CSCI 566: Deep Learning and its Applications

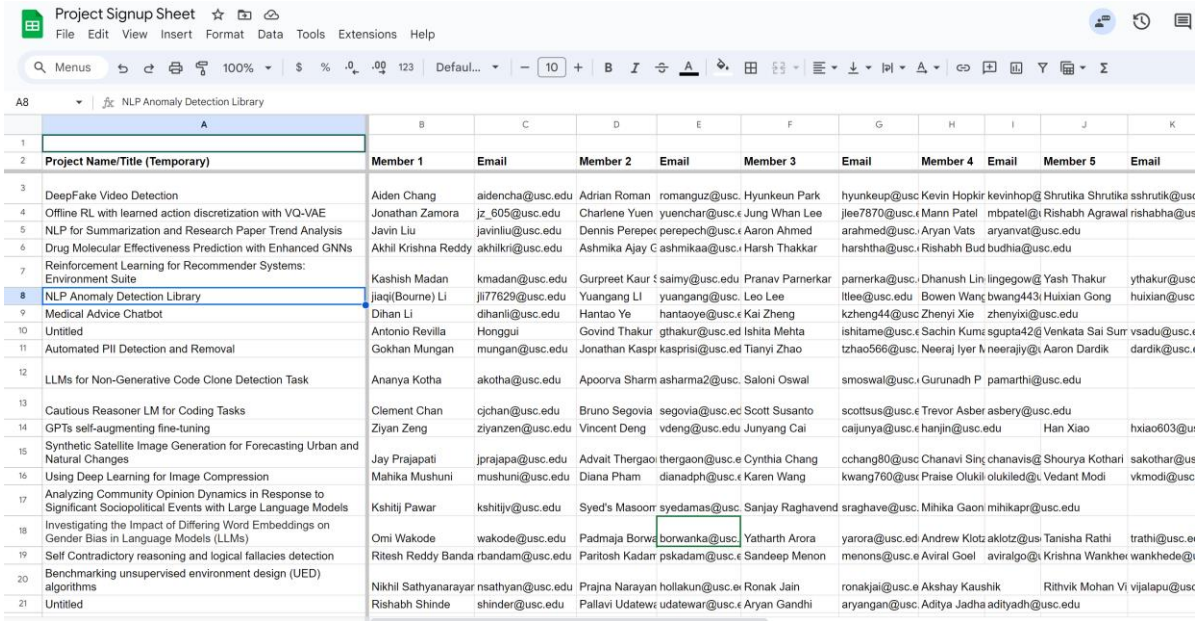
Yue Zhao
Thomas Lord Department of Computer Science
University of Southern California

Credits to previous versions of USC CSCI566, CMU 10601/701, Stanford CS 229, 231n, 224w

Special thanks to Prof. Jure Leskovec on GNN notes.

Logistics

Project titles (please add them so we could assign a TA):



Project Signup Sheet

File Edit View Insert Format Data Tools Extensions Help

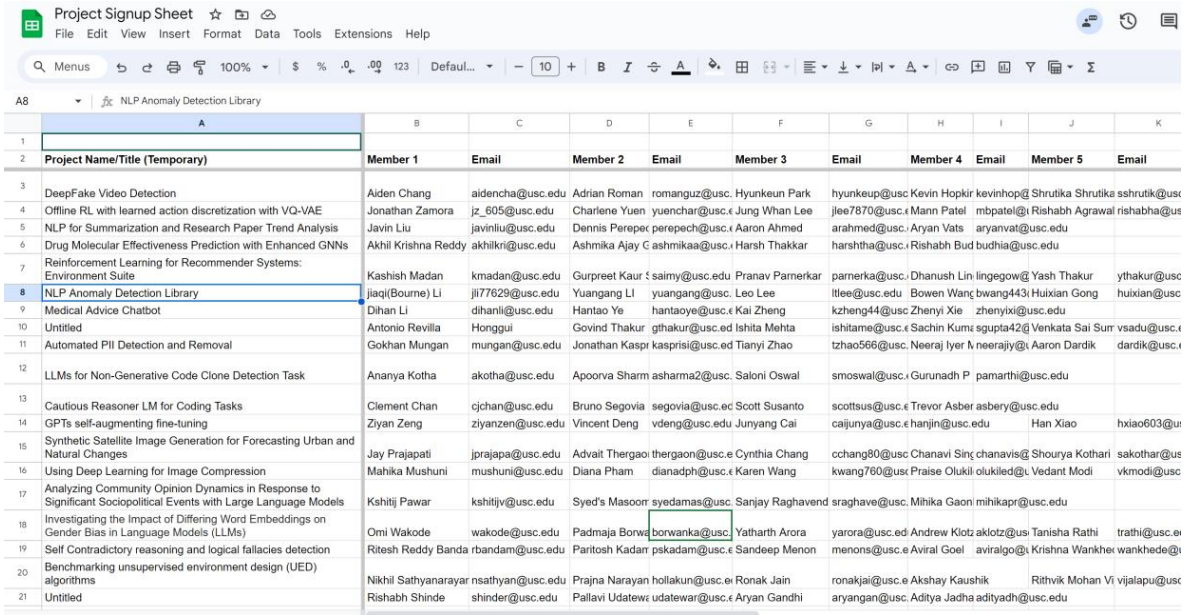
A8 NLP Anomaly Detection Library

	Project Name/Title (Temporary)	Member 1	Email	Member 2	Email	Member 3	Email	Member 4	Email	Member 5	Email
3	DeepFake Video Detection	Aiden Chang	aidencha@usc.edu	Adrian Roman	romanguz@usc.	Hyunkeun Park	hyunkeup@usc	Kevin Hopkir	kevinhop@	Shrutika Shrutika	sshrutik@usc
4	Offline RL with learned action discretization with VQ-VAE	Jonathan Zamora	jz_605@usc.edu	Charlene Yuen	yuenchar@usc.e	Jung Whan Lee	jlee7870@usc.e	Mann Patel	mbpatel@u	Rishabh Agrawal	rishabha@us
5	NLP for Summarization and Research Paper Trend Analysis	Javin Liu	javinliu@usc.edu	Dennis Perepec	perepech@usc.e	Aaron Ahmed	arahmed@usc.	Aryan Vats	aryanvat@usc.edu		
6	Drug Molecular Effectiveness Prediction with Enhanced GNNs	Akhil Krishna Reddy	akhilkri@usc.edu	Ashmika Ajay G	ashmikaa@usc.e	Harsh Thakkar	harshtha@usc.e	Rishabh Bud	budhia@usc.edu		
7	Reinforcement Learning for Recommender Systems: Environment Suite	Kashish Madan	kmadan@usc.edu	Gurpreet Kaur S	saimy@usc.edu	Pranav Parnerkar	parnerka@usc.	Dhanush Lin	lingegow@	Yash Thakur	ythakur@usc
8	NLP Anomaly Detection Library	jiaqi(Bourne) Li	jli77629@usc.edu	Yuangang Li	yuangang@usc.	Leo Lee	ltlee@usc.edu	Bowen Wang	bwang443(	Huixian Gong	huixian@usc
9	Medical Advice Chatbot	Dihan Li	dihanli@usc.edu	Hantao Ye	hantaoye@usc.e	Kai Zheng	kzheng44@usc	Zhenyi Xie	zhenyixi@usc.edu		
10	Untitled	Antonio Revilla	Honggui	Govind Thakur	gthakur@usc.ed	Ishita Mehta	ishitame@usc.e	Sachin Kumz	sgupta42@	Venkata Sai Sum	vsadu@usc.e
11	Automated PII Detection and Removal	Gokhan Mungan	mungan@usc.edu	Jonathan Kaspr	kasprisi@usc.ed	Tianyi Zhao	tzhao566@usc.	Neeraj Iyer N	neerajiy@u	Aaron Dardik	dardik@usc.e
12	LLMs for Non-Generative Code Clone Detection Task	Ananya Kotha	akotha@usc.edu	Apoorva Sharm	asharma2@usc.	Saloni Oswal	smoswal@usc.e	Gurunadh P	pamarthi@usc.edu		
13	Cautious Reasoner LM for Coding Tasks	Clement Chan	cjchan@usc.edu	Bruno Segovia	segovia@usc.ed	Scott Susanto	scottsus@usc.e	Trevor Asber	asbery@usc.edu		
14	GPTs self-augmenting fine-tuning	Ziyan Zeng	ziyanzen@usc.edu	Vincent Deng	vdeng@usc.edu	Junyang Cai	caijunya@usc.e	hanjin@usc.edu		Han Xiao	hxiao603@us
15	Synthetic Satellite Image Generation for Forecasting Urban and Natural Changes	Jay Prajapati	jprajapa@usc.edu	Advait Thergao	thergaon@usc.e	Cynthia Chang	cchang80@usc	Chanavi Sing	chanavis@	Shourya Kothari	sakothar@us
16	Using Deep Learning for Image Compression	Mahika Mushuni	mushuni@usc.edu	Diana Pham	dianadph@usc.e	Karen Wang	kwang760@usc	Praise Olukil	olukiled@u	Vedant Modi	vkmodi@usc
17	Analyzing Community Opinion Dynamics in Response to Significant Sociopolitical Events with Large Language Models	Kshitij Pawar	kshitijv@usc.edu	Syed's Masoom	syedamas@usc.	Sanjay Raghavend	sraghave@usc.	Mihika Gaon	mihikapr@usc.edu		
18	Investigating the Impact of Differing Word Embeddings on Gender Bias in Language Models (LLMs)	Omi Wakode	wakode@usc.edu	Padmaja Borwa	borwanka@usc.	Yatharth Arora	yarora@usc.ed	Andrew Klotz	aklotz@usc	Tanisha Rathi	trathi@usc.e
19	Self Contradictory reasoning and logical fallacies detection	Ritesh Reddy Banda	rbandam@usc.edu	Paritosh Kadam	pskadam@usc.e	Sandeep Menon	menons@usc.e	Aviral Goel	aviralgo@u	Krishna Wankhe	wankhede@u
20	Benchmarking unsupervised environment design (UED) algorithms	Nikhil Sathyanarayar	nsathyan@usc.edu	Prajna Narayan	hollakun@usc.e	Ronak Jain	ronakjai@usc.e	Akshay Kaushik		Rithvik Mohan Vi	vijalapu@usc
21	Untitled	Rishabh Shinde	shinder@usc.edu	Pallavi Udatewa	udatewar@usc.e	Aryan Gandhi	aryangan@usc.	Aditya Jadha	adityadh@usc.edu		

USC Viterbi School of Engineering

Logistics

If you do not put your project here – you consent to do this without TA's help…



Project Signup Sheet

File Edit View Insert Format Data Tools Extensions Help

A8 | NLP Anomaly Detection Library

	Project Name/Title (Temporary)	Member 1	Email	Member 2	Email	Member 3	Email	Member 4	Email	Member 5	Email
3	DeepFake Video Detection	Aiden Chang	aidencha@usc.edu	Adrian Roman	romanguz@usc.	Hyunkeun Park	hyunkeup@usc	Kevin Hopkir	kevinhop@	Shrutika Shrutika	sshrutik@usc
4	Offline RL with learned action discretization with VQ-VAE	Jonathan Zamora	jz_605@usc.edu	Charlene Yuen	yuenchar@usc.e	Jung Whan Lee	jlee7870@usc.e	Mann Patel	mbpatel@u	Rishabh Agrawal	rishabha@us
5	NLP for Summarization and Research Paper Trend Analysis	Javin Liu	javinliu@usc.edu	Dennis Perepec	perepech@usc.e	Aaron Ahmed	arahmed@usc.e	Aryan Vats	aryanvat@usc.edu		
6	Drug Molecular Effectiveness Prediction with Enhanced GNNs	Akhil Krishna Reddy	akhilkri@usc.edu	Ashmika Ajay G	ashmikaa@usc.e	Harsh Thakkar	harshtha@usc.e	Rishabh Bud	budhia@usc.edu		
7	Reinforcement Learning for Recommender Systems: Environment Suite	Kashish Madan	kmadan@usc.edu	Gurpreet Kaur S	saimy@usc.edu	Pranav Parnerkar	parnerka@usc.e	Dhanush Lin	lingegow@	Yash Thakur	ythakur@usc
8	NLP Anomaly Detection Library	jiaqi(Bourne) Li	jli77629@usc.edu	Yuangang LI	yuangang@usc.	Leo Lee	ltlee@usc.edu	Bowen Wang	bwang443@	Huixian Gong	huixian@usc
9	Medical Advice Chatbot	Dihan Li	dihanli@usc.edu	Hantao Ye	hantaoye@usc.e	Kai Zheng	kzheng44@usc	Zhenyi Xie	zhenyixi@usc.edu		
10	Untitled	Antonio Revilla	Honggui	Govind Thakur	gthakur@usc.ed	Ishita Mehta	ishitame@usc.e	Sachin Kuma	sgupta42@	Venkata Sai Sum	vsadu@usc.e
11	Automated PII Detection and Removal	Gokhan Mungan	mungan@usc.edu	Jonathan Kaspr	kasprisi@usc.ed	Tianyi Zhao	tzhao566@usc.	Neeraj Iyer N	neerajiy@u	Aaron Dardik	dardik@usc.e
12	LLMs for Non-Generative Code Clone Detection Task	Ananya Kotha	akotha@usc.edu	Apoorva Sharm	asharma2@usc.	Saloni Oswal	smoswal@usc.e	Gurunadh P	pamarthi@usc.edu		
13	Cautious Reasoner LM for Coding Tasks	Clement Chan	cjchan@usc.edu	Bruno Segovia	segovia@usc.ed	Scott Susanto	scottsus@usc.e	Trevor Asber	asbery@usc.edu		
14	GPTs self-augmenting fine-tuning	Ziyan Zeng	ziyanzen@usc.edu	Vincent Deng	vdeng@usc.edu	Junyang Cai	caijunya@usc.e	hanjin@usc.edu		Han Xiao	hxiao603@u
15	Synthetic Satellite Image Generation for Forecasting Urban and Natural Changes	Jay Prajapati	jprajapa@usc.edu	Advait Thergao	thergaon@usc.e	Cynthia Chang	cchang80@usc	Chanavi Sing	chanavis@	Shourya Kothari	sakothar@us
16	Using Deep Learning for Image Compression	Mahika Mushuni	mushuni@usc.edu	Diana Pham	dianadph@usc.e	Karen Wang	kwang760@usc	Praise Olukil	olukiled@u	Vedant Modi	vkmodi@usc
17	Analyzing Community Opinion Dynamics in Response to Significant Sociopolitical Events with Large Language Models	Kshitij Pawar	kshitijv@usc.edu	Syed's Masoom	syedamas@usc.	Sanjay Raghavend	sraghave@usc.	Mihika Gaon	mihikapr@usc.edu		
18	Investigating the Impact of Differing Word Embeddings on Gender Bias in Language Models (LLMs)	Omi Wakode	wakode@usc.edu	Padmaja Borwa	borwanka@usc.	Yatharth Arora	yarora@usc.ed	Andrew Klotz	aklotz@usc	Tanisha Rathi	trathi@usc.e
19	Self Contradictory reasoning and logical fallacies detection	Ritesh Reddy Banda	rbandam@usc.edu	Paritosh Kadam	pskadam@usc.e	Sandeep Menon	menons@usc.e	Aviral Goel	aviralgo@u	Krishna Wankhe	wankhede@
20	Benchmarking unsupervised environment design (UED) algorithms	Nikhil Sathyanarayar	nsathyan@usc.edu	Prajna Narayan	hollakun@usc.e	Ronak Jain	ronakjai@usc.e	Akshay Kaushik		Rithvik Mohan Vi	vijalapu@usc
21	Untitled	Rishabh Shinde	shinder@usc.edu	Pallavi Udatewa	udatewar@usc.e	Aryan Gandhi	aryangan@usc.	Aditya Jadha	adityadh@usc.edu		

USC Viterbi School of Engineering

Project Check-in

Each project has 2 mandatory check-in with your TA

- Potentially one before the mid-report due (March 22nd)
- Potentially another one before final report is due (May 3rd)
- We will work out the logistic then

We have 55 projects – each TA is in charge of 7-8 projects

- They will be your point of contact for project questions
- Use Piazza, office hours, or scheduled appointment with them to make sure your questions are addressed

Cloud Credits

First, **Google** has $300 for all new signups -- https://cloud.google.com/free?hl=en

- Also, we have applied for the Google Cloud credit ($50/student) -- considering pooling them for the project.

I have also applied for **USC advanced computing credit** -- https://www.carc.usc.edu/ to request the usage of it, please add your information

- https://docs.google.com/spreadsheets/d/1KWPlHyMVZ-2o47IhpFC9gxDYusbchPuvX971HhtUULI/edit?usp=sharing

Cloud Credits

Google Cloud Credit:

https://vector.my.salesforce-sites.com/GCPEDU?cid=U4cYmFRYRNMCY3QWOi3mu7niNVKxs%2Fq76pJeSiD9VVD23kqHSaeE71KE2OJOYly6/

USC Credit: https://docs.google.com/spreadsheets/d/1KWPlHyMVZ-2o47IhpFC9gxDYusbchPuvX971HhtUULI/edit#gid=0

See details: https://piazza.com/class/loendg1jxkryr/post/21

Assignments

Designed by TAs:

- neural network, decision tree, and random forests
- Note Python is a prerequisite of the course
- PyTorch has a 60 min quick tutorial: https://pytorch.org/tutorials/beginner/deep_learning_60min_blitz.html

We leverage Gradescope similarity check + point check:

- Also potentially my research, which is?
- Do the homework independently while general discussion is allowed
- ChatGPT is not recommended, but hard to check though…

Assignments

Due in 3 weeks…

- The project should still be ongoing
- Questions on piazza will be addressed by TA
    - Yuehan, Varun, and Ziyi will help
- Also TA hours will be helpful for all the Tas

Do not do it in the last minute – recall we do not do late days due to the size of the course

Interaction on Piazza

Please post general questions to all students:

- You will get your answer much faster!
- Again – do not post your assignment answer, but the general questions are fine!

If you actively answer others' questions, we will take note of your name – if you need a bump by the end of the semester, we will get you there automatically due to your contribution to the course!

TA Office Hours

We have a slightly easier way to handle office hours (not that ideal)

Zihao He 4 days ago

Actions

Hi everyone, we will share our TA OHs in here in the future.

	Varun Bhatt	Zihao He	Zizhao Hu	Ayush Jain	Ziyi Liu	Yuehan Qin	Pengda Xiang
Week of Feb 5	Tuesday 14:00 - 15:00, RTH 419 (preferred) or Zoom,	Thu, 2-3pm Leavey 202C, Zoom Link	Wed, 10-11am Leavey, 201F Zoom	Wed, 2-3pm RTH 4th floor lobby		Tuesday 10:00-11:00 Zoom	
Week of Feb 12							

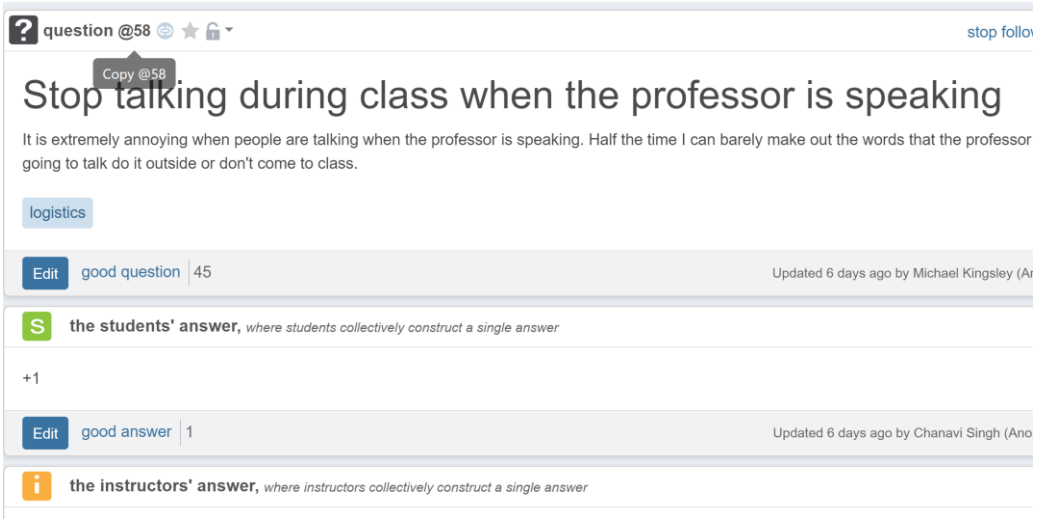
good comment | 2

Midterm – Note there is no Final!

