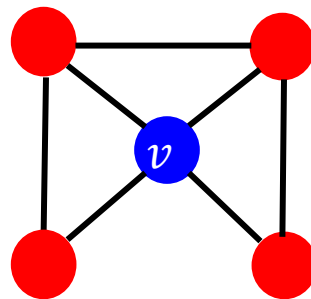
$$e_v = \frac{\#(\text{edges among neighboring nodes})}{\binom{k_v}{2}} \in [0,1]$$

#(node pairs among  $k_v$  neighboring nodes)  
In our examples below the denominator is 6 (4 choose 2).

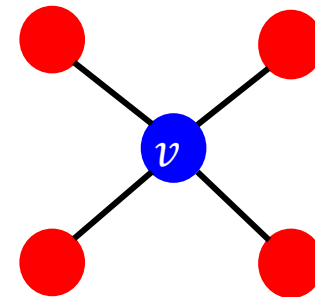
- **Examples:**



$$e_v = 1$$



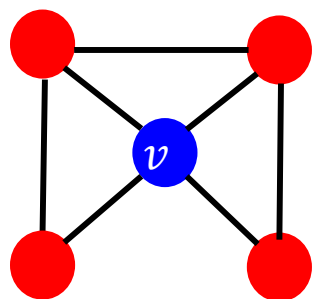
$$e_v = 0.5$$



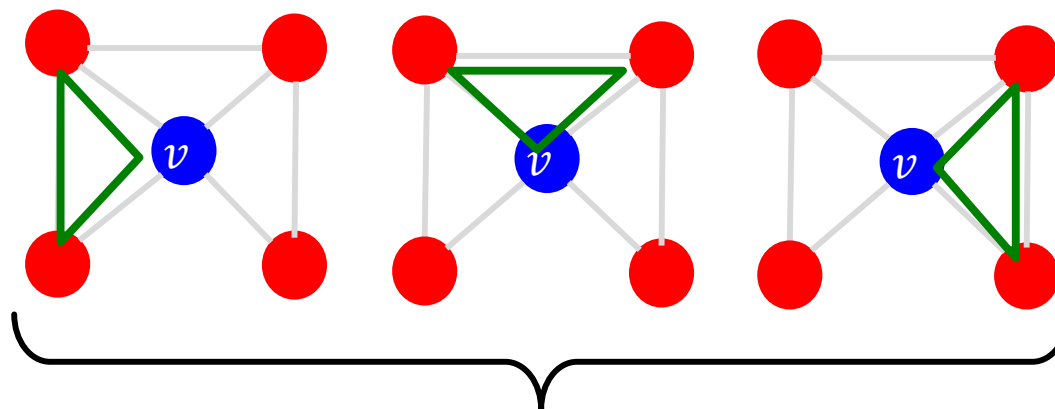
$$e_v = 0$$

## Node-level Features (4): Graphlets

- **Observation:** Clustering coefficient counts the #(triangles) in the **ego-network**



$$e_v = 0.5$$



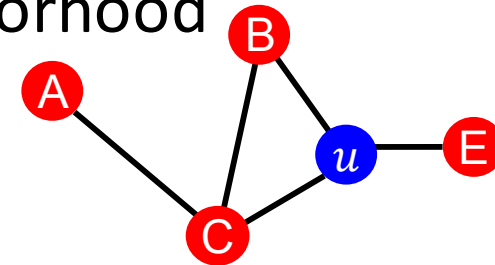
3 triangles (out of 6 node triplets)

- We can generalize the above by counting #(pre-specified subgraphs, i.e., **graphlets**).



# Node-level Features (4): Graphlets

- **Goal:** Describe network structure around node  $u$ 
  - **Graphlets** are small subgraphs that describe the structure of node  $u$ 's network neighborhood



## Analogy:

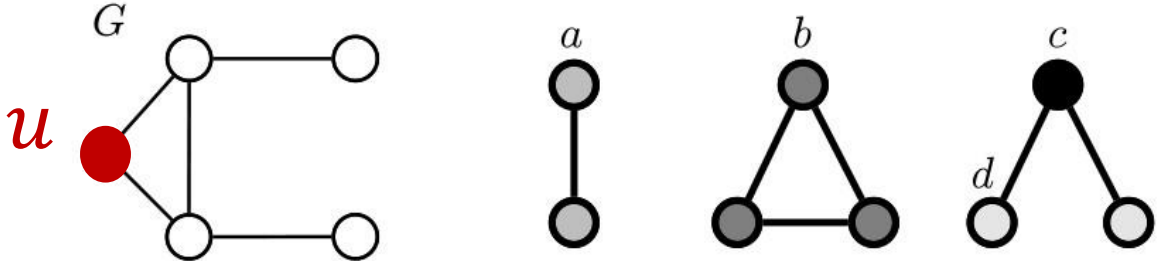
- **Degree** counts **#(edges)** that a node touches
- **Clustering coefficient** counts **#(triangles)** that a node touches.
- **Graphlet Degree Vector (GDV):** Graphlet-base features for nodes
  - **GDV** counts **#(graphlets)** that a node touches

# Node-level Features (4): Graphlets

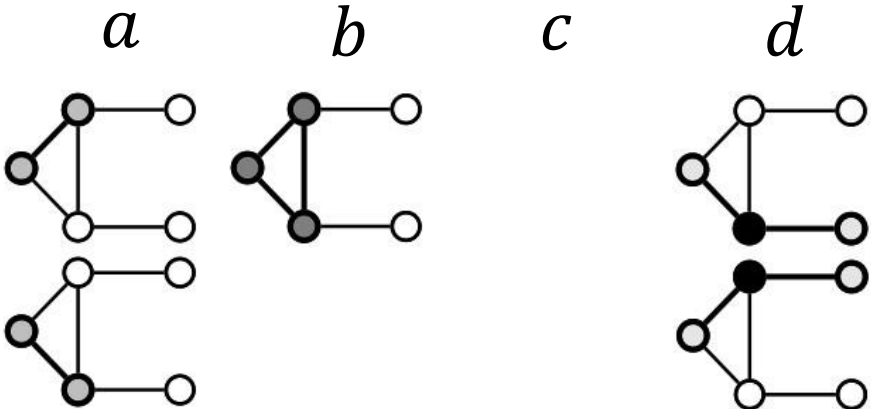
- **Graphlet Degree Vector (GDV):** A count vector of graphlets rooted at a given node.