Feb 23rd – **in-class** – **paper-based** – open books (although I have no idea which books to use)

- 50 + 6 (bonus) multiple choice questions + single answer questions, e.g., a single number
- there is no make-up unless you have medical emergency with proof ☺
- 1 bonus for each quiz and three bonus for the exam (0.5 each so it is 6 questions)

Stop Talking during the Lecture



- Ask these students to share their questions/topics on the stage
- Also, I will start walking around during the lecture time to help address these questions more closely (for the benefit of my neck)

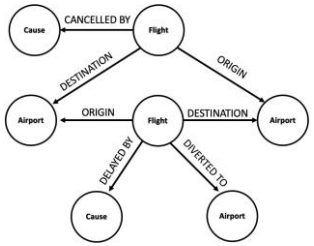
Graph Neural Networks (GNN)

Basics

Why Graphs?

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

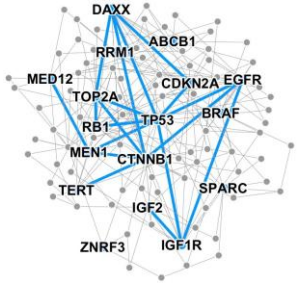
Many Types of Data are Graphs (1)



Event Graphs

Image credit: SalientNetworks

Computer Networks



Disease Pathways

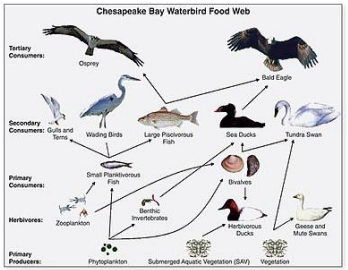


Image credit: Wikipedia

Food Webs

Image credit: Pinterest

Particle Networks

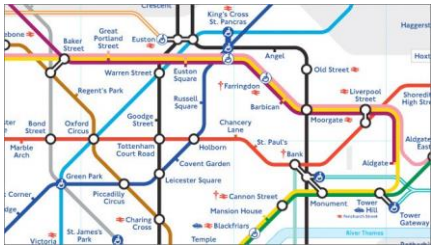


Image credit: visitlondon.com

Underground Networks

USC Viterbi School of Engineering

Many Types of Data are Graphs (2)

Image credit: Medium

Social Networks

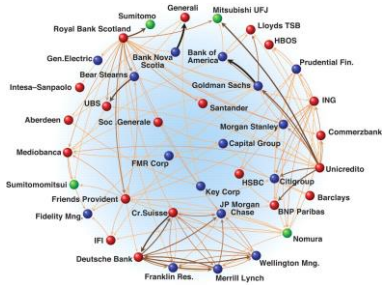


Image credit: Science

Economic Networks

Image credit: Lumen Learning

Communication Networks

Citation Networks

Image credit: Missoula Current News

Internet

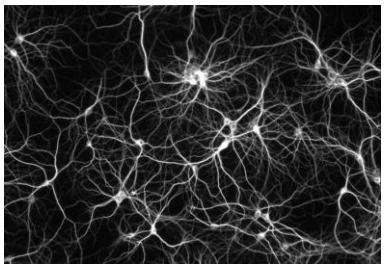


Image credit: The Conversation

Networks of Neurons

Many Types of Data are Graphs (3)

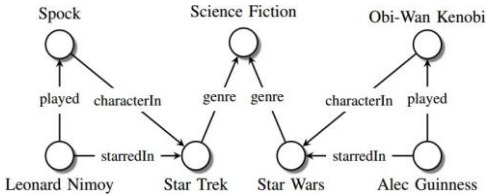


Image credit: Maximilian Nickel et al

Knowledge Graphs

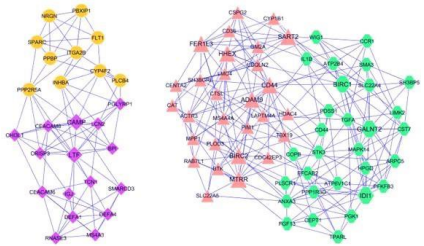


Image credit: ese.wustl.edu

Regulatory Networks

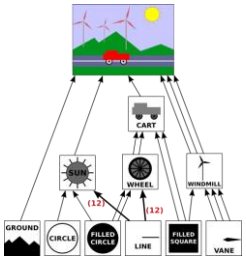


Image credit: math.hws.edu

Scene Graphs

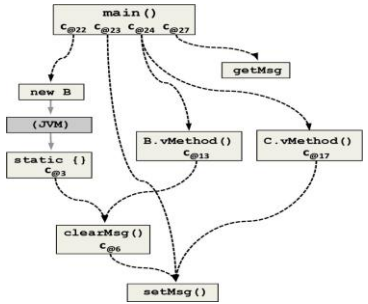
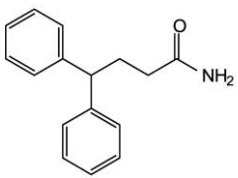


Image credit: ResearchGate

Code Graphs



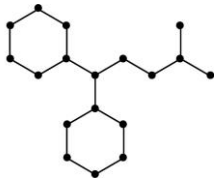


Image credit: MDPI

Molecules

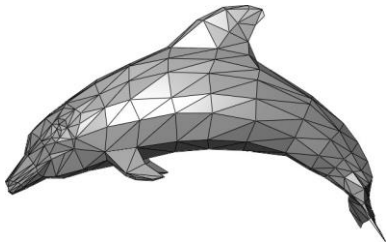


Image credit: Wikipedia

3D Shapes

USC Viterbi School of Engineering

Graphs: Machine Learning

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

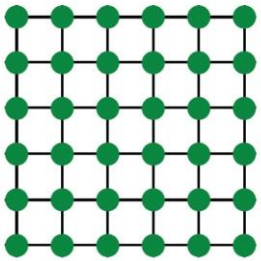
By explicitly modeling relationships, we achieve better performance!

Main question:

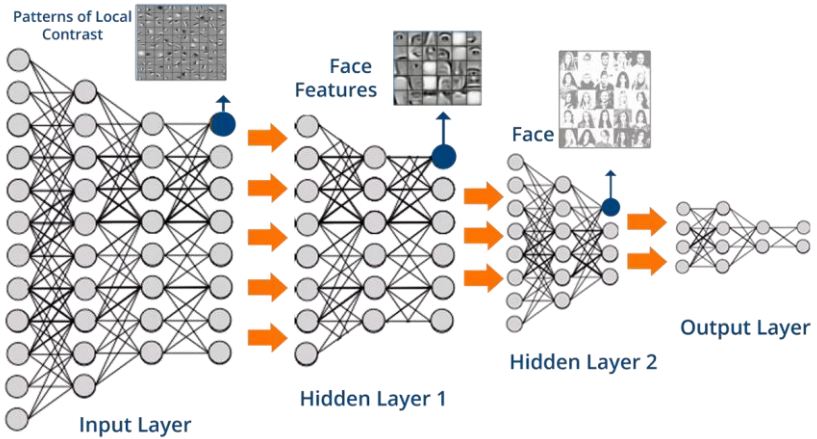
How do we take advantage of relational structure for better prediction?

Today: Modern ML Toolbox

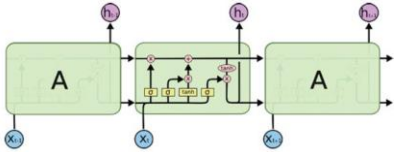
CNN



Images



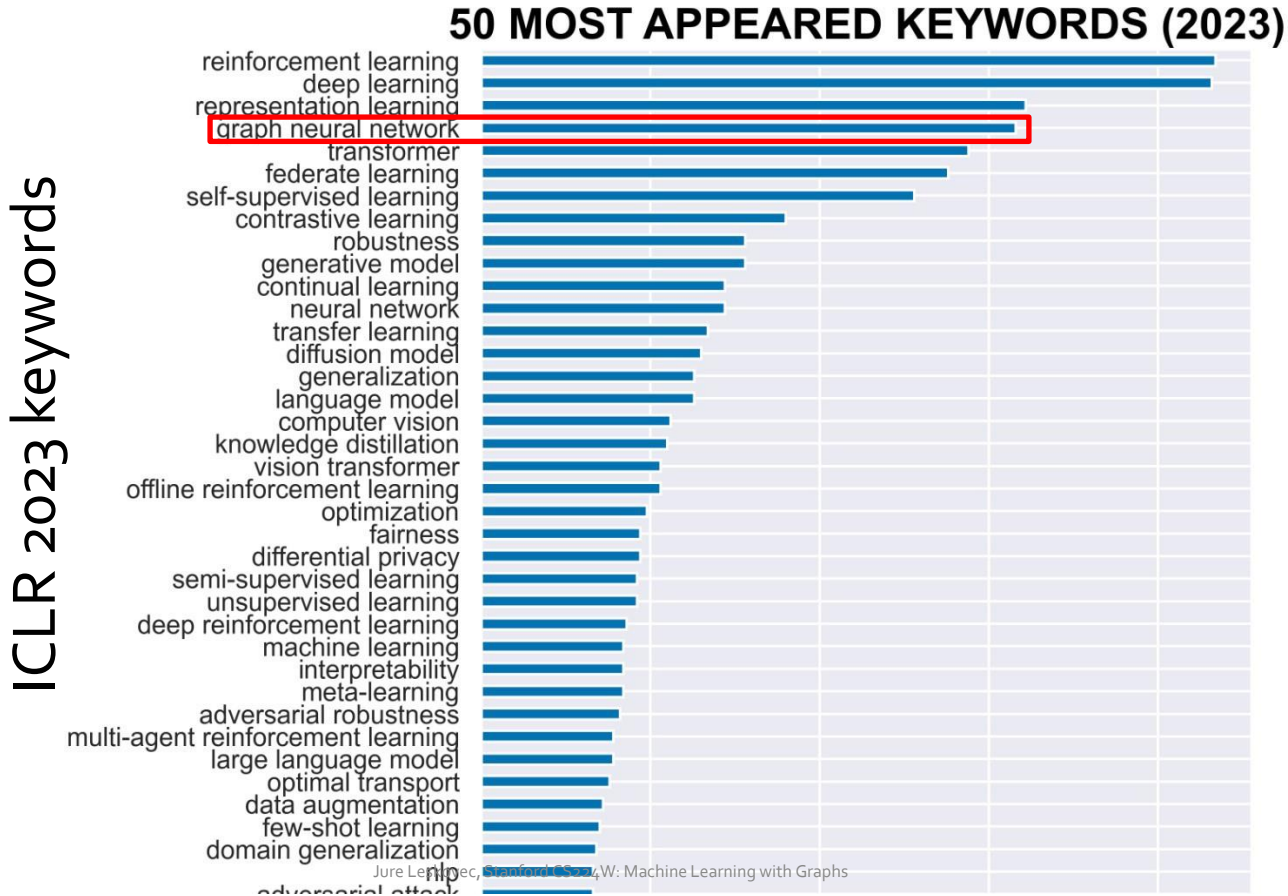
Text/Speech



RNN/ LSTM

Modern deep learning toolbox is designed for simple sequences & grids

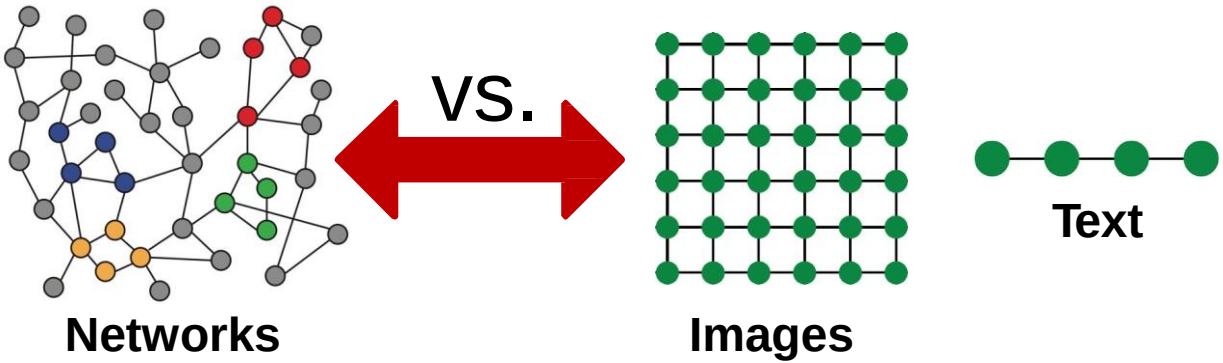
Hot subfield in ML



Why is Graph Deep Learning Hard?

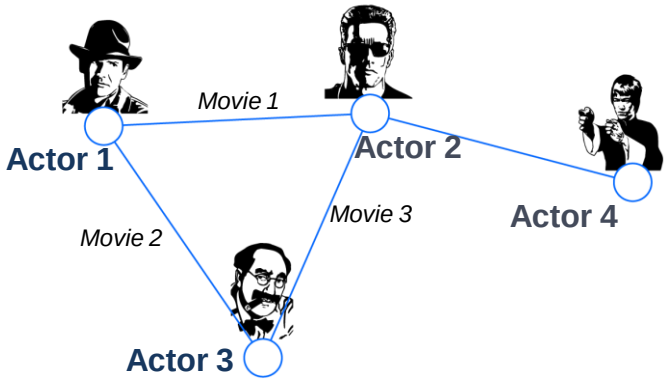
Networks are complex:

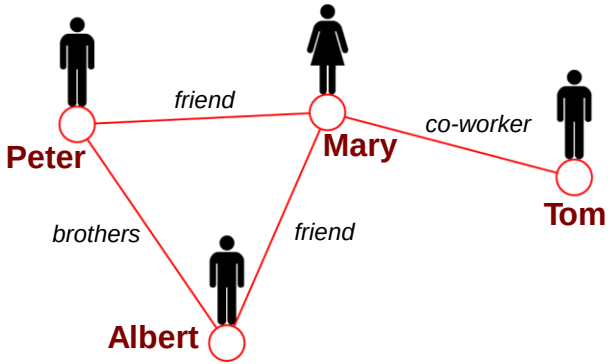
- Arbitrary size and complex topological structure (*i.e.*, no spatial locality like grids)

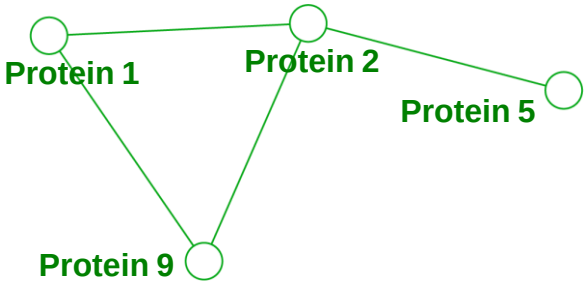


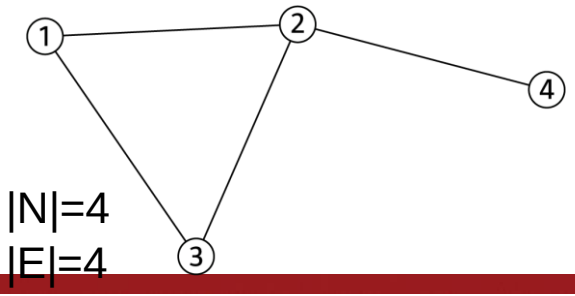
- No fixed node ordering or reference point
- Often dynamic and have multimodal features

Graphs: A Common Language









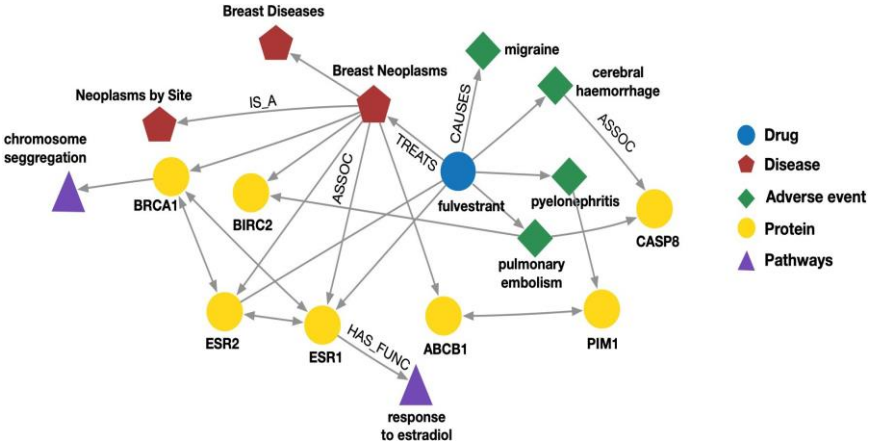
Heterogeneous Graphs

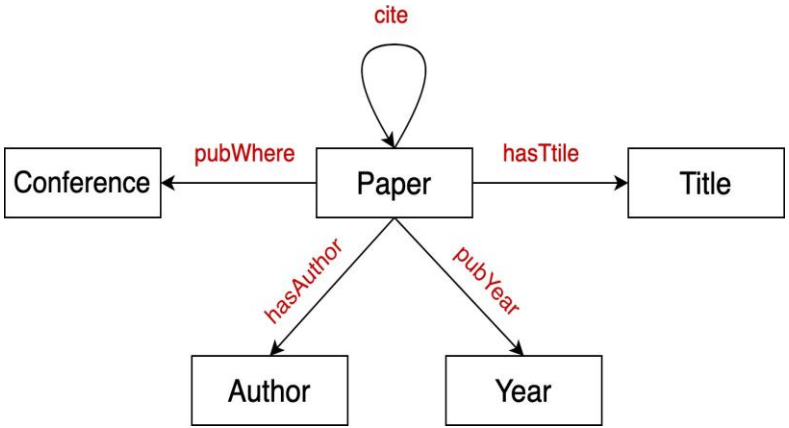
- **A heterogeneous graph is defined as**