- **Example:**

Possible graphlets on up to 3 nodes



Graphlet instances of node  $u$ :



GDV of node  $u$ :  
 $a, b, c, d$   
 $[2, 1, 0, 2]$

# Node-level Features: Summary

- **We have introduced different ways to obtain node features.**
- **They can be categorized as:**
  - **Importance-based features:**
    - Node degree
    - Different node centrality measures
  - **Structure-based features:**
    - Node degree
    - Clustering coefficient
    - Graphlet count vector

# Node-level Features: Summary

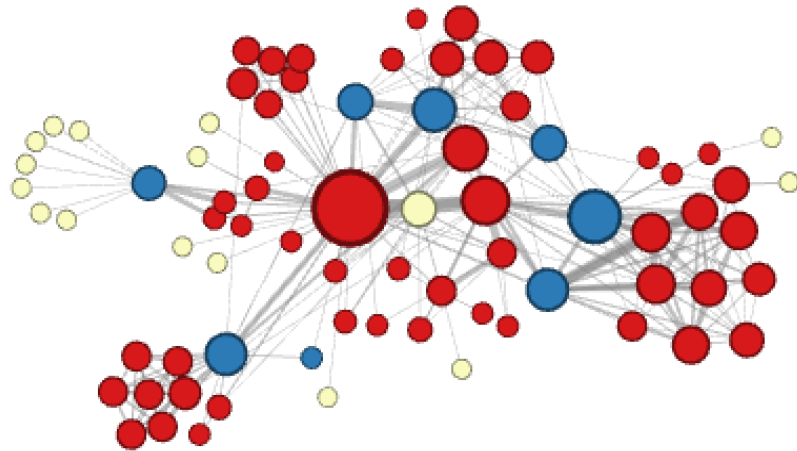
- **Importance-based features:** capture the importance of a node in a graph
  - Node degree:
    - Simply counts the number of neighboring nodes
  - Node centrality:
    - Models **importance of neighboring nodes** in a graph
    - Different modeling choices: eigenvector centrality, betweenness centrality, closeness centrality
- Useful for predicting influential nodes in a graph
  - **Example:** predicting celebrity users in a social network

# Node-level Features: Summary

- **Structure-based features:** Capture topological properties of local neighborhood around a node.
  - **Node degree:**
    - Counts the number of neighboring nodes
  - **Clustering coefficient:**
    - Measures how connected neighboring nodes are
  - **Graphlet degree vector:**
    - Counts the occurrences of different graphlets
- **Useful for predicting a particular role a node plays in a graph:**
  - **Example:** Predicting protein functionality in a protein-protein interaction network.

# Node-level Features: Discussion

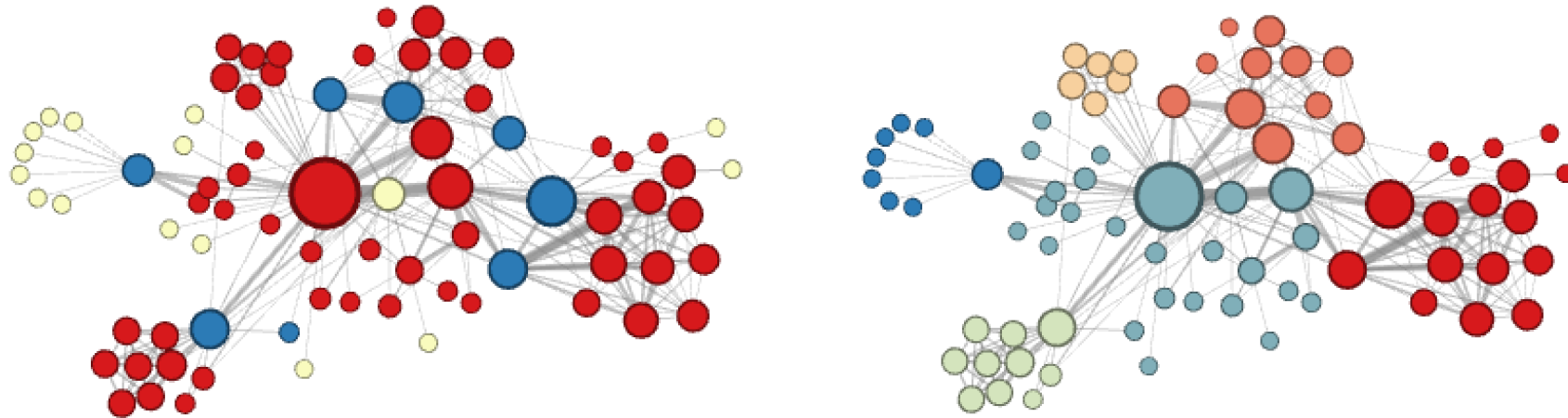
**Different ways to label nodes of the network:**



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

# Node-level Features: Discussion

**Different ways to label nodes of the network:**

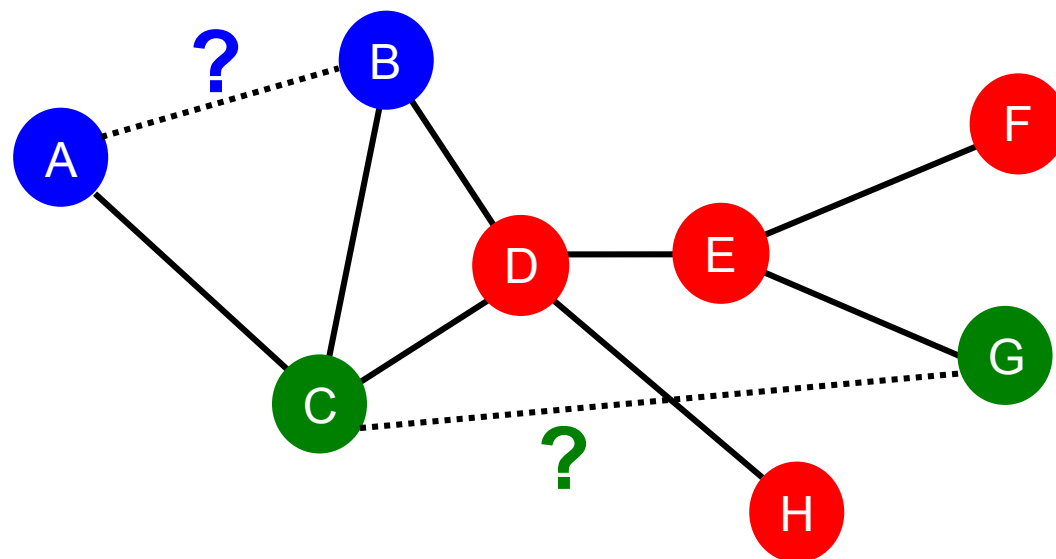


Node features defined so far would allow to distinguish nodes in the above example