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

  - 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

Many Graphs are Heterogeneous





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

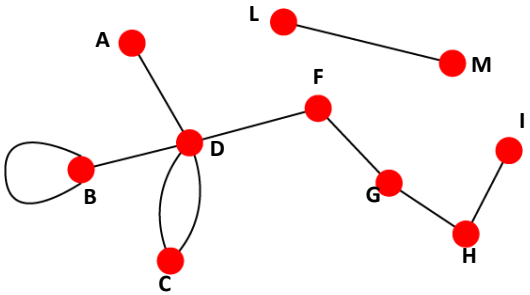
Choosing a Proper Representation

- **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

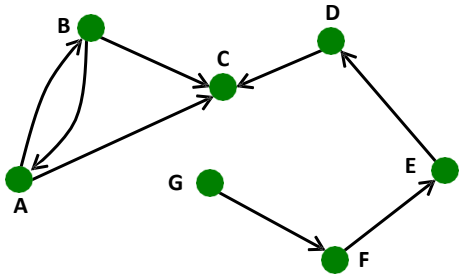
Undirected

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



Directed

- **Links:** directed

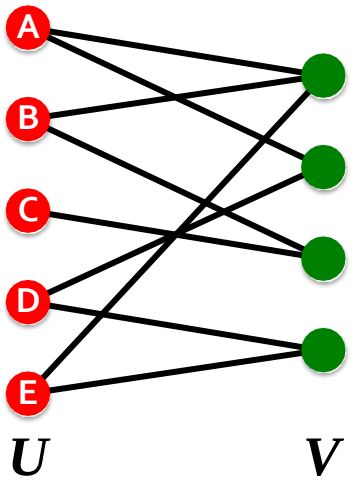


- **Other considerations:**
  - Weights
  - Properties
  - Types
  - Attributes

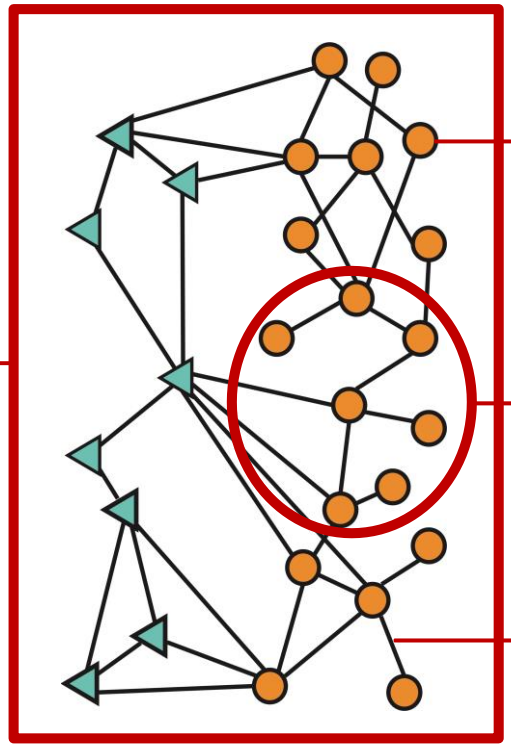
Bipartite Graph

- **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



Graph-level prediction, Graph generation

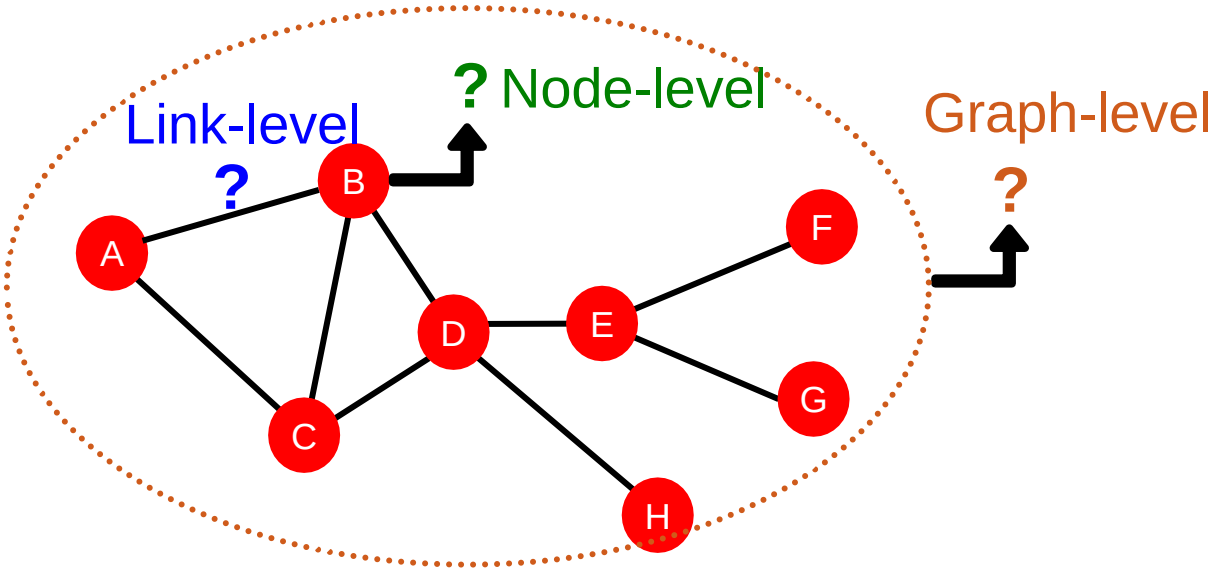
Node level

Community (subgraph) level

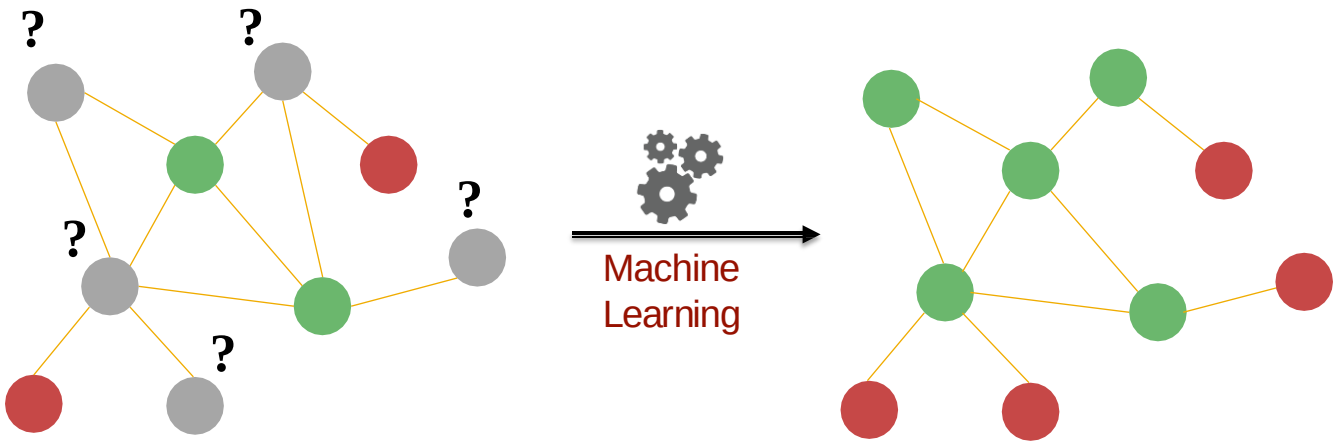
Edge-level

Machine Learning Tasks: Review

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



Node-Level Tasks

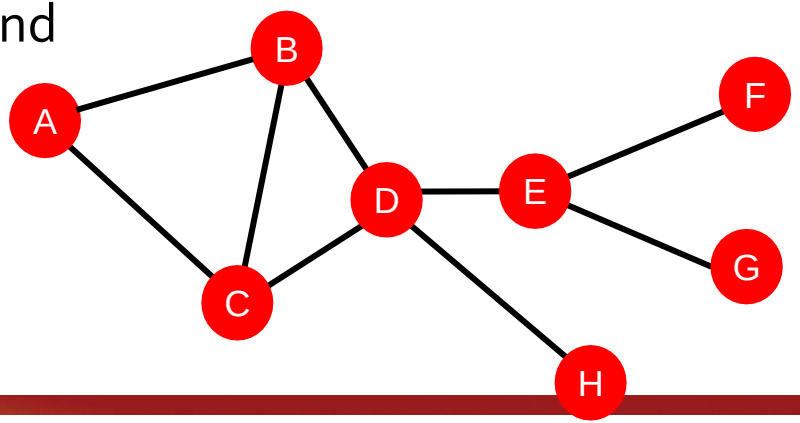


Node classification

Node-Level Network Structure

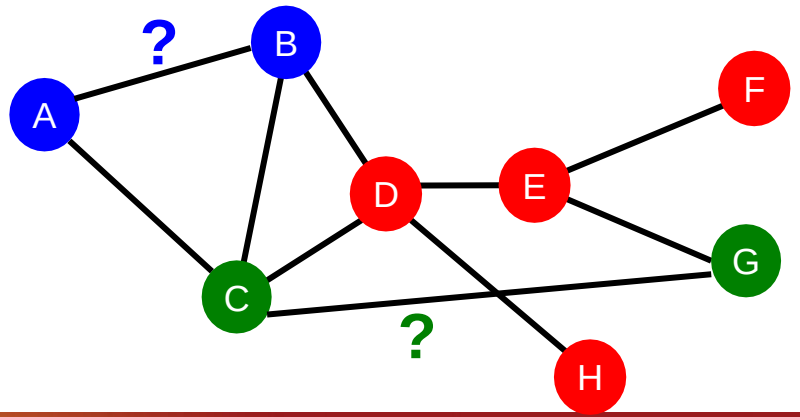
Goal: Characterize the structure and position of a node in the network:

- 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



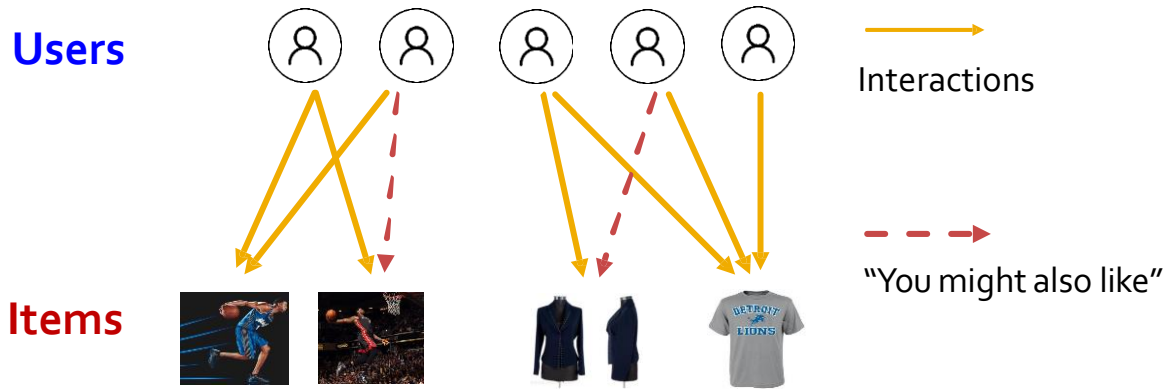
Link-Level Prediction Task

- 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**.



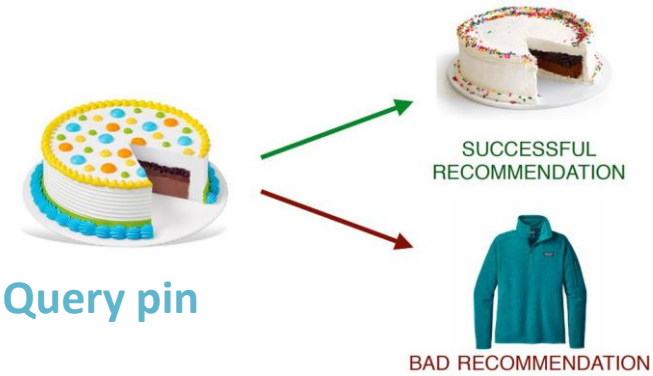
Example (1): Recommender Systems

- **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**



PinSage: Graph-based Recommender

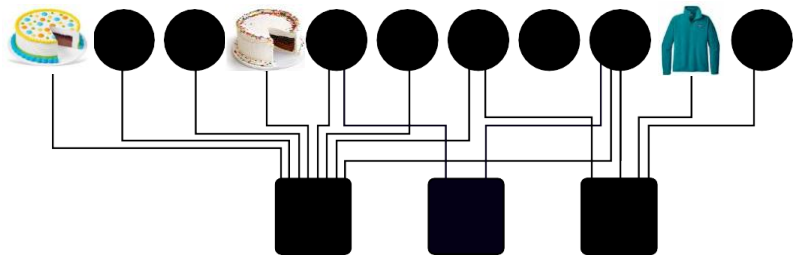
Task: Recommend related pins to users

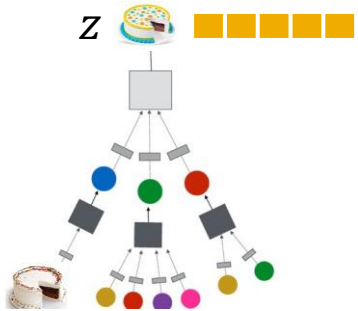


Task: Learn node embeddings z_i such that

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

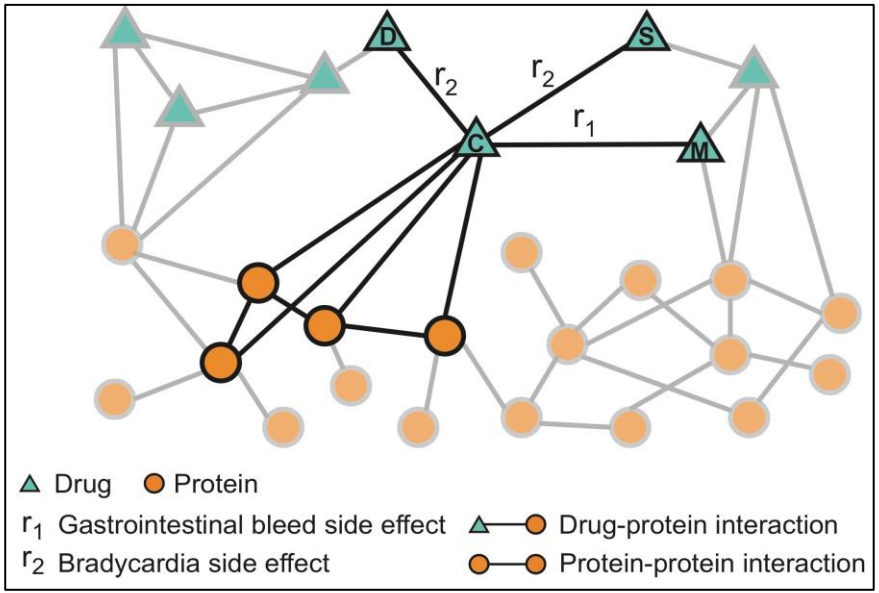
Predict whether two nodes in a graph are related



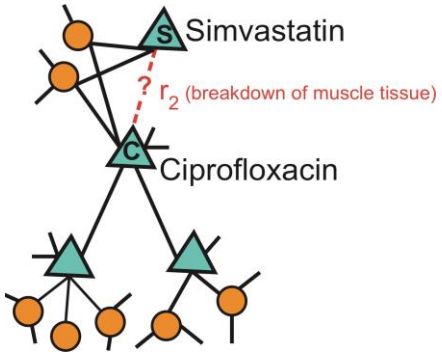


Biomedical Graph Link Prediction

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



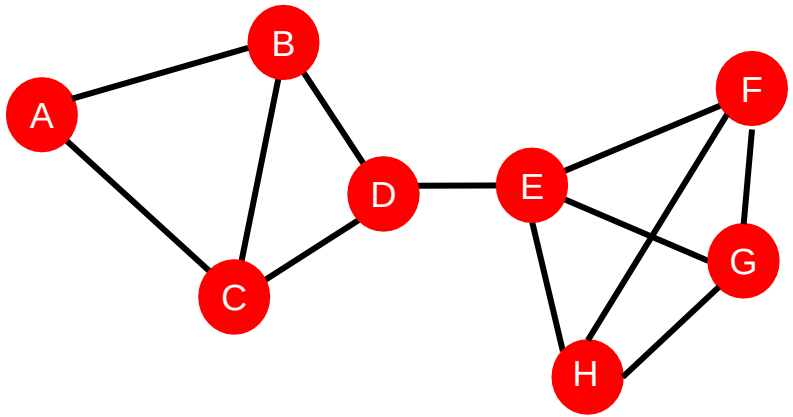
Query: How likely will Simvastatin and Ciprofloxacin, when taken together, break down muscle tissue?