# Link-level Task

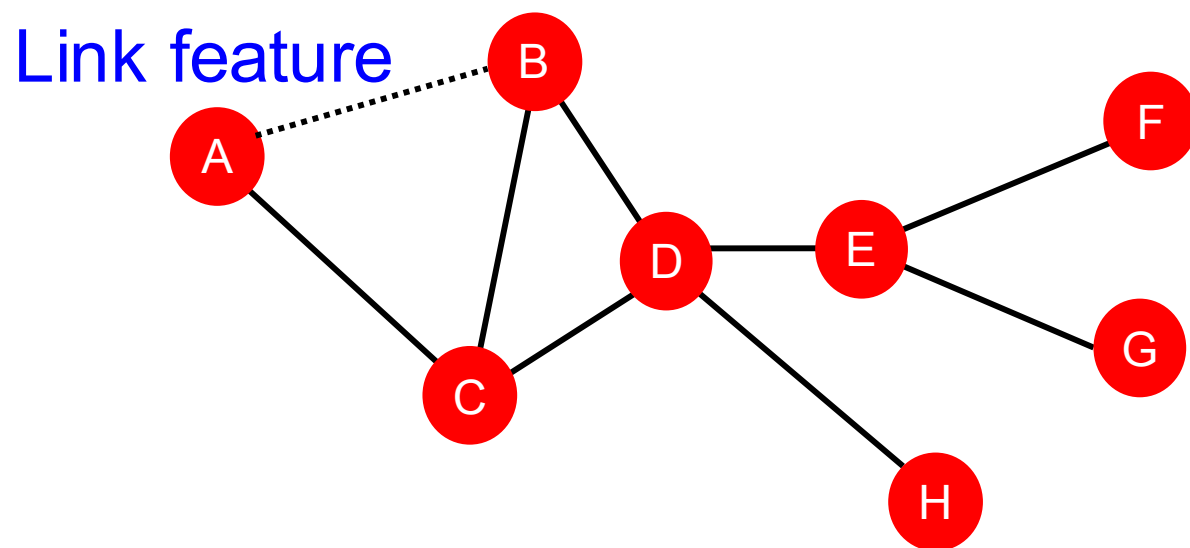
- The task is to predict **new links** based on the existing links.
- At test time, node pairs (with no existing links) are ranked, and top  $K$  node pairs are predicted.
- The key is to design features for a **pair of nodes**.





# Link-level Features: Quick Overview

- Distance-based feature
- Local neighborhood overlap
- Global neighborhood overlap



# Link-level Features: Quick Overview

- **Distance-based features:**
  - Uses the shortest path length between two nodes but does not capture how neighborhood overlaps.

# Link-level Features: Quick Overview

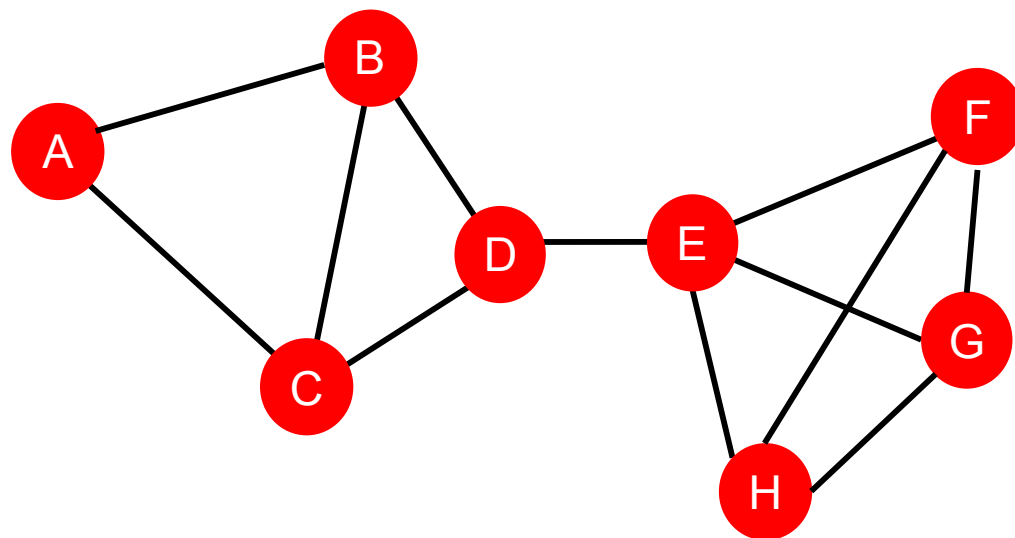
- **Distance-based features:**
  - Uses the shortest path length between two nodes but does not capture how neighborhood overlaps.
- **Local neighborhood overlap:**
  - Captures how many neighboring nodes are shared by two nodes.
  - Becomes zero when no neighbor nodes are shared.

# Link-level Features: Quick Overview

- **Distance-based features:**
  - Uses the shortest path length between two nodes but does not capture how neighborhood overlaps.
- **Local neighborhood overlap:**
  - Captures how many neighboring nodes are shared by two nodes.
  - Becomes zero when no neighbor nodes are shared.
- **Global neighborhood overlap:**
  - Uses global graph structure to score two nodes.
  - Katz index counts #walks of all lengths between two nodes.

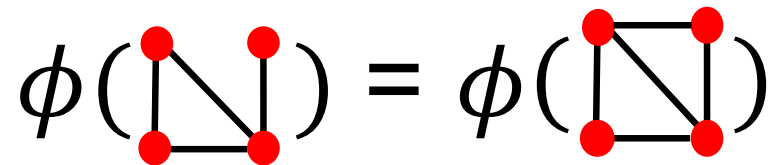
# Graph-level Features

- **Goal:** We want features that characterize the structure of an entire graph.
- **For example:**



# Graph-level Features

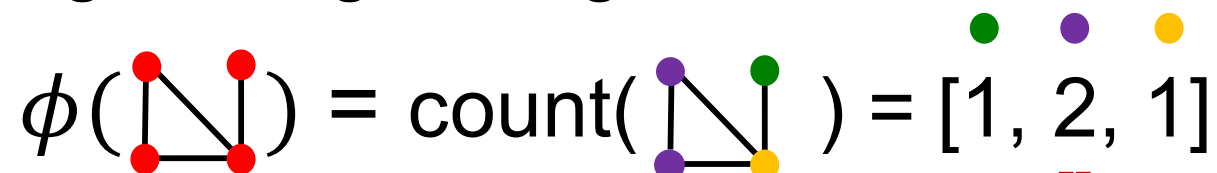
- **Key idea: Bag-of-Words (BoW)** for a graph
  - **Recall:** BoW simply uses the word counts as features for documents (no ordering considered).
  - Naïve extension to a graph: **Regard nodes as words.**
  - Since both graphs have **4 red nodes**, we get the same feature vector for two different graphs...

$$\phi(\text{graph}_1) = \phi(\text{graph}_2)$$


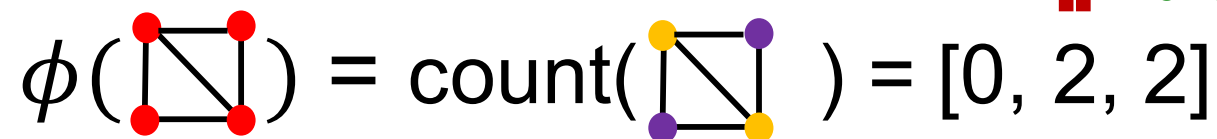
# Graph-level Features

What if we use Bag of node degrees?

Deg1: ● Deg2: ● Deg3: ●

$$\phi(\text{Graph 1}) = \text{count}(\text{Graph 2}) = [1, 2, 1]$$


Obtains different features for different graphs!

$$\phi(\text{Graph 3}) = \text{count}(\text{Graph 4}) = [0, 2, 2]$$


- Both Graphlet Kernel and Weisfeiler-Lehman (WL) Kernel use **Bag-of-\*** representation of graph, where **\*** is more sophisticated than node degrees!

# Graph-level Features: Graphlet Features

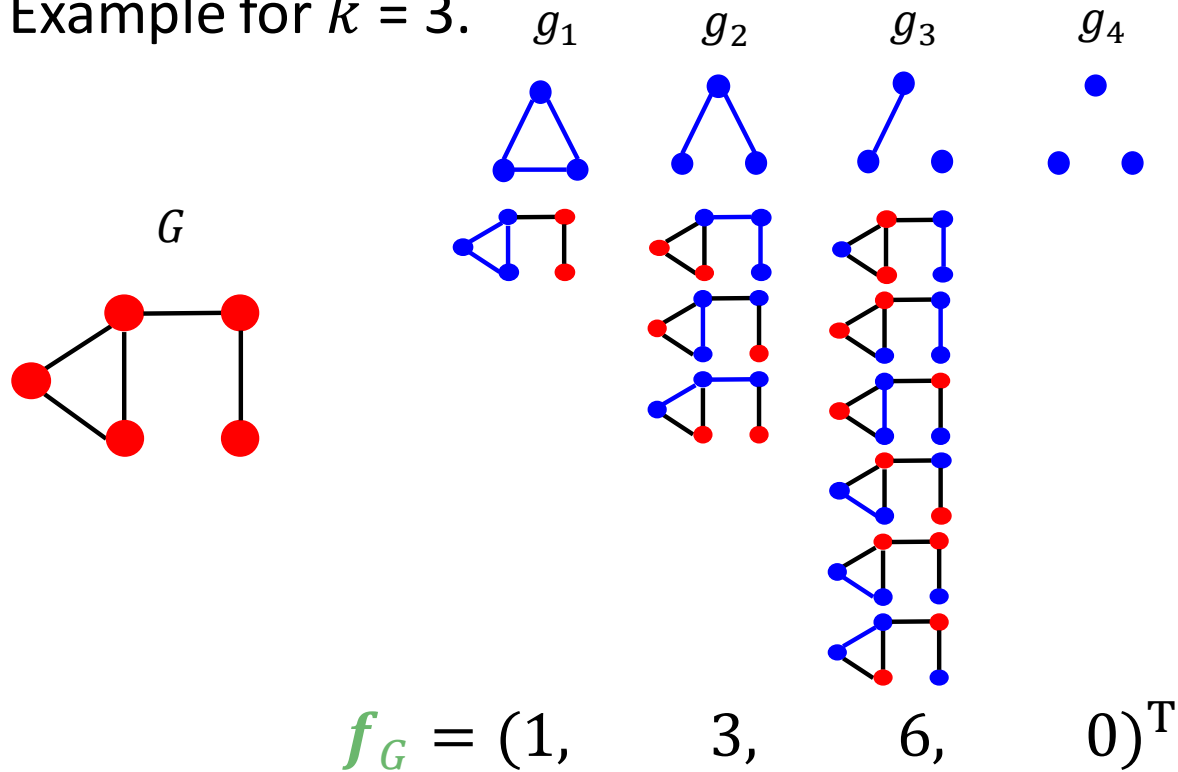
- **Key idea:** Count the number of **different graphlets** in a graph.
- Given graph  $G$ , and a graphlet list  $\mathcal{G}_k = (g_1, g_2, \dots, g_{n_k})$ , define the graphlet count vector  $f_G \in \mathbb{R}^{n_k}$  as

$$(f_G)_i = \#(g_i \subseteq G) \text{ for } i = 1, 2, \dots, n_k.$$



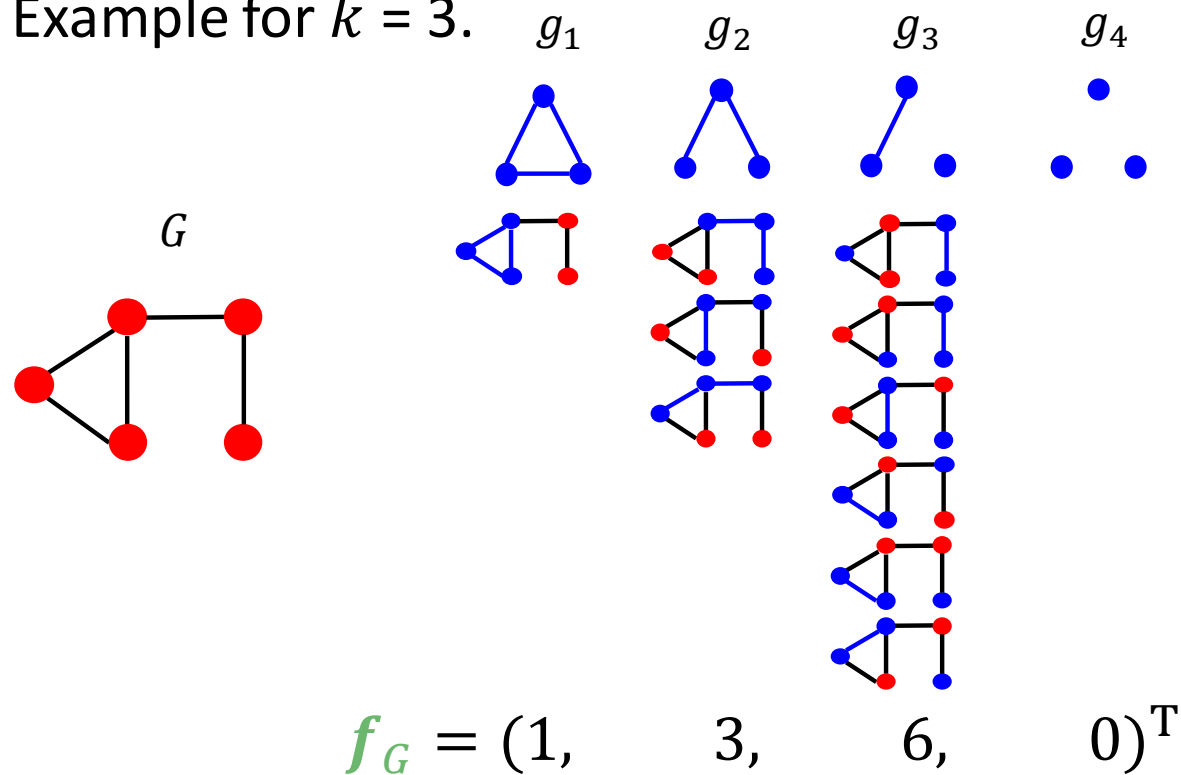
# Graph-level Features: Graphlet Features