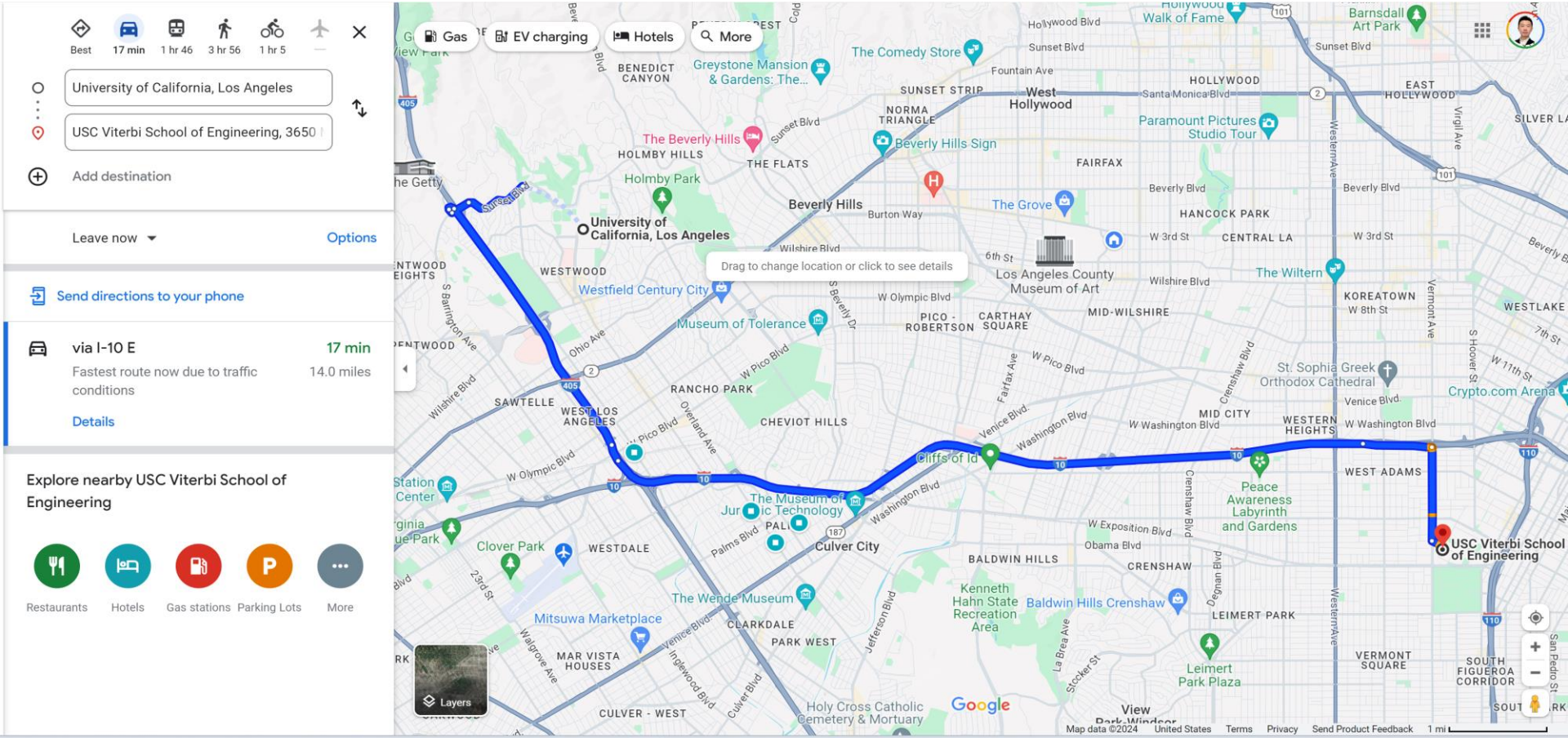
Graph-Level Features

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

- **For example:**

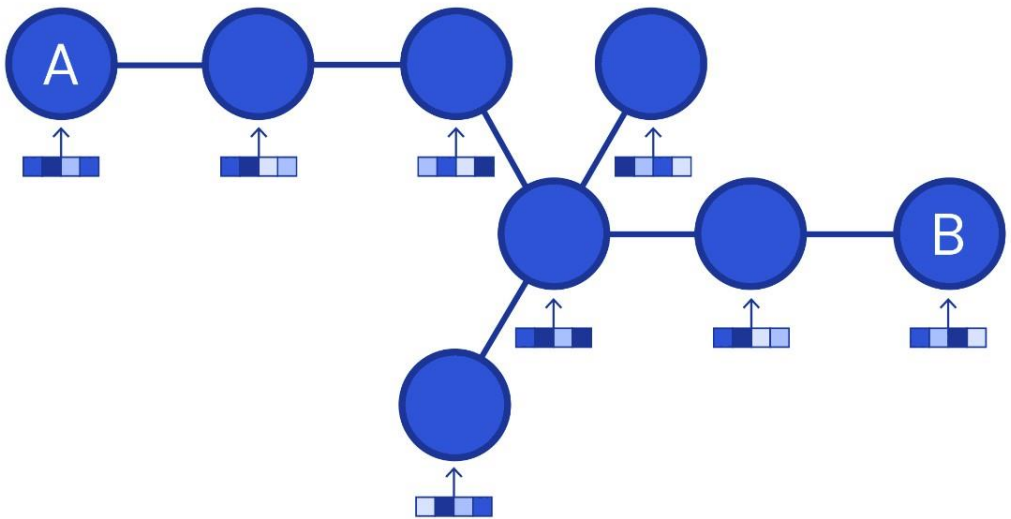


Example (1): Traffic Prediction



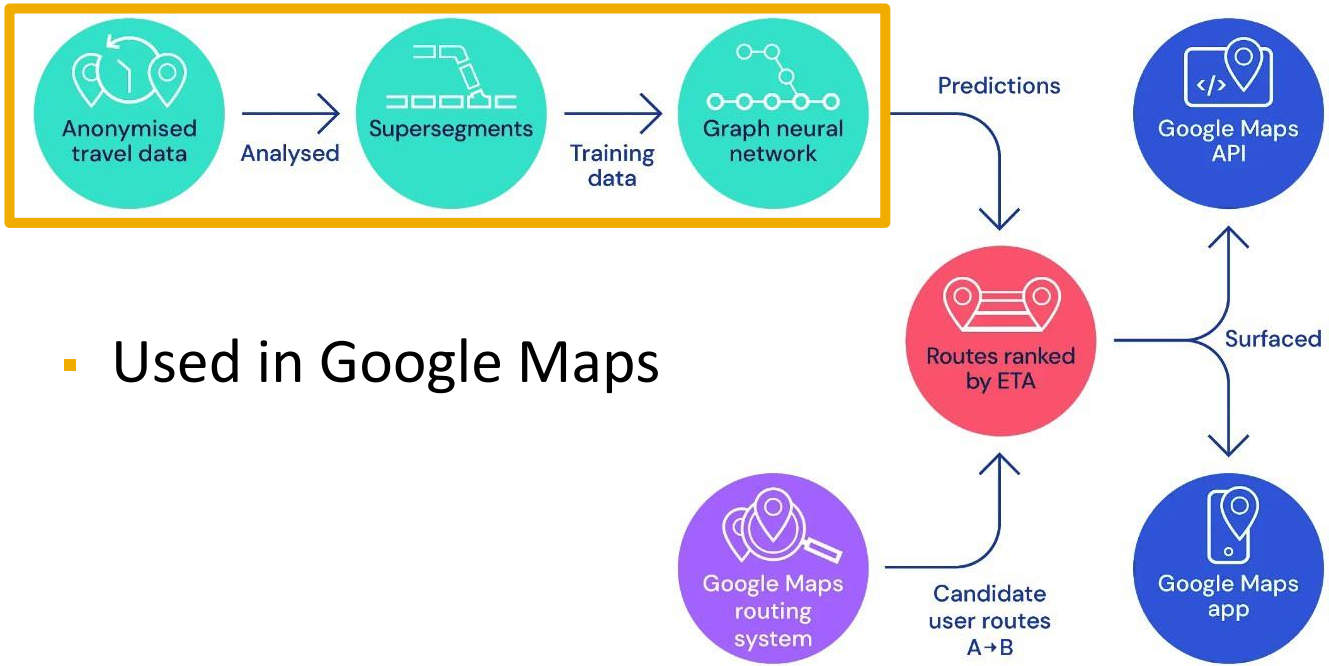
Road Network as a Graph

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



Traffic Prediction via GNN

Predicting Time of Arrival with Graph Neural Networks



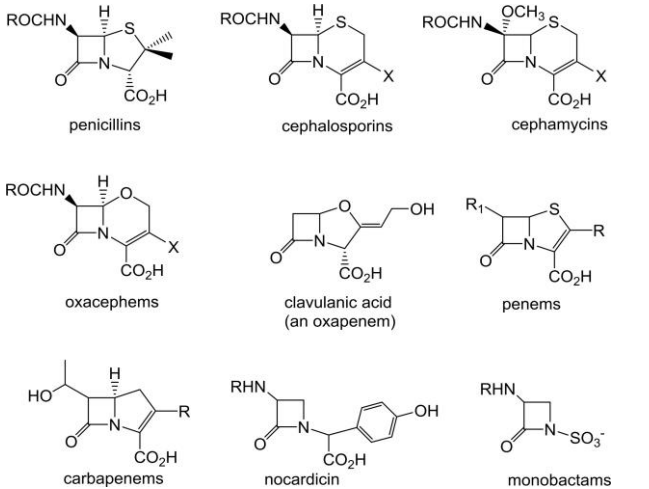
- Used in Google Maps

THE MODEL ARCHITECTURE FOR DETERMINING OPTIMAL ROUTES AND THEIR TRAVEL TIME.

Image credit: DeepMind

Example (2): 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.

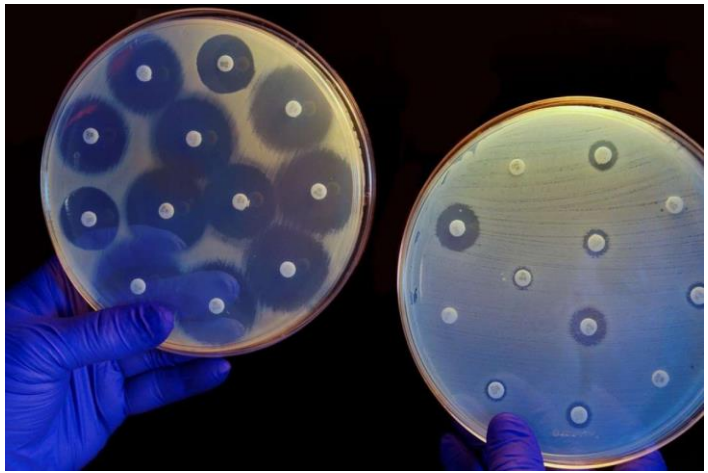
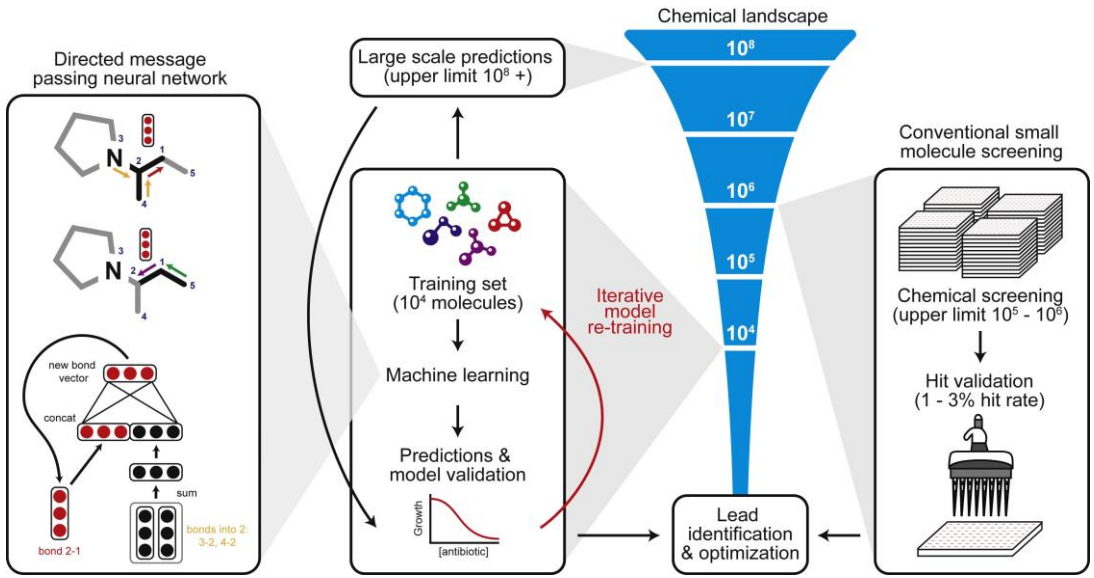


Image credit: CNN

Deep Learning for Antibiotic 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.

USC Viterbi School of Engineering

Home Start Datasets Data Functions Leaderboards HF Models News Team NeurIPS Paper Nat Chem Bio Paper

Therapeutics Data Commons

Artificial intelligence foundation for therapeutic scien

Get Started | Follow us on Twitter | Join us on Slack | Our HuggingFac

Home Start Datasets Data Functions Leaderboards HF Models News Team NeurIPS Paper Nat Chem Bio Paper Docs GitHub

Core Team

Developing and Maintaining Therapeutics Data Commons

Contact us

Kexin Huang
Stanford
Website

Tianfan Fu
Georgia Tech
Website

Wenhao Gao
MIT
Website

Yuanqi Du
Cornell
Website

Ada Fang
Harvard
Website

George Dasoulas
Harvard
Website

Yue Zhao
CMU
Website

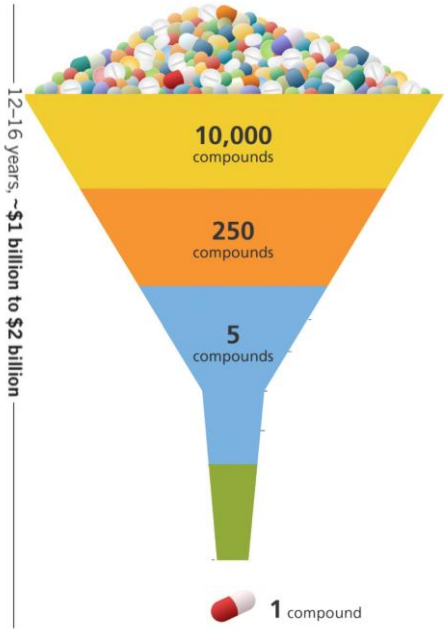
Nitin Pasumarthy
LinkedIn
Website

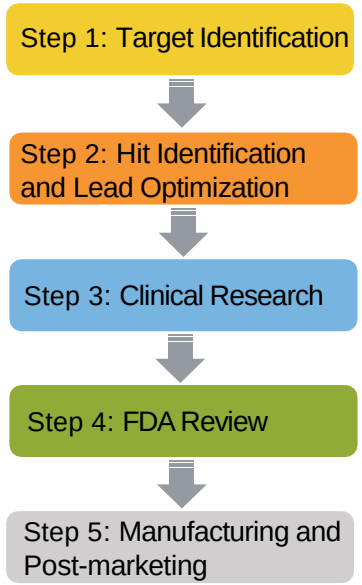
Challenges in Drug Discovery & Development

High cost, long time

Various and emerging diseases

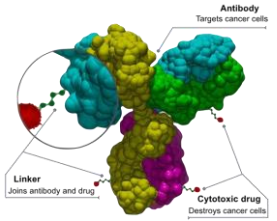
Abundant molecular modality





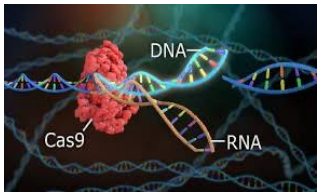
Small molecules

Proteins

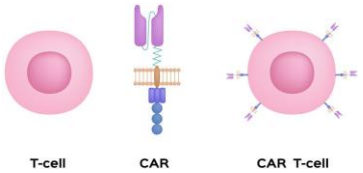


Anti- Nano-bodies

Vaccines



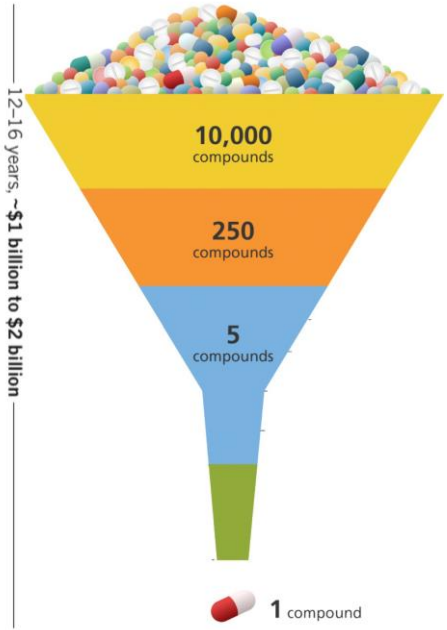
Gene-editing



CAR-T

Challenges of AI in Drug Discovery & Development

- Formulate various tasks as machine-learning-solvable tasks.
- Identify, retrieve, and process datasets of many different types scattered around.
- Assess algorithmic advances to align with real-world and clinical deployment.

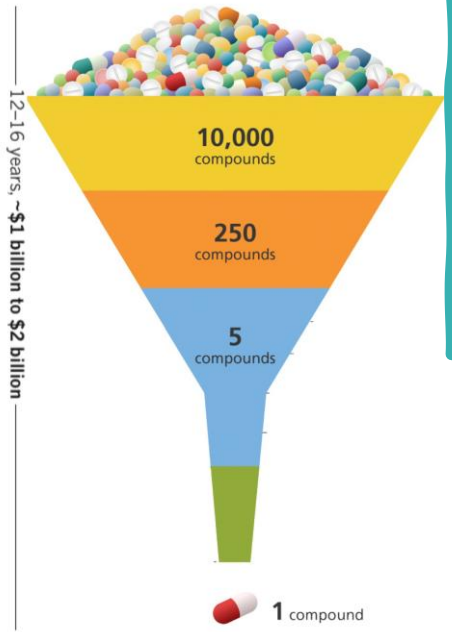


Step 1: Target Identification	Gene-disease/drug interaction prediction; PPI prediction …
Step 2: Hit Identification and Lead Optimization	Molecular optimization; Protein structure prediction; Molecule property prediction …
Step 3: Clinical Research	Clinical trial recruitment; Drug repurposing …
Step 4: FDA Review	Automating document generation, review, and processing …
Step 5: Manufacturing and Post-marketing	Synthesis planning and prediction; Polypharmacy/food-drug prediction…

- Supervised learning
- Semi/self-supervised learning
- Generative modeling
- Optimization
- Reinforcement learning
- Interaction prediction
- Causality inferring
- …

Challenges of AI in Drug Discovery & Development

- Formulate various tasks as machine-learning-solvable tasks.
- Identify, retrieve, and process datasets of many different types scattered around.
- Assess algorithmic advances to align with real-world and clinical deployment.



How to Advance Data-driven Therapeutics Design?

Step 3: Clinical Research

Clinical trial recruitment; Drug

Step 4: FDA Review

Automating document generation, review, and processing …

Step 5: Manufacturing and Post-marketing

Synthesis planning and prediction; Polypharmacy/food-drug prediction…

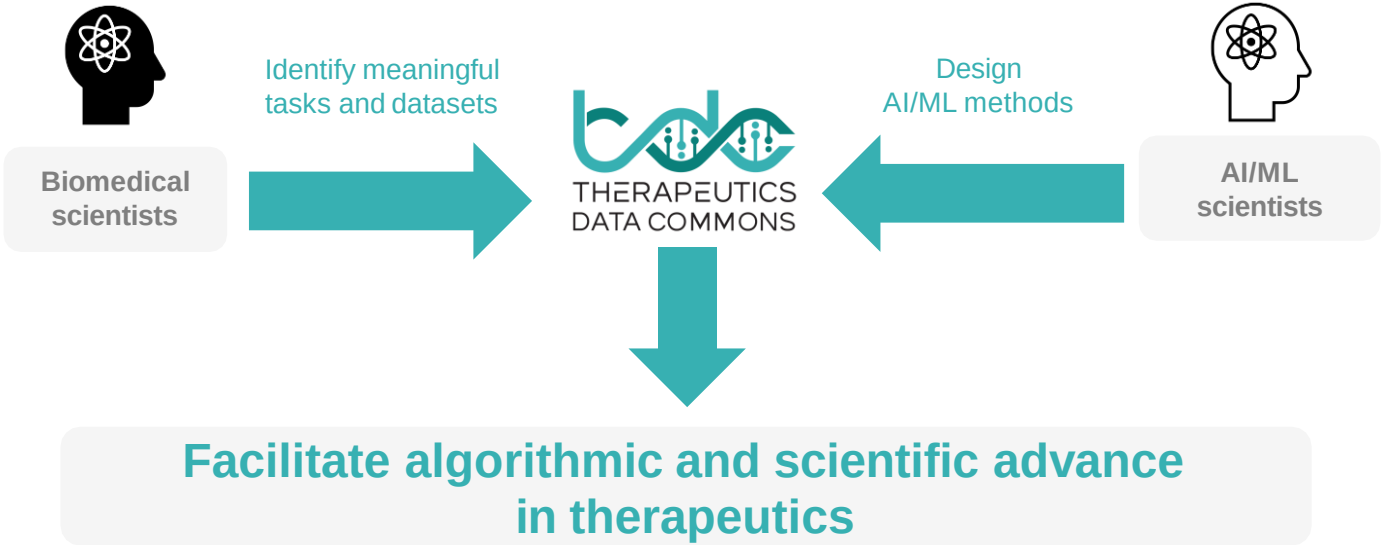
- Supervised learning
- Semi/self-supervised learning
- Generative modeling
- Optimization
- Reinforcement learning
- Interaction prediction
- Causality inferring
- …

Therapeutics Data Commons

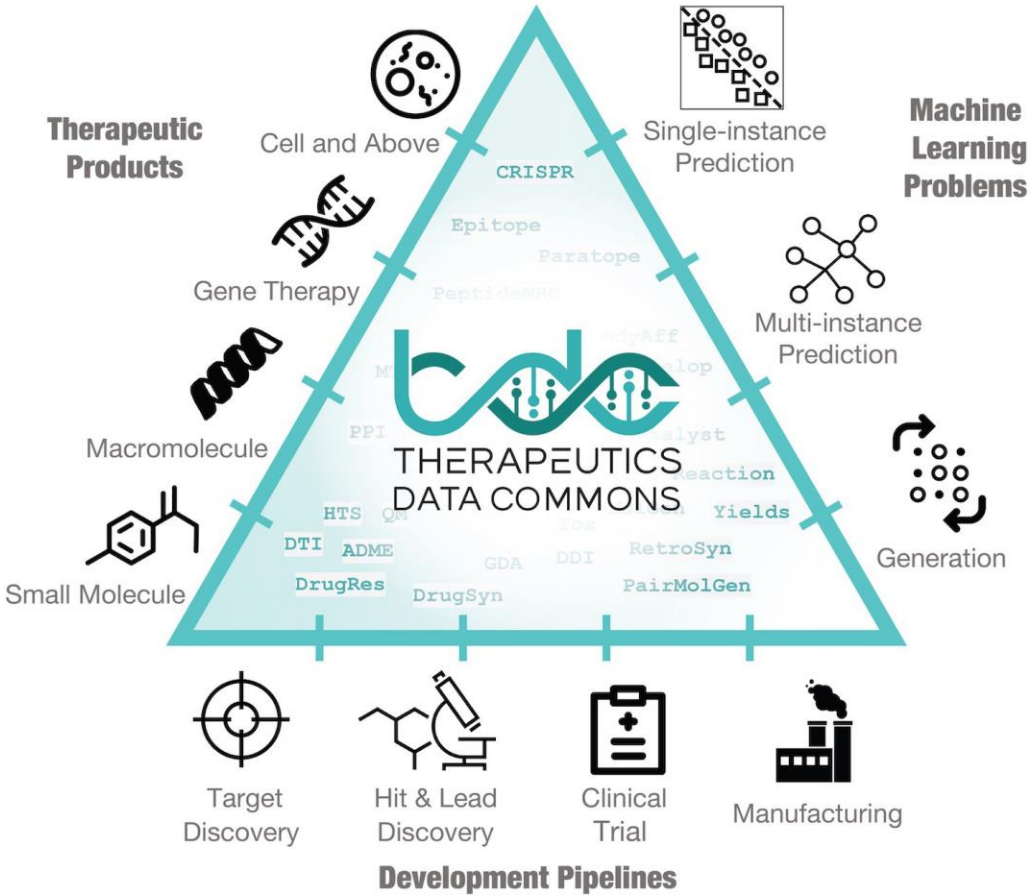
Website: https://tdcommons.ai (or QR code)

Paper: https://www.nature.com/articles/s41589-022-01131-2

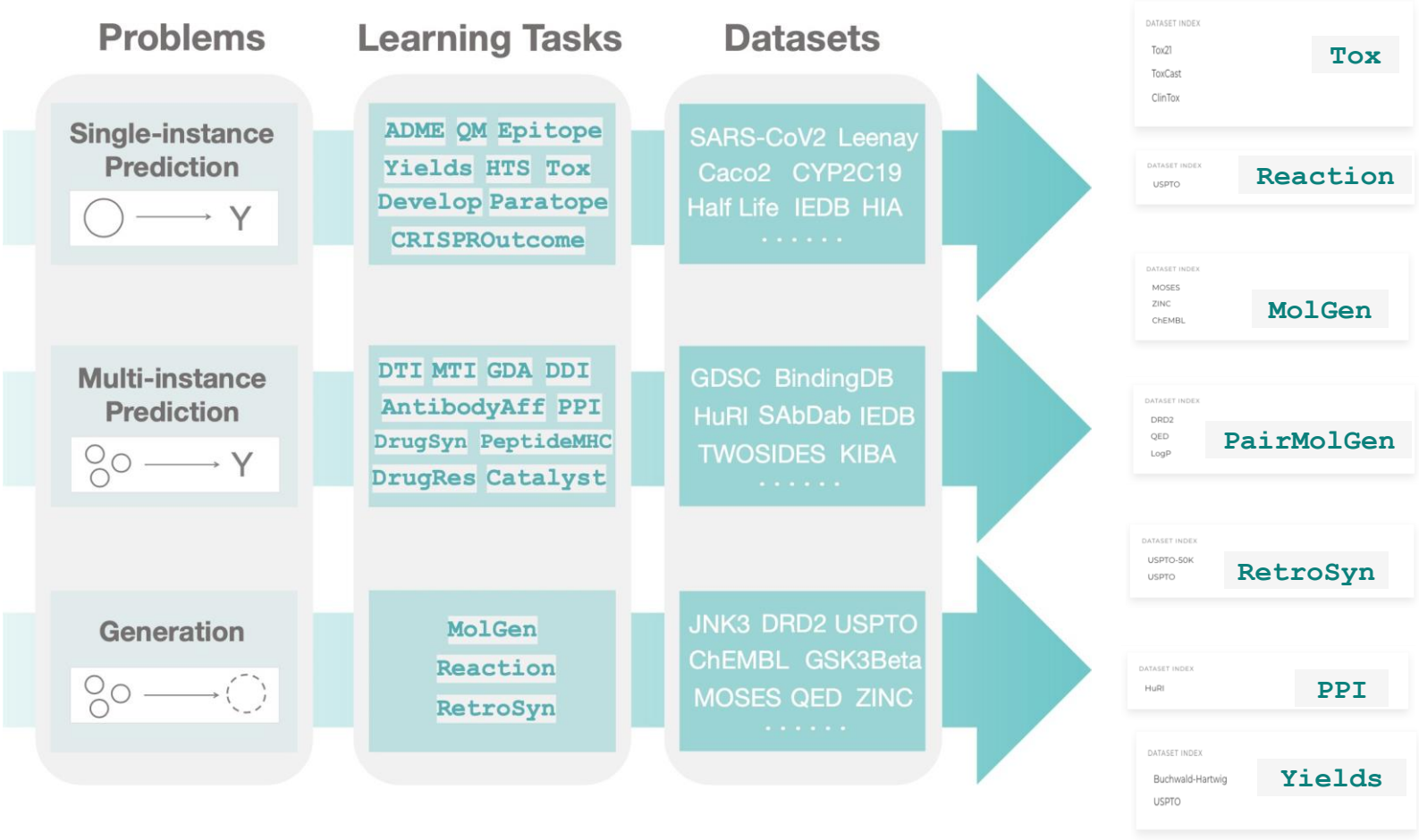
Github: https://github.com/mims-harvard/TDC



Wide Range of Therapeutic Modalities and Pipeline



Three-Tier Design

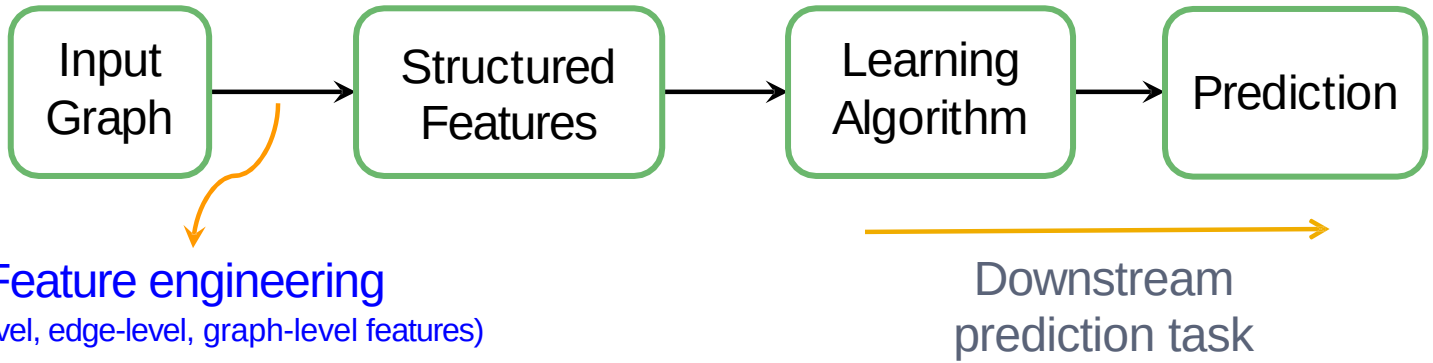


Graph Neural Networks (GNN)

Details

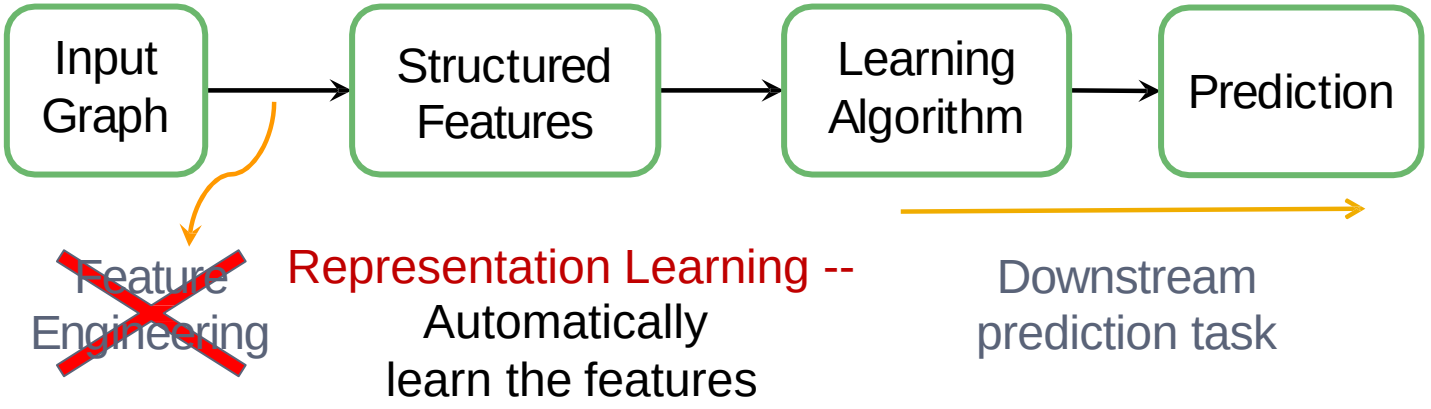
Recap: Traditional ML for Graphs

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

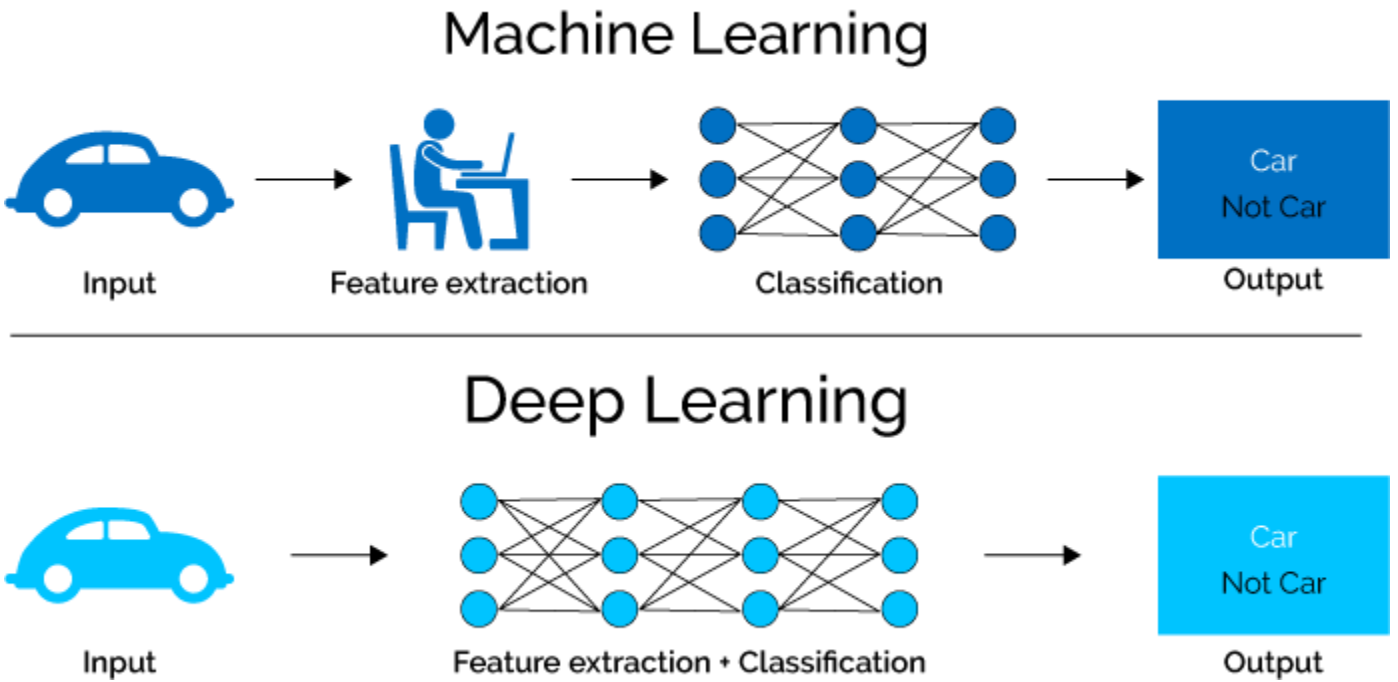


Graph Representation Learning

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

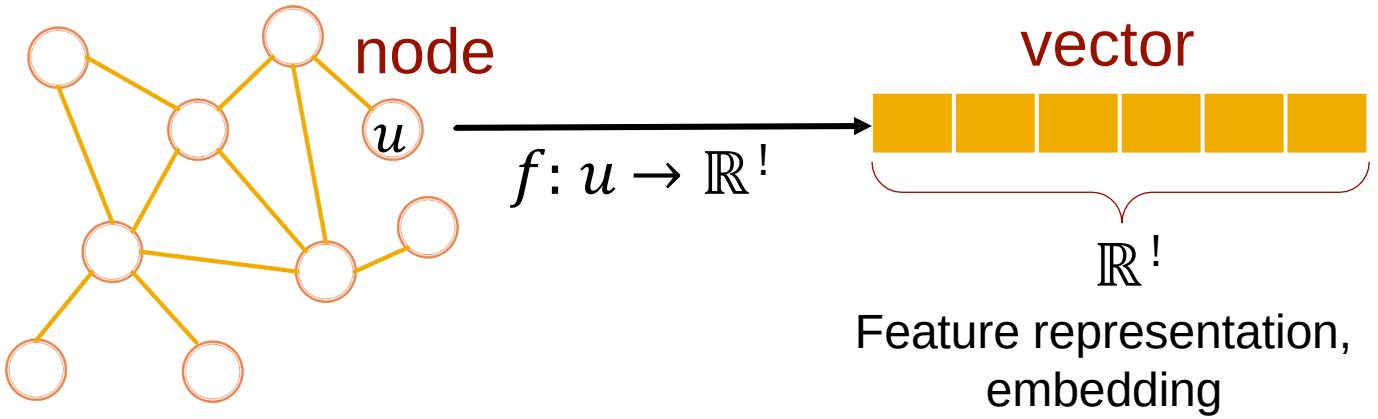


Recap: Classical ML vs. DL (Better Representations)



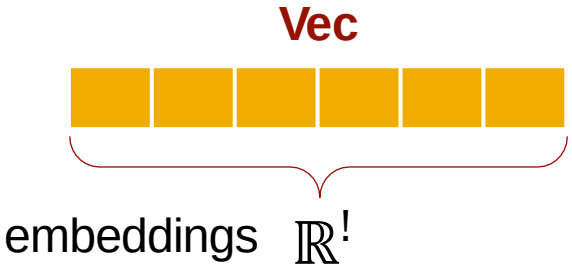
Graph Representation Learning

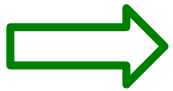
Goal: Efficient task-independent feature learning for machine learning with graphs!



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



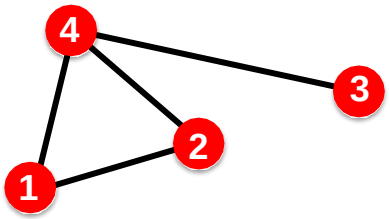


Tasks

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

Setup

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

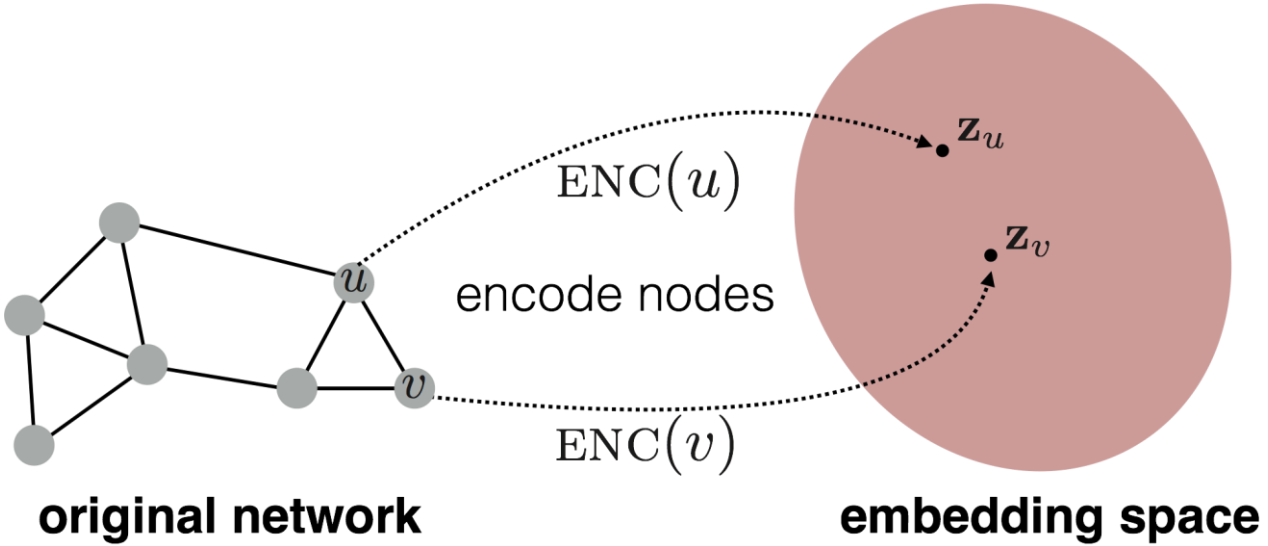


V: {1, 2, 3, 4}

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

Embedding Nodes

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



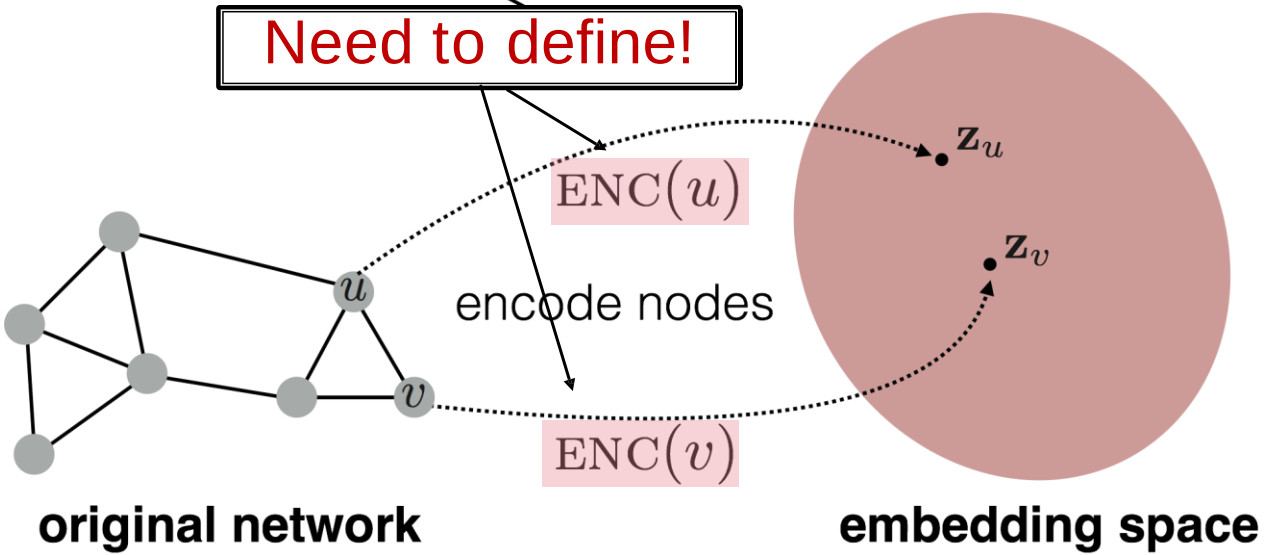
Embedding Nodes

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

in the original network

Similarity of the embedding

Need to define!



Learning Node 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:**

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

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

Two Key Components

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

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

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

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

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

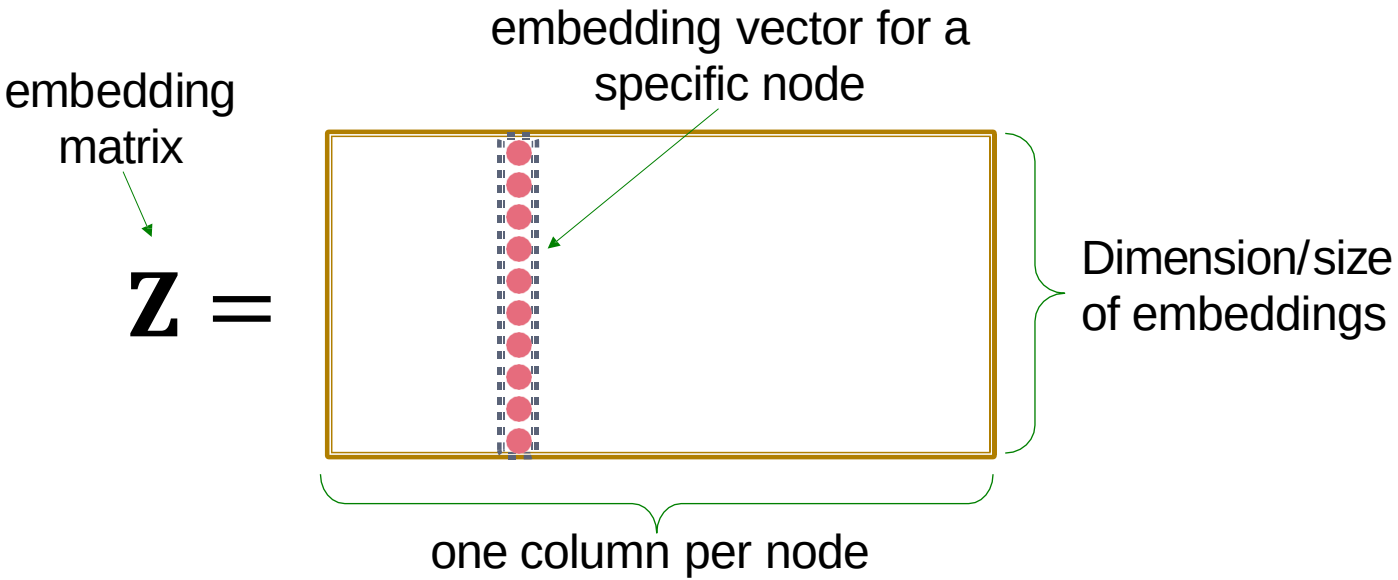
$$\mathrm{ENC}(v) = \mathbf{z}_v = \mathbf{Z} \cdot v$$

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

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 in 20 mins
  - **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”?