- Example for  $k = 3$ .



# Graph-level Features: Graphlet Features

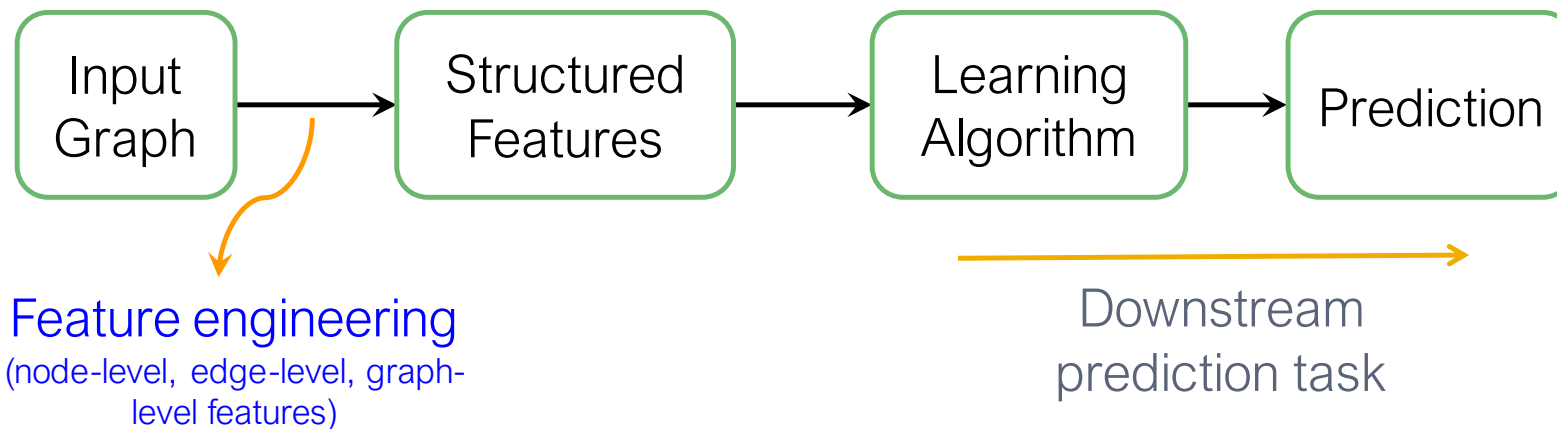
- Example for  $k = 3$ .



- Limitations: Counting graphlets is **expensive!**
- More advanced methods: color refinement, etc.

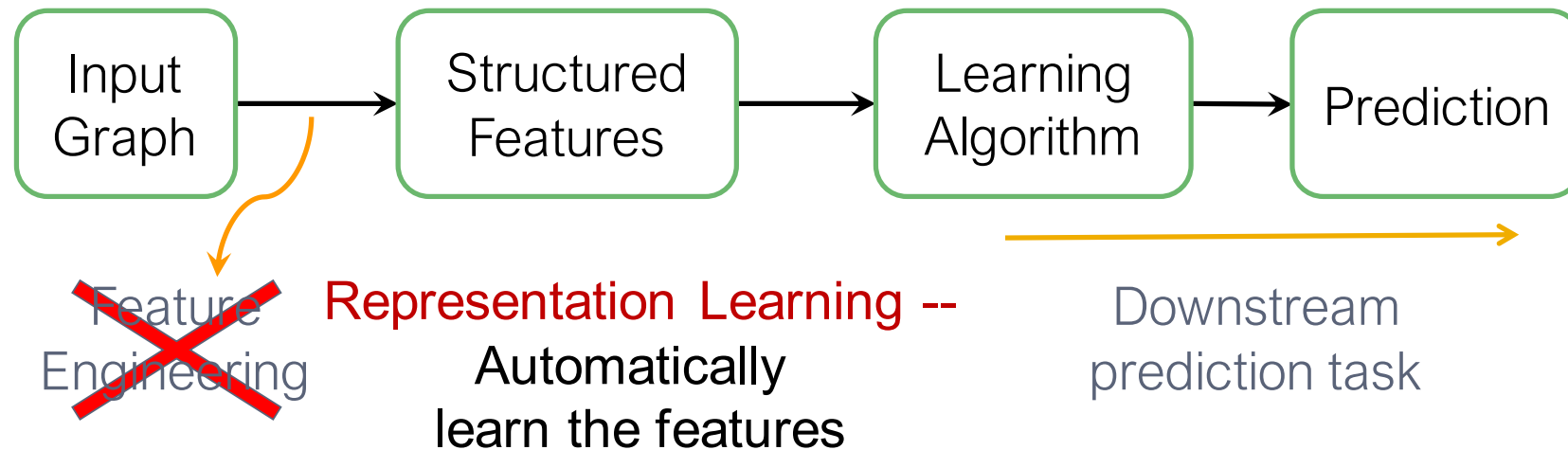
# Summary so far: feature engineering

- Node-level:
  - Node degree, centrality, clustering coefficient, graphlets
- Link-level:
  - Distance-based feature
  - Local/global neighborhood overlap
- Graph-level:
  - Graphlet kernel