Note on Node Embeddings

- 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 is preserved.
- These embeddings are **task independent:**
  - They are not trained for a specific task but can be used for any task.

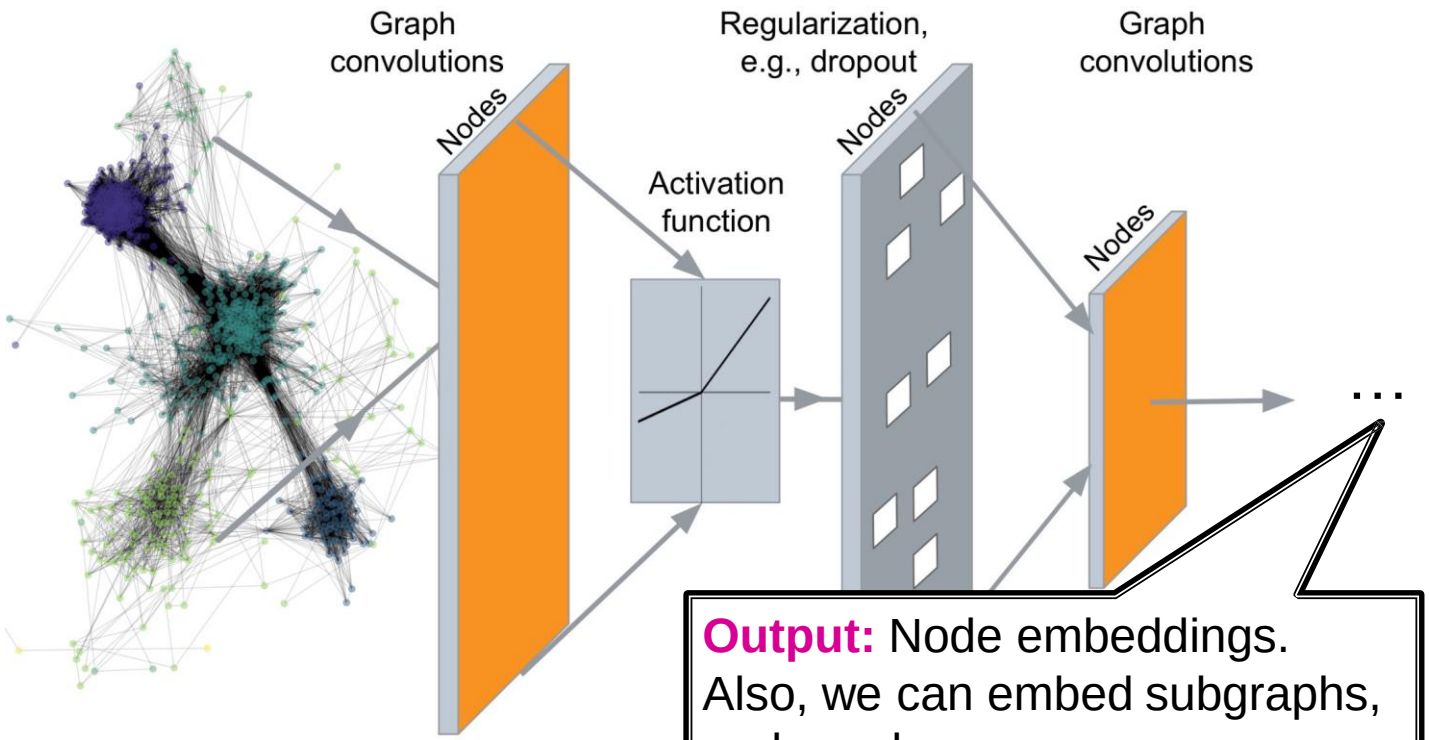
Today: Deep Graph Encoders

- **Now:** discuss deep learnig methods based on **graph neural networks (GNNs):**

$$\mathrm{ENC}(v) = \text{multiple layers of non-linear transformations based on graph structure}$$

- **Note:** All these deep encoders can be **combined with node similarity functions** defined earlier.

Deep Graph Encoders



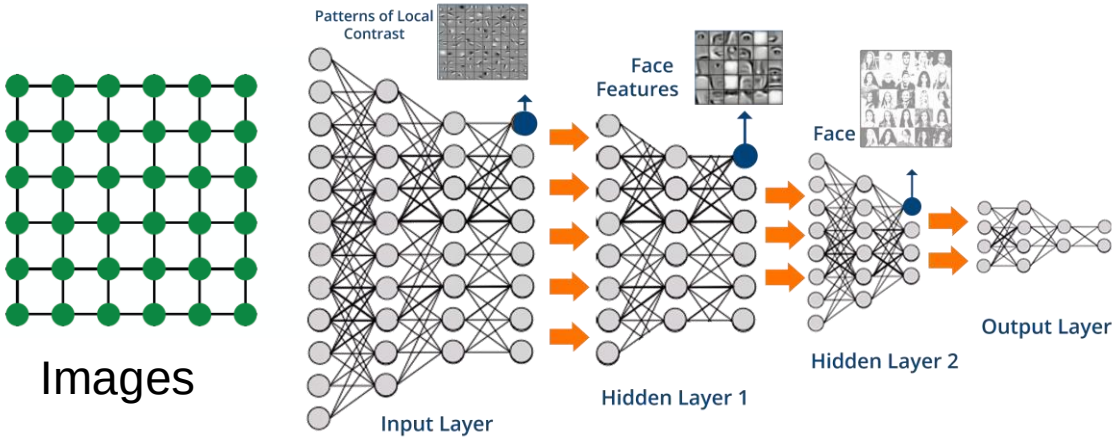
Output: Node embeddings. Also, we can embed subgraphs, and graphs

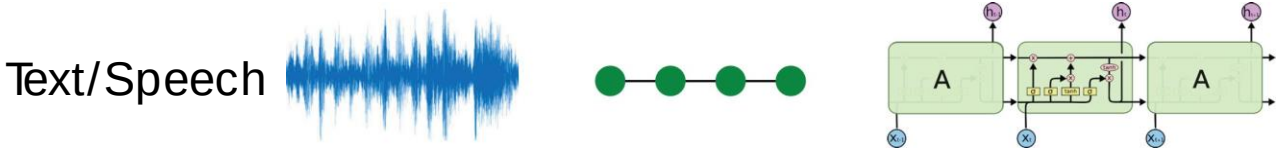
Tasks on Networks

Tasks we will be able to solve:

- Node classification
  - Predict the type of a given node
- Link prediction
  - Predict whether two nodes are linked
- Community detection
  - Identify densely linked clusters of nodes
- Network similarity
  - How similar are two (sub)networks

Modern ML Toolbox



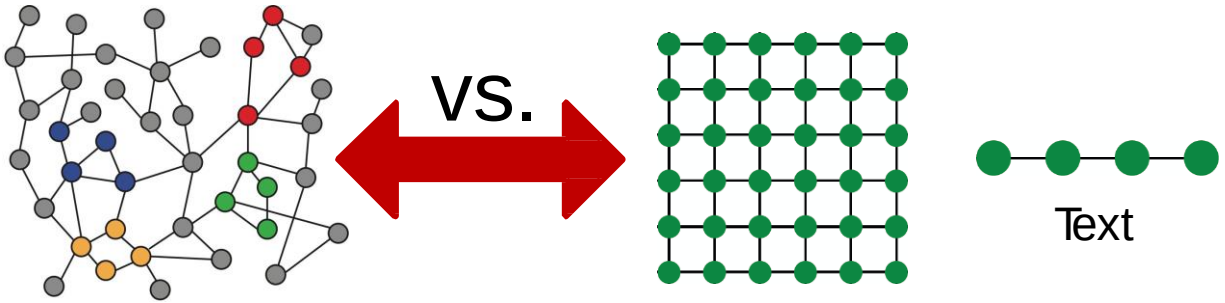


Modern deep learning toolbox is designed for simple sequences & grids

Why is it Hard?

But networks are far more 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

Summary: Basics of Deep Learning

- **Loss function:** $\min_{\Theta} \mathcal{L}(\boldsymbol{y}, f_{\Theta}(\boldsymbol{x}))$
- f can be a simple linear layer, an MLP, or other neural networks (e.g., a GNN later)
- Sample a minibatch of input $\boldsymbol{x}$
- **Forward propagation:** Compute $\mathcal{L}$ given $\boldsymbol{x}$
- **Back-propagation:** Obtain gradient $\nabla_{\Theta}\mathcal{L}$ using a chain rule.
- Use **stochastic gradient descent (SGD)** to optimize $\mathcal{L}$ for Θ over many iterations.

Content

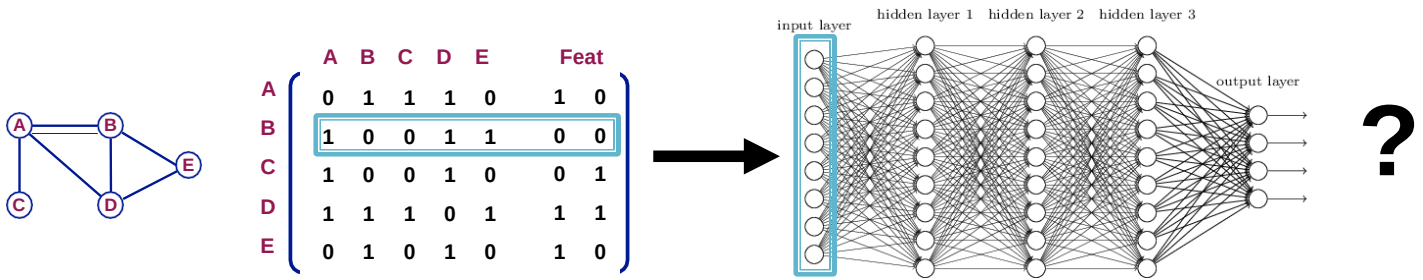
- **Local network neighborhoods:**
  - Describe aggregation strategies
  - Define computation graphs
- **Stacking multiple layers:**
  - Describe the model, parameters, training
  - How to fit the model?
  - Simple example for unsupervised and supervised training

Setup

- **Assume we have a graph G:**
  - V is the **vertex set**
  - $\boldsymbol{A}$ is the **adjacency matrix** (assume binary)
  - $\boldsymbol{X} \in \mathbb{R}^{|V| \times m}$ is a matrix of **node features**
  - v: a node in V; $N(v)$: the set of neighbors of v.
  - **Node features:**
    - Social networks: User profile, User image
    - Biological networks: Gene expression profiles, gene functional information
    - When there is no node feature in the graph dataset:
      - Indicator vectors (one-hot encoding of a node)
      - Vector of constant 1: [1, 1, …, 1]

A Naïve Approach

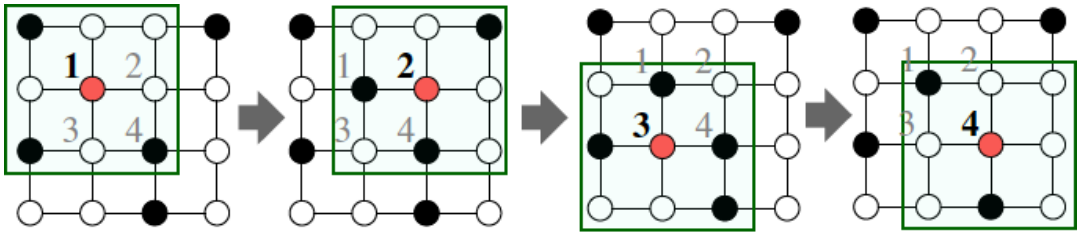
- Join adjacency matrix and features
- Feed them into a deep neural net:

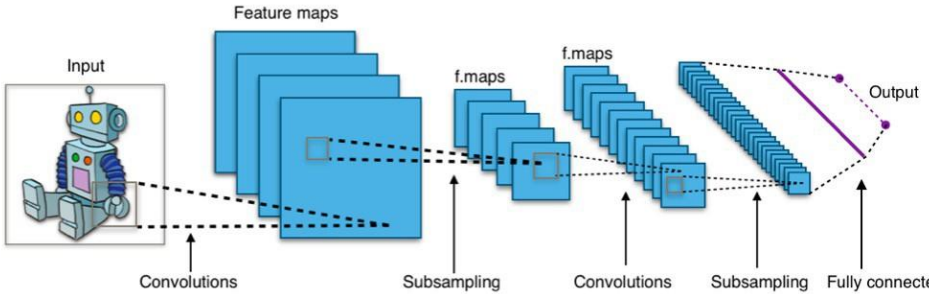


- Issues with this idea:
  - $O(|V|)$ parameters
  - Not applicable to graphs of different sizes
  - Sensitive to node ordering

Idea: Convolutional Networks

CNN on an image:



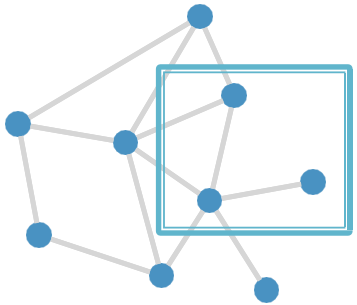


Goal is to generalize convolutions beyond simple lattices

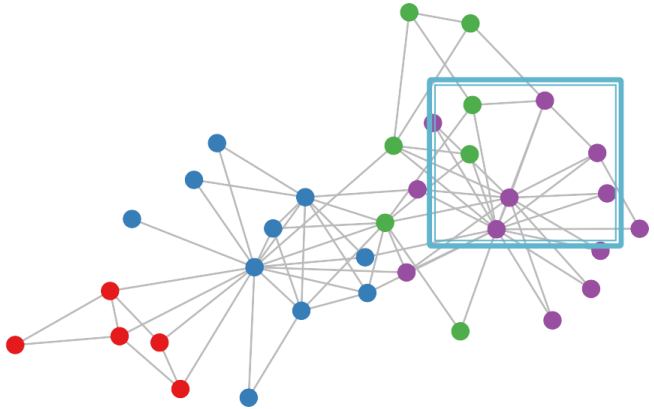
Leverage node features/attributes (e.g., text, images)

Real-World Graphs

But our graphs look like this:



or this:



- There is no fixed notion of locality or sliding window on the graph
- Graph is permutation invariant

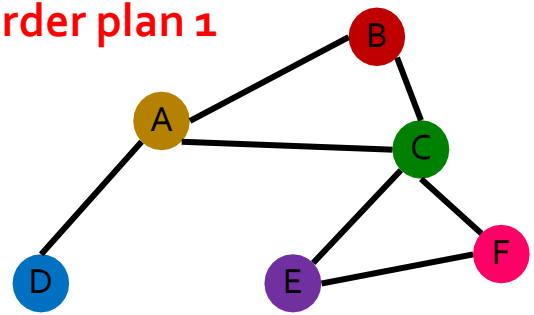
Permutation Invariance

- **Graph does not have a canonical order of the nodes!**
- We can have many different order plans.

Permutation Invariance

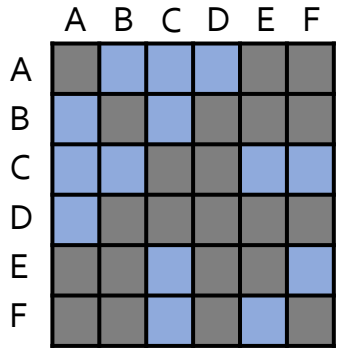
- **Graph does not have a canonical order of the nodes!**

Order plan 1



Node features X_1

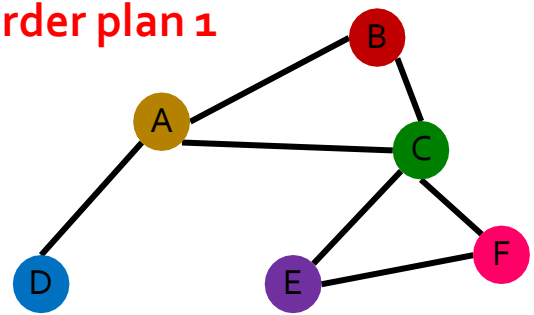
Adjacency matrix A_1



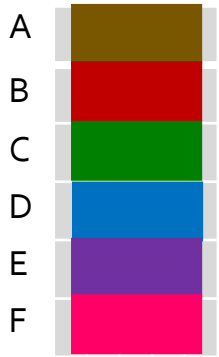
Permutation Invariance

- **Graph does not have a canonical order of the nodes!**

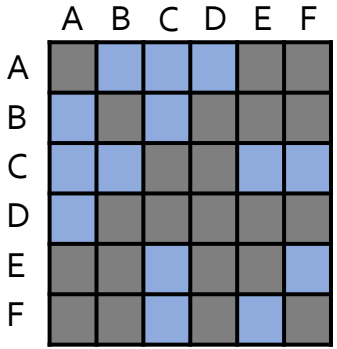
Order plan 1



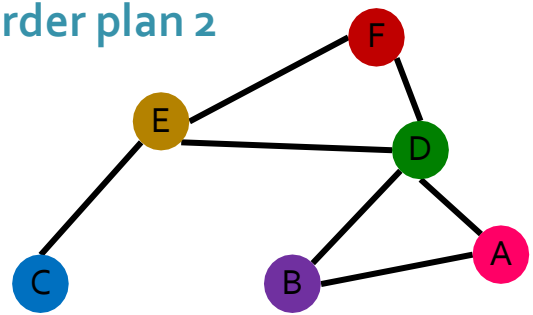
Node features X_1



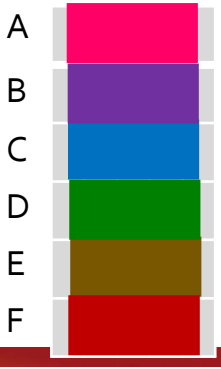
Adjacency matrix A_1



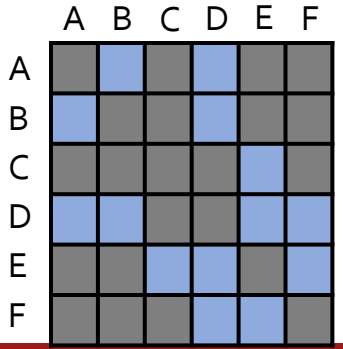
Order plan 2



Node features X_2

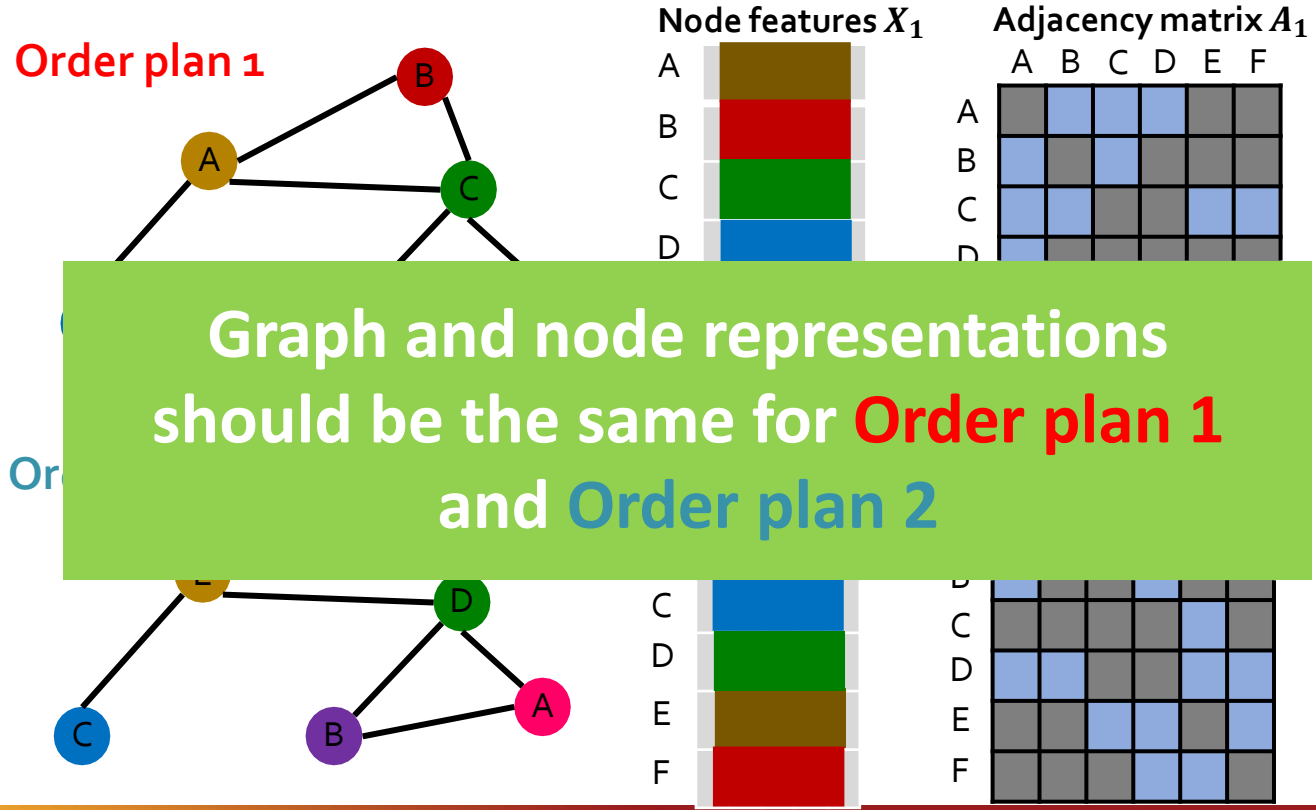


Adjacency matrix A_2



Permutation Invariance

- **Graph does not have a canonical order of the nodes!**



Graph and node representations should be the same for Order plan 1 and Order plan 2

Permutation Invariance

What does it mean by "graph representation is same for two order plans"?

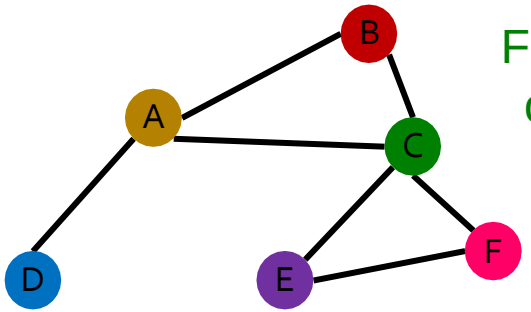
In other words, f maps a graph to a d-dim embedding

- Consider we learn a function f that maps a graph $G = (\boldsymbol{A}, \boldsymbol{X})$ to a vector $\mathbb{R}^{\mathrm{d}}$ then

$$f(\boldsymbol{A}_1, \boldsymbol{X}_1) = f(\boldsymbol{A}_2, \boldsymbol{X}_2)$$

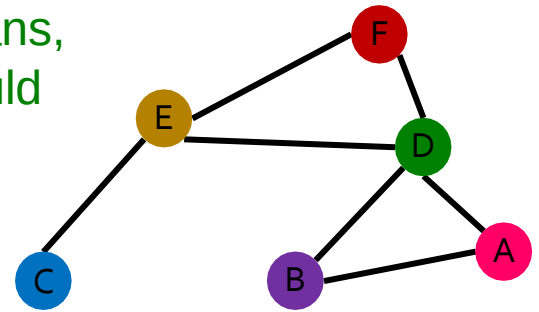
$\boldsymbol{A}$ is the adjacency matrix
$\boldsymbol{X}$ is the node feature matrix

Order plan 1: $\boldsymbol{A}_1, \boldsymbol{X}_1$



For two order plans, output of f should be the same!

Order plan 2: $\boldsymbol{A}_2, \boldsymbol{X}_2$



Permutation Invariance

What does it mean by "graph representation is same for two order plans"?

- Consider we learn a function f that maps a graph $G = (\boldsymbol{A}, \boldsymbol{X})$ to a vector $\mathbb{R}^d$.

  $\boldsymbol{A}$ is the adjacency matrix
  $\boldsymbol{X}$ is the node feature matrix

- Then, if $f(\boldsymbol{A}_i, \boldsymbol{X}_i) = f(\boldsymbol{A}_j, \boldsymbol{X}_j)$ for any order plan i and j, we formally say f is a **permutation invariant function**.

  For a graph with $|V|$ nodes, there are $|V|!$ different order plans.

  m… each node has a m-dim feature vector associated with it.

- **Definition:** For any graph function $f: \mathbb{R}^{|V| \times m} \times \mathbb{R}^{|V| \times |V|} \rightarrow \mathbb{R}^d$, f is **permutation-invariant** if $f(A, X) = f(PAP^T, PX)$ for any permutation P.

  d… output embedding dimensionality of embedding the graph $G = (A, X)$

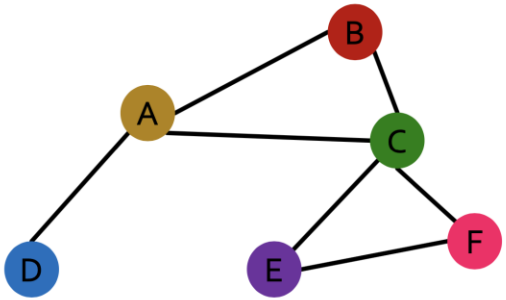
  Permutation P: a shuffle of the node order
  Example: (A,B,C)->(B,C,A)

Permutation Equivariance

For node representation: We learn a function f that maps nodes of G to a matrix $\mathbb{R}^{|V|\times d}$.

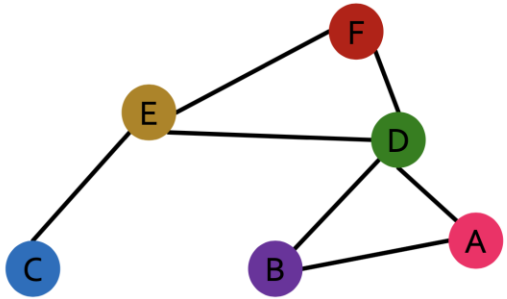
In other words, each node in V is mapped to a d-dim embedding.

Order plan 1: A_1, X_1



$$f(\boldsymbol{A}_1, \boldsymbol{X}_1) =$$

Order plan 2: A_2, X_2

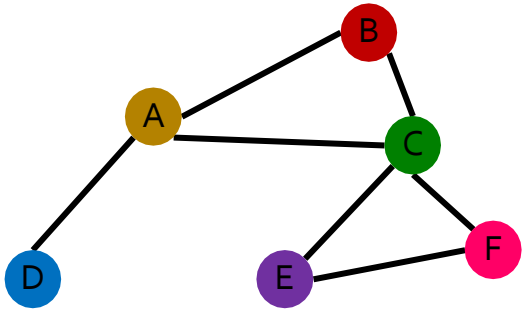


$$f(\boldsymbol{A}_2, \boldsymbol{X}_2) =$$

Permutation Equivariance

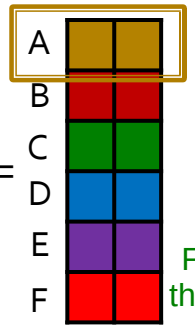
For node representation: We learn a function f that maps nodes of G to a matrix $\mathbb{R}^{|v| \times d}$.

Order plan 1: A_1, X_1

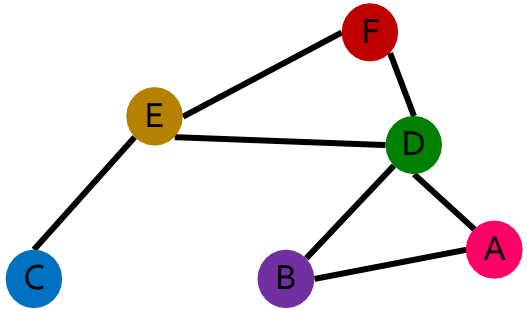


Representation vector of the brown node A

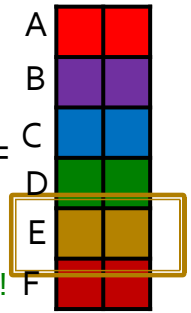
$f(A_1, X_1) =$



Order plan 2: A_2, X_2



$f(A_2, X_2) =$



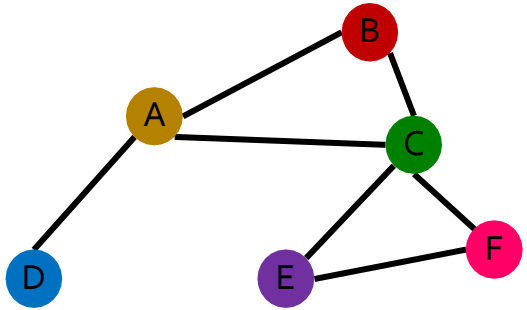
Representation vector of the brown node E

For two order plans, the vector of node at the same position in the graph is the same!

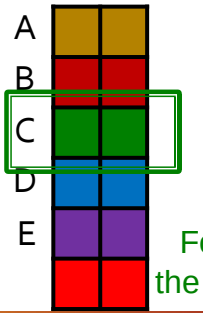
Permutation Equivariance

For node representation: We learn a function f that maps nodes of G to a matrix $\mathbb{R}^{|V|\times d}$.

Order plan 1: A_1, X_1

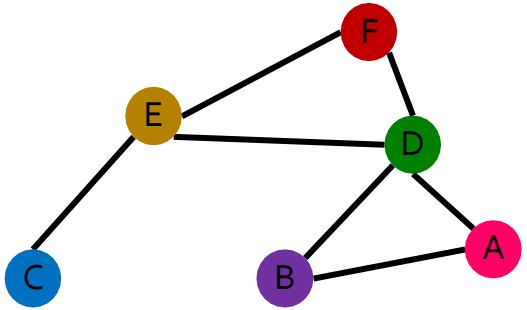


$f(A_1, X_1) =$

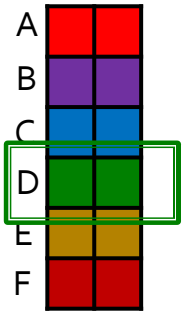


Representation vector of the green node C

Order plan 2: A_2, X_2



$f(A_2, X_2) =$



Representation vector of the green node D

For two order plans, the vector of node at the same position in the graph is the same!

Permutation Equivariance

For node representation:

- Consider we learn a function f that maps a graph $G = (\boldsymbol{A}, \boldsymbol{X})$ to a matrix $\mathbb{R}^{|V| \times d}$
- If the output vector of a node at the same position in the graph remains unchanged for any order plan, we say f is **permutation equivariant**.

m… each node has a m-dim feature vector associated with it.

- **Definition:** For any node function $f: \mathbb{R}^{|V| \times m} \times \mathbb{R}^{|V| \times |V|} \rightarrow \mathbb{R}^{|V| \times d}$, f is **permutation-equivariant** if $Pf(A, X) = f(PAP^T, PX)$ for any permutation P.

f maps each node in V to a d-dim embedding.

Summary: Invariance and Equivariance

- **Permutation-invariant**

$$f(A, X) = f(PAP^T, PX)$$

Permute the input, the output stays the same. (map a graph to a vector)

graph level

- **Permutation-equivariant**

$$Pf(A, X) = f(PAP^T, PX)$$

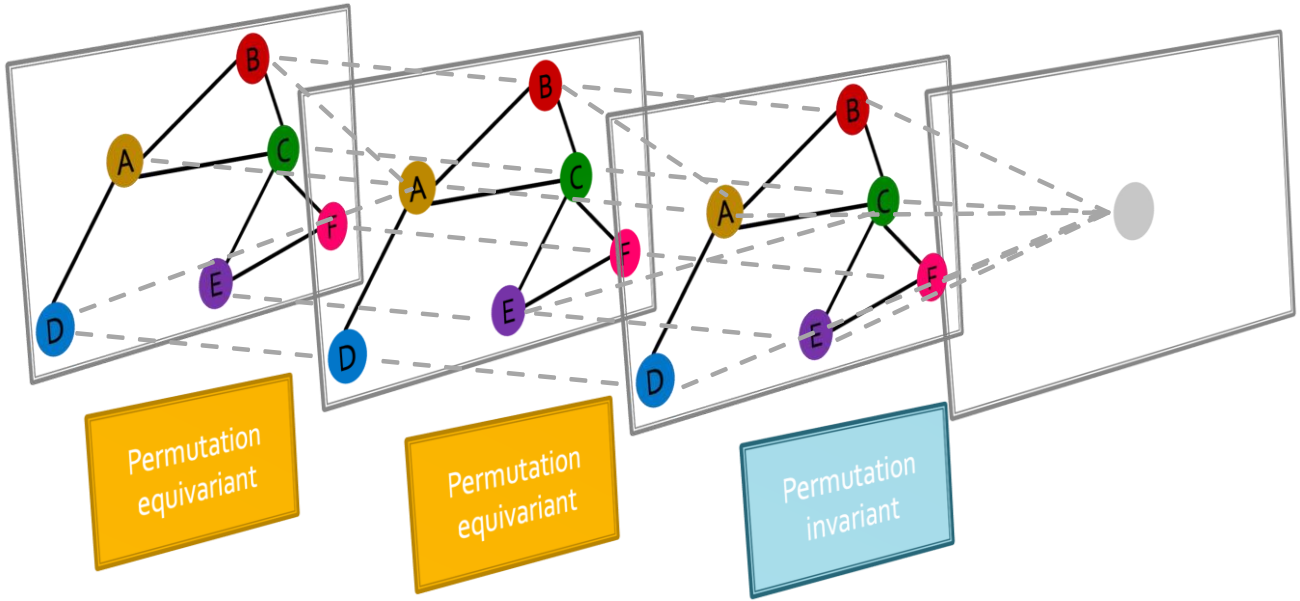
Permute the input, output also permutes accordingly. (map a graph to a matrix)

node level

- **Examples:**
  - $f(A, X) = 1^T X$: Permutation-**invariant**
    - Reason: $f(PAP^T, PX) = 1^T PX = 1^T X = f(A, X)$
  - $f(A, X) = X$: Permutation-**equivariant**
    - Reason: $f(PAP^T, PX) = PX = Pf(A, X)$
  - $f(A, X) = AX$: Permutation-**equivariant**
    - Reason: $f(PAP^T, PX) = PAP^T PX = PAX = Pf(A, X)$

Graph Neural Network Overview

- **Graph neural networks consist of multiple permutation equivariant / invariant functions.**

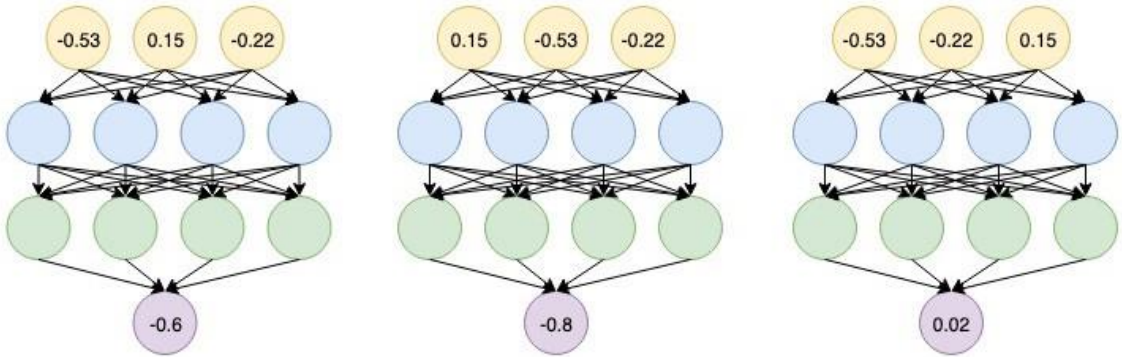


Graph Neural Network Overview

Are other neural network architectures, e.g., MLPs, permutation invariant / equivariant?

- **No.**

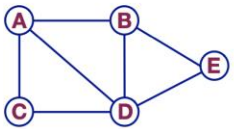
Switching the order of the input leads to different outputs!



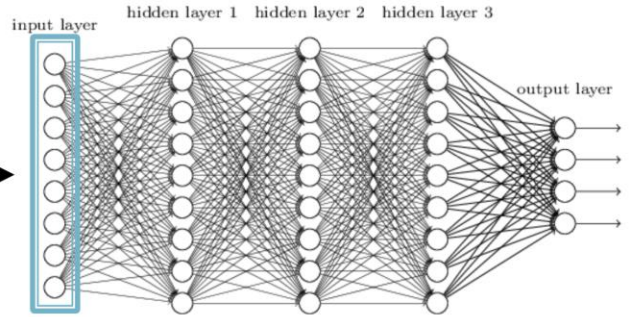
Graph Neural Network Overview

Are other neural network architectures, e.g., MLPs, permutation invariant / equivariant?

- **No.**



	A	B	C	D	E	Feat	
A	0	1	1	1	0	1	0
B	1	0	0	1	1	0	0
C	1	0	0	1	0	0	1
D	1	1	1	0	1	1	1
E	0	1	0	1	0	1	0



?

This explains why **the naïve MLP approach fails for graphs**!

Graph Neural Network Overview

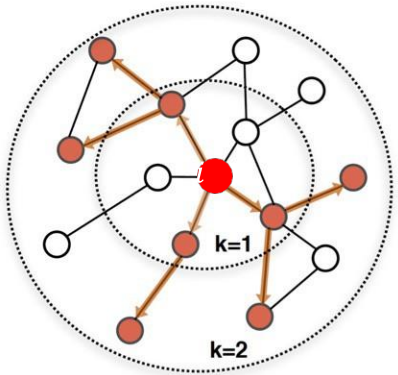
- **Are any neural network architectures, e.g.,**

Next: Design graph neural networks that are permutation invariant / equivariant by passing and aggregating information from neighbors!

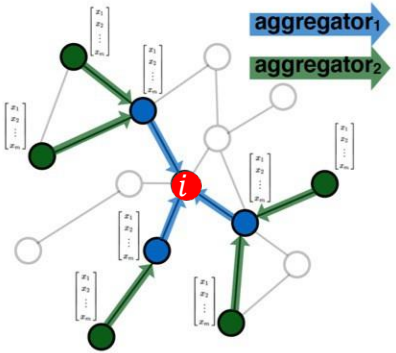
?