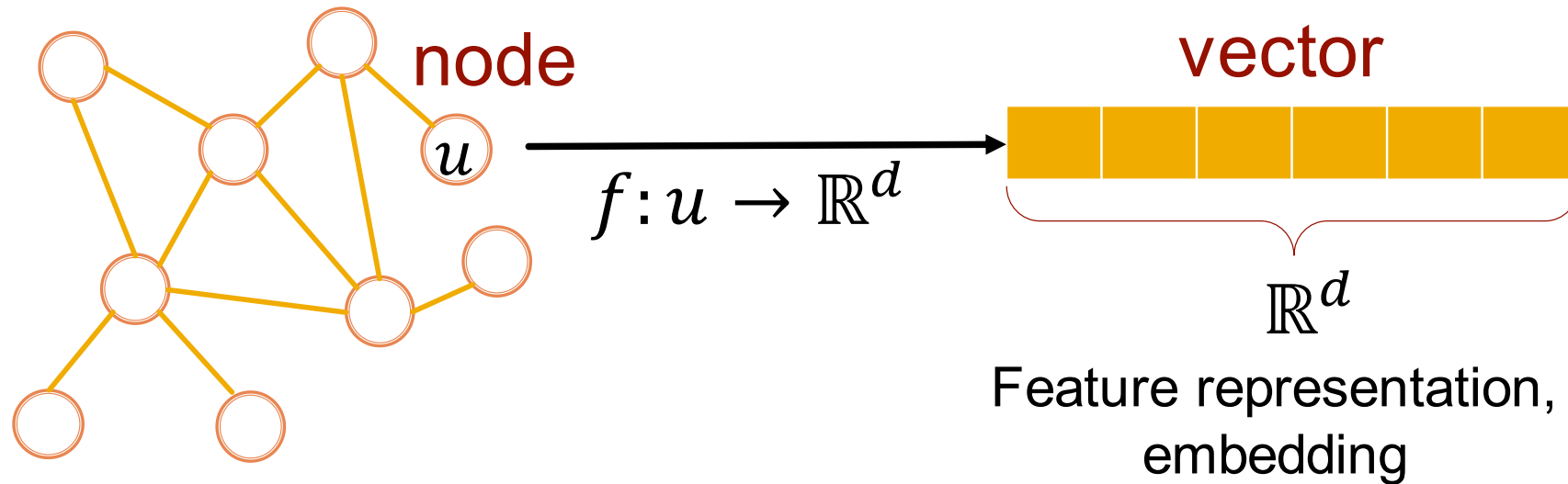
# Graph Representation Learning

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



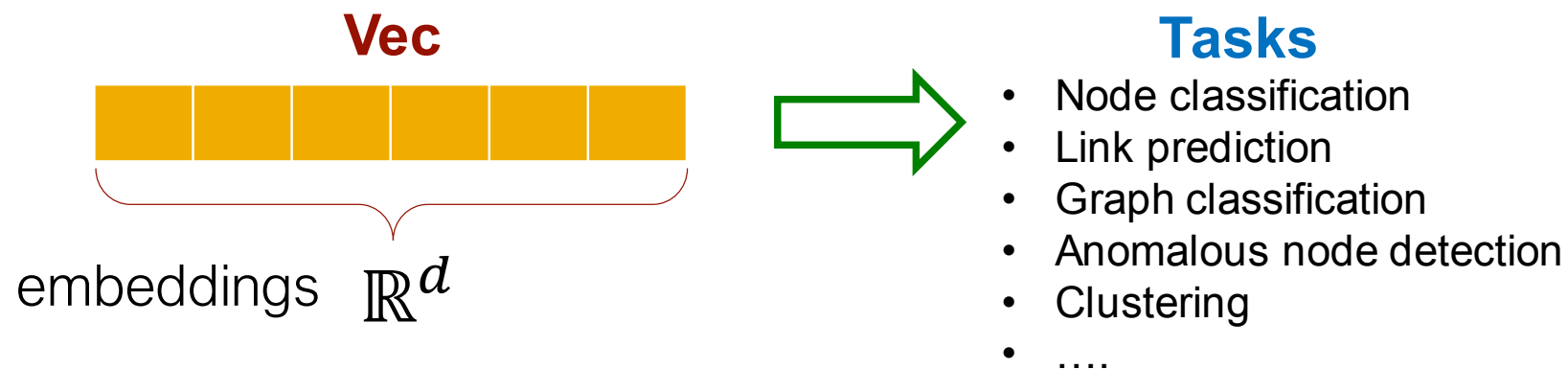
# Graph Representation Learning

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



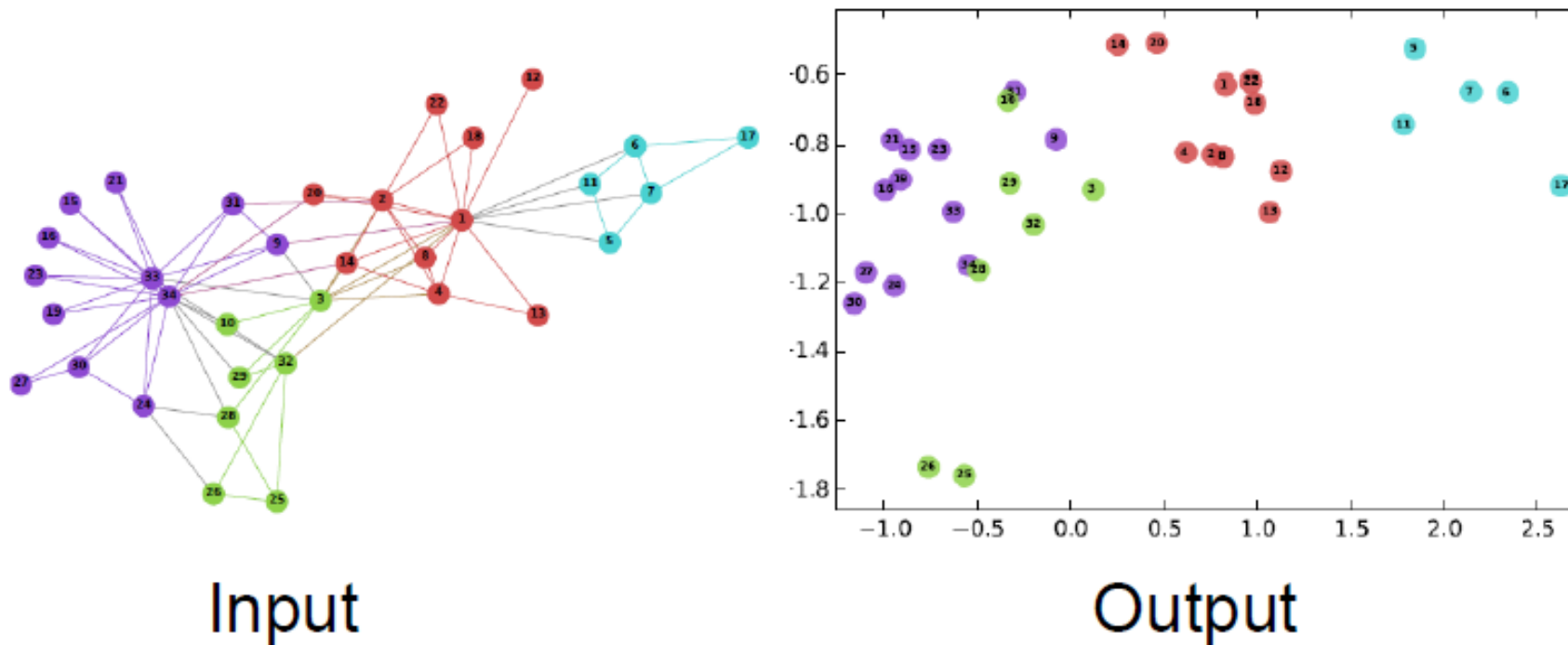
# Node Embedding

- **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



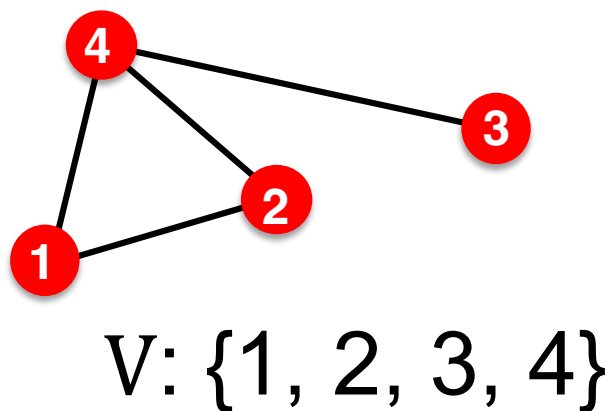
# Example Node Embedding

- 2D embedding of nodes of the Zachary's Karate Club network:



# Node Embedding: Setup

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

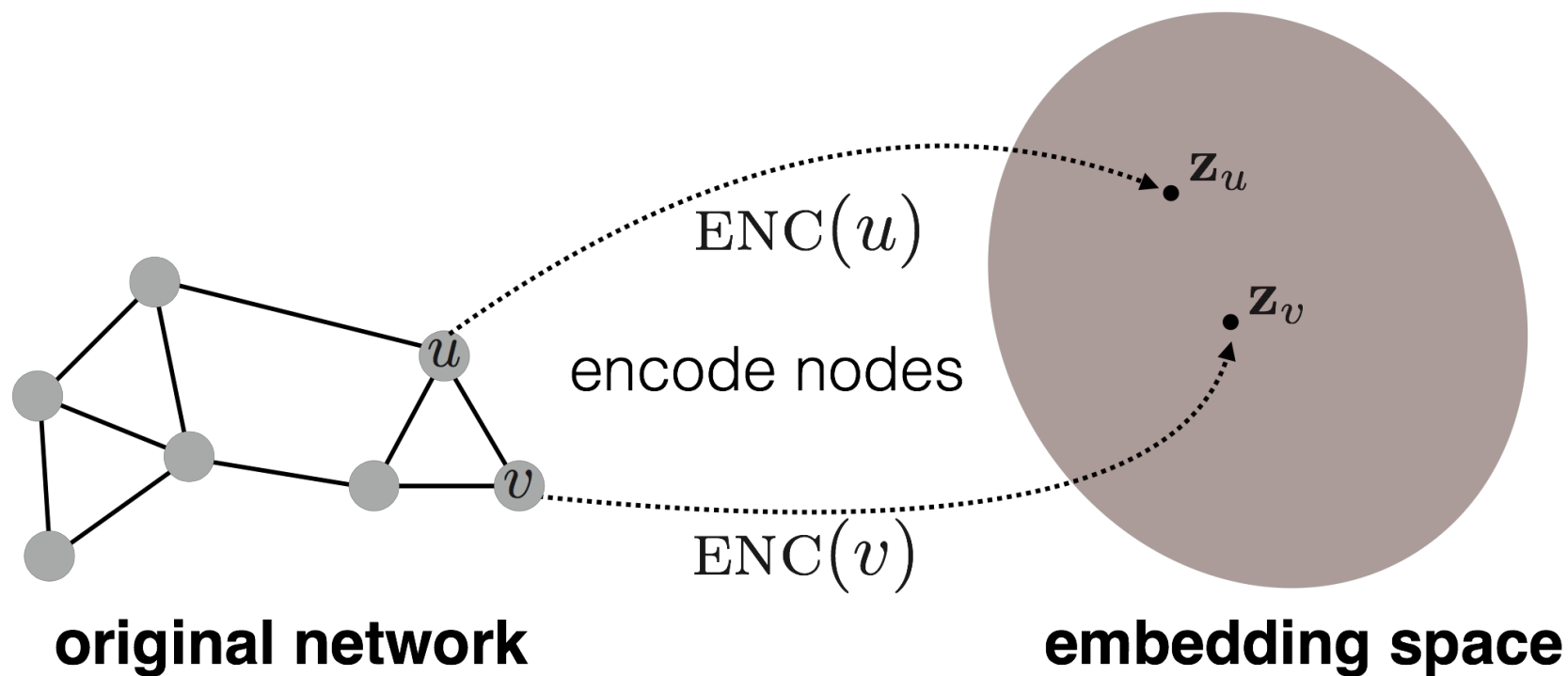


$$A = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$



# Node Embedding

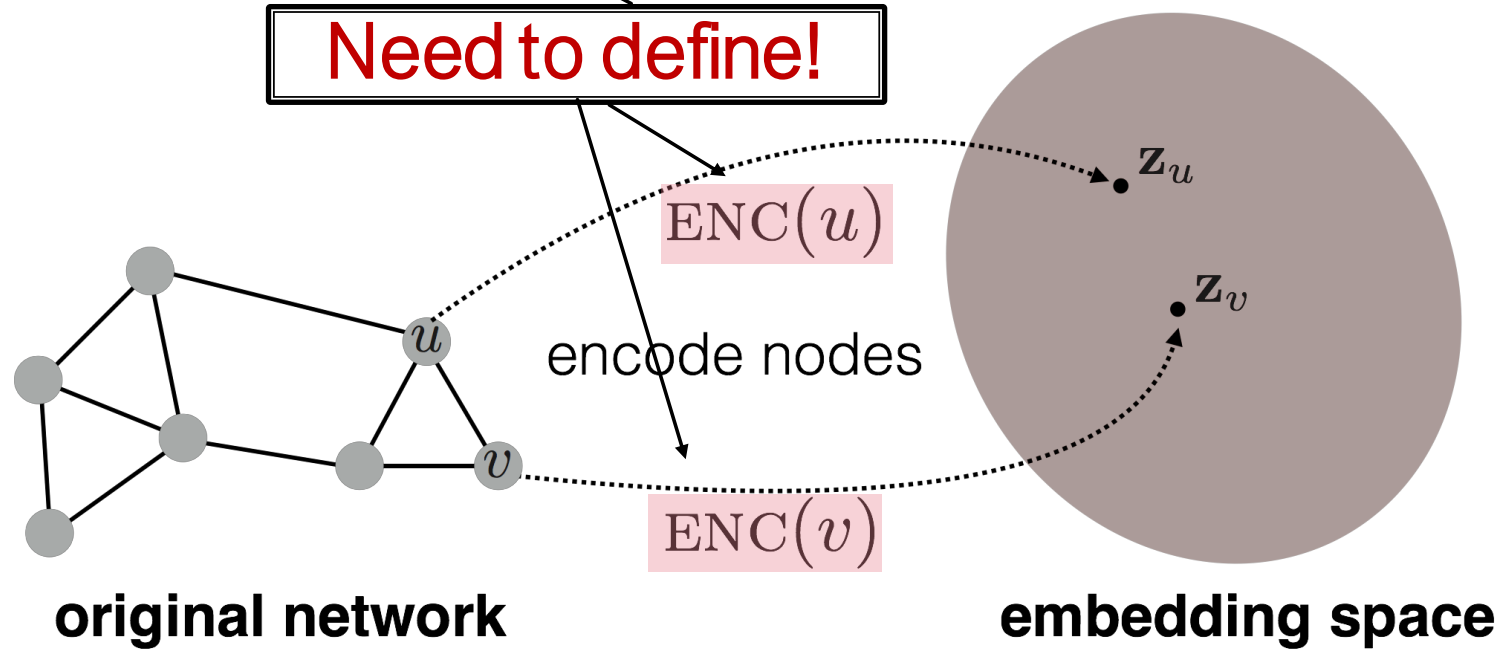
- Goal is to encode nodes so that **similarity in the embedding space** (e.g., dot product) approximates **similarity in the graph**



# Node Embedding

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

**Need to define!**



# Node Embedding: Key Components

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

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

node in the input graph

$d$ -dimensional embedding

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

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

Similarity of  $u$  and  $v$  in the original network

**Decoder**

dot product between node embeddings

# “Shallow” Encoding

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

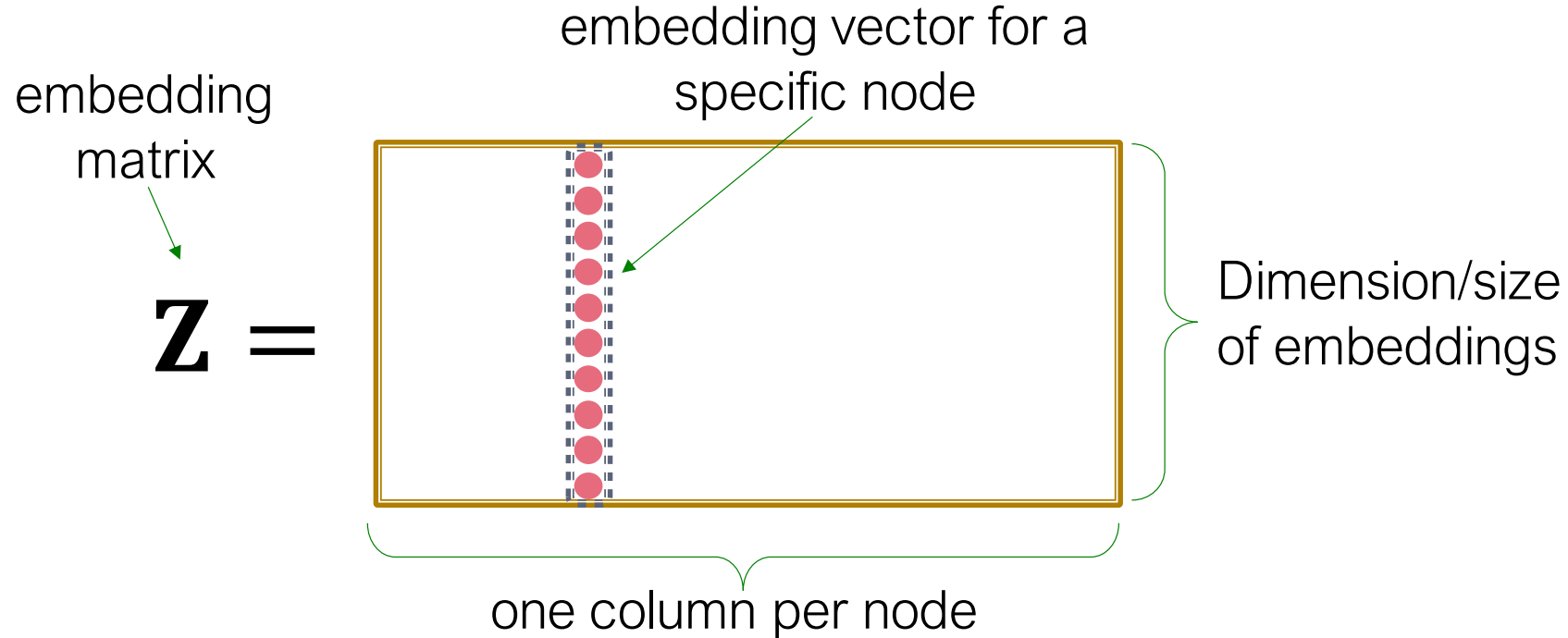
$$\text{ENC}(v) = \mathbf{z}_v = \mathbf{Z} \cdot v$$

$\mathbf{Z} \in \mathbb{R}^{d \times |\mathcal{V}|}$  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**



# “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)

Many methods: DeepWalk, node2vec

# 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  $\mathbf{z}_v^T \mathbf{z}_u$  for node pairs  $(u, v)$  that are **similar**

# Discussion: 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”?



Questions?