Graph Convolutional Networks

Idea: Node's neighborhood defines a computation graph



Determine node computation graph

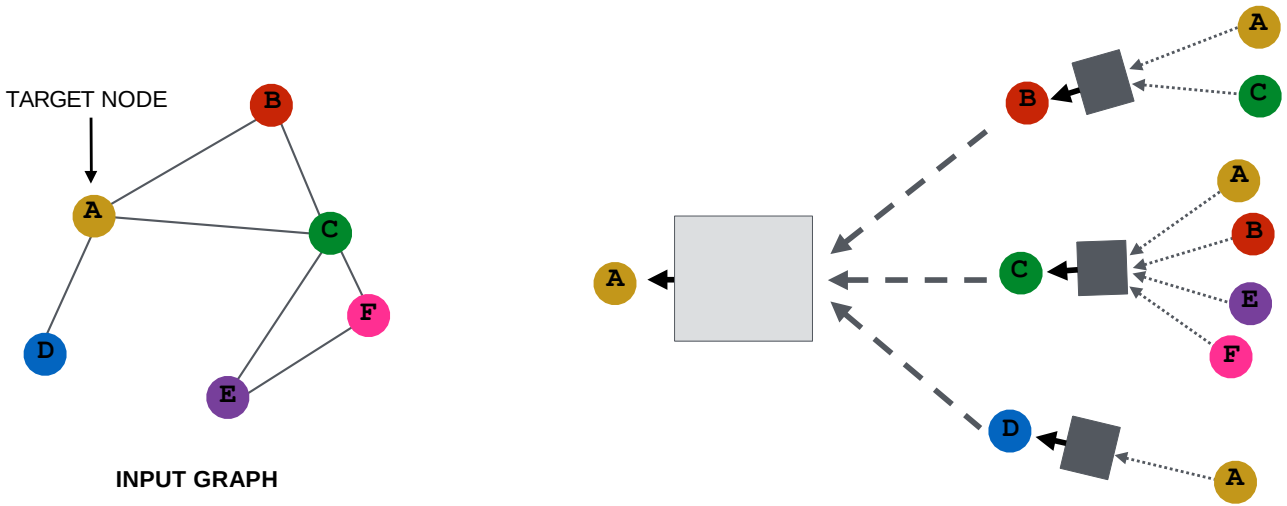


Propagate and transform information

Learn how to propagate information across the graph to compute node features

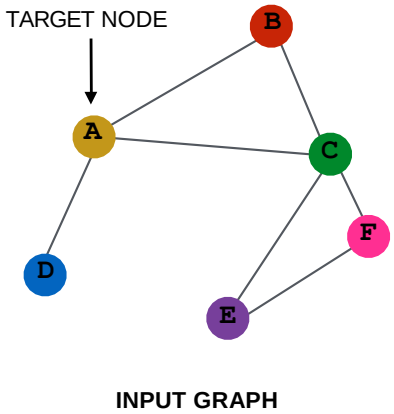
Idea: Aggregate Neighbors

- **Key idea:** Generate node embeddings based on **local network neighborhoods**

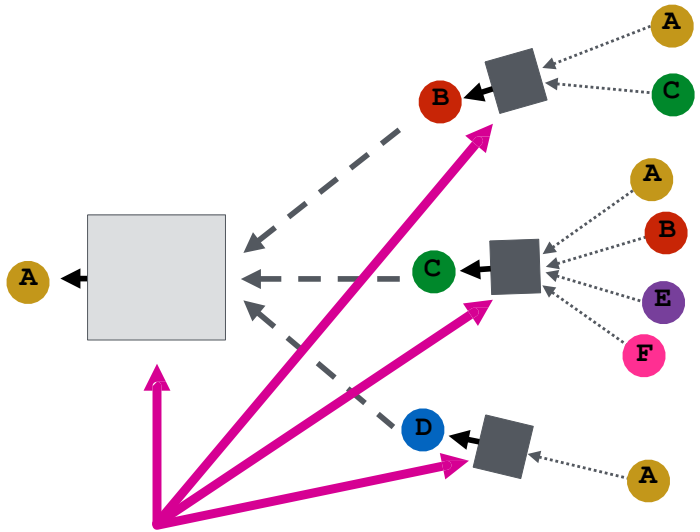


Idea: Aggregate Neighbors

- **Intuition:** Nodes aggregate information from their neighbors using neural networks



INPUT GRAPH

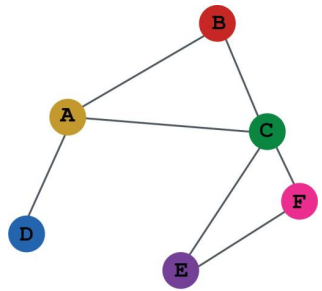


Neural networks

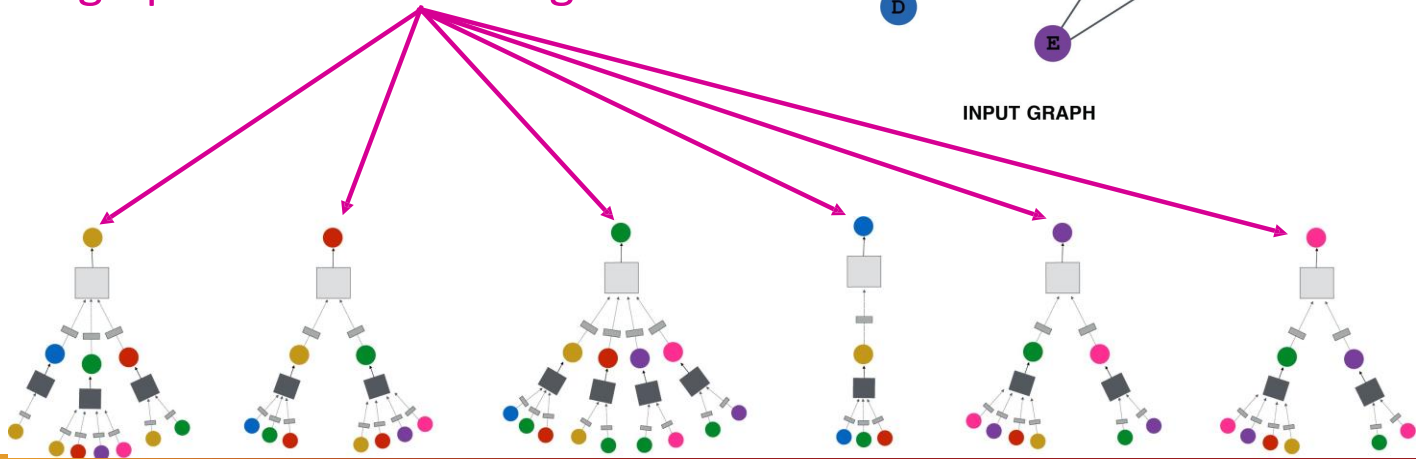
Idea: Aggregate Neighbors

- **Intuition:** Network neighborhood defines a computation graph

Every node defines a computation graph based on its neighborhood!

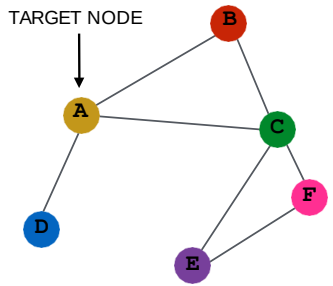


INPUT GRAPH

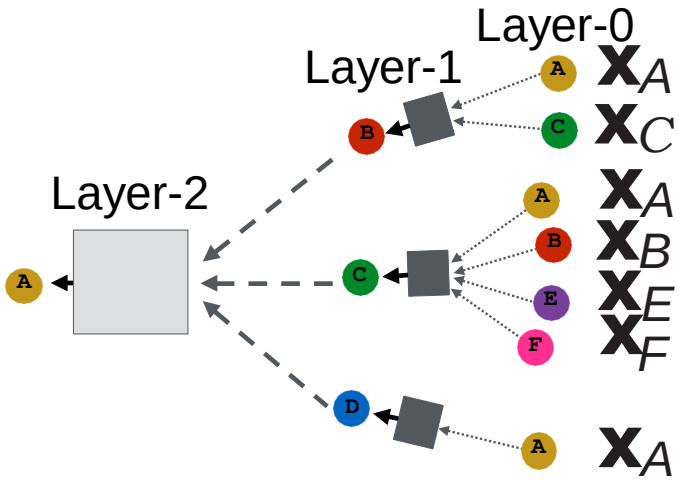


Deep Model: Many Layers

- Model can be of arbitrary depth:
  - Nodes have embeddings at each layer
  - Layer-0 embedding of node v is its input feature, x_v
  - Layer-k embedding gets information from nodes that are k hops away

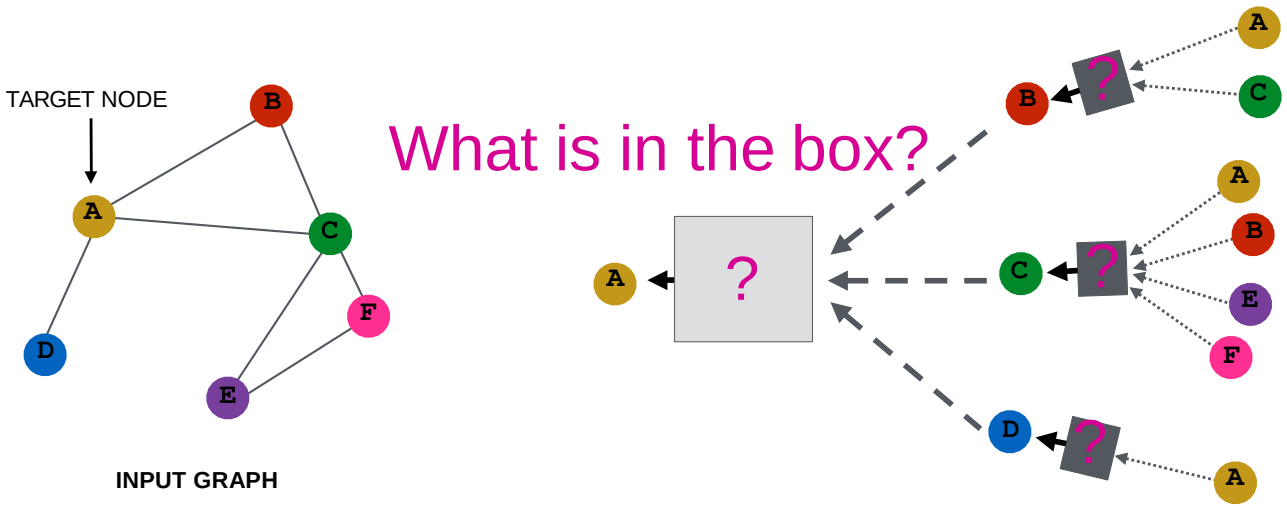


INPUT GRAPH



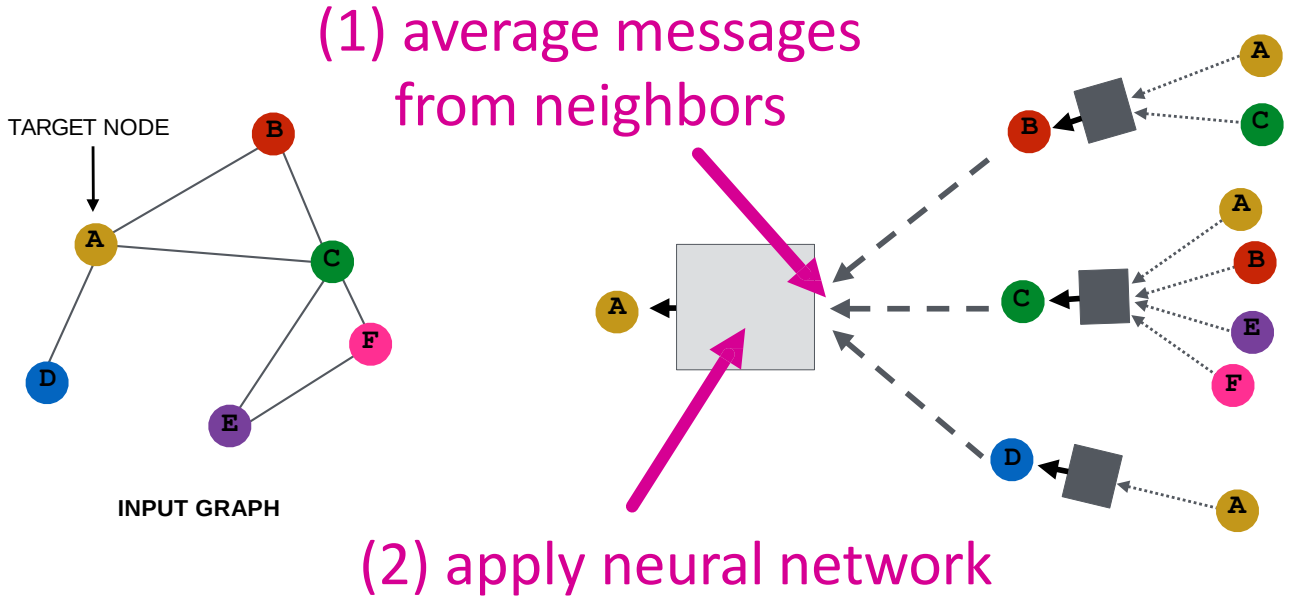
Neighborhood Aggregation

- **Neighborhood aggregation:** Key distinctions are in how different approaches aggregate information across the layers



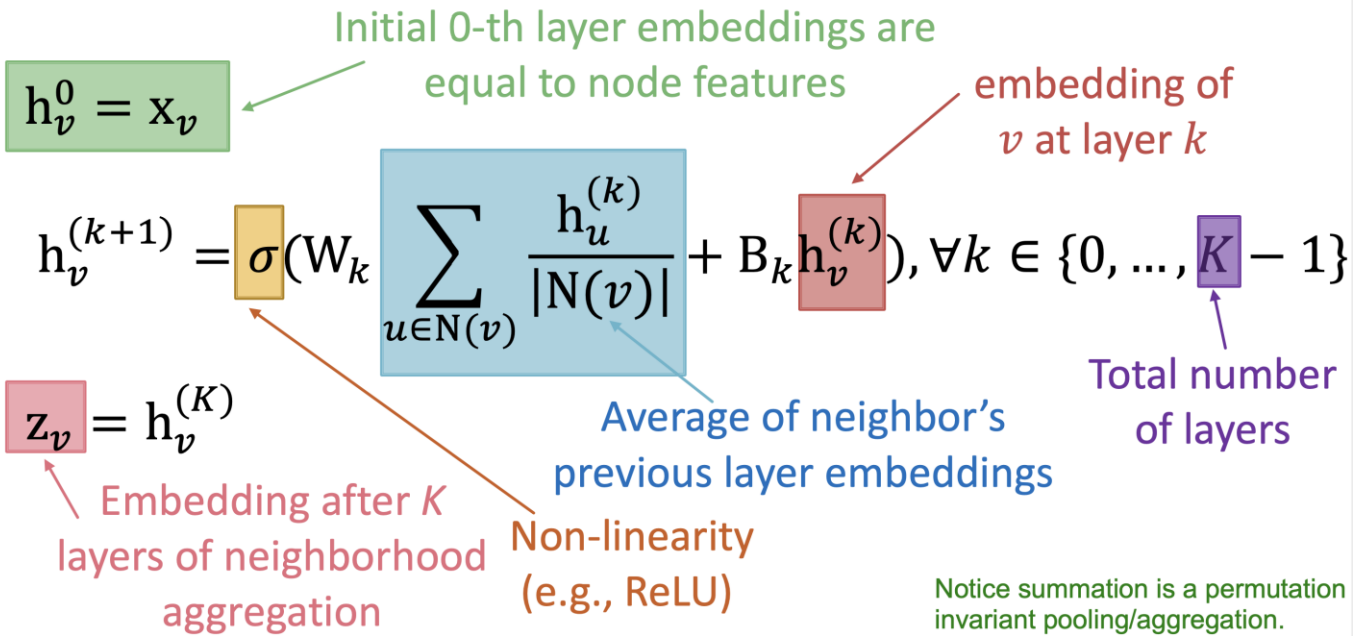
Neighborhood Aggregation

- **Basic approach:** Average information from neighbors and apply a neural network



The Math: Deep Encoder

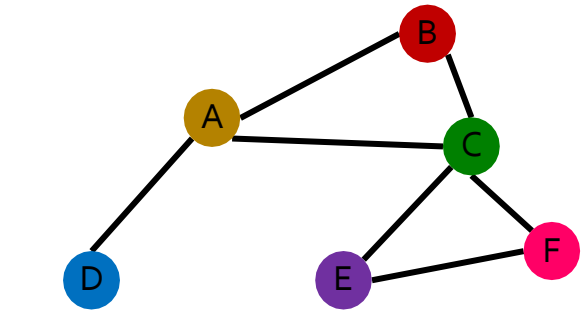
- **Basic approach:** Average neighbor messages and apply a neural network



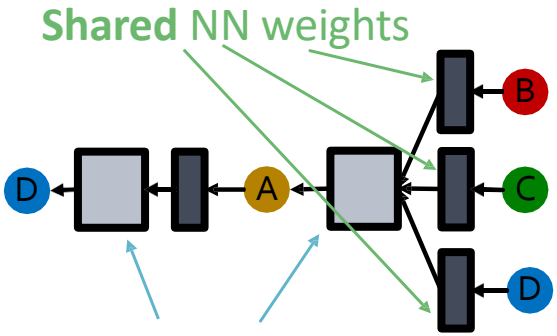
GCN: Invariance and Equivariance

What are the invariance and equivariance properties for a GCN?

- **Given a node**, the GCN that computes its embedding is **permutation invariant**



Target Node

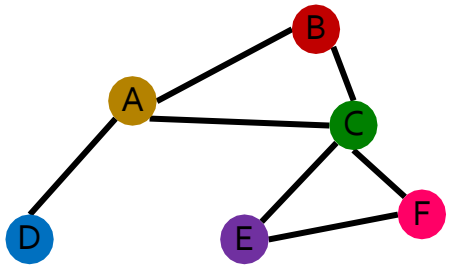


Average of neighbor's previous layer embeddings - **Permutation invariant**

GCN: Invariance and Equivariance

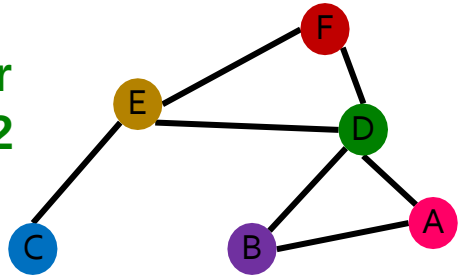
- **Considering all nodes in a graph**, GCN computation is **permutation equivariant**

Order plan 1

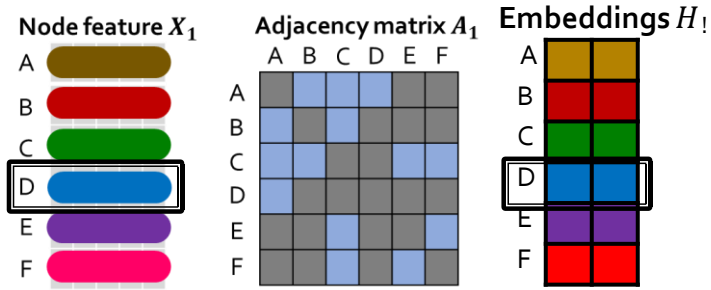


Target Node

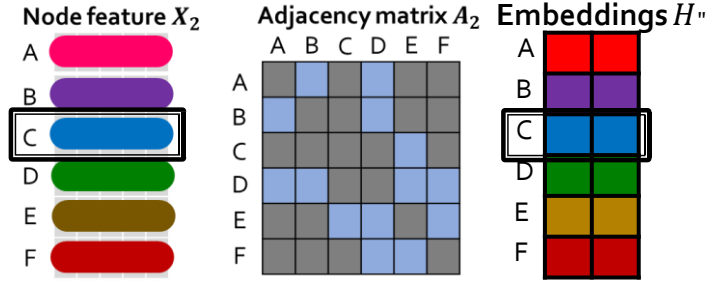
Order plan 2



Target Node



Permute the input, the output also permutes accordingly - permutation equivariant



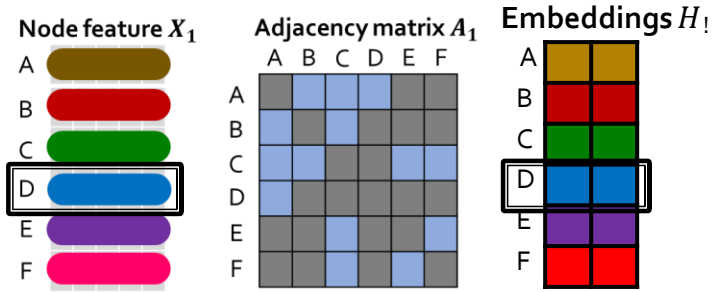
GCN: Invariance and Equivariance

- **Considering all nodes in a graph**, GCN computation is **permutation equivariant**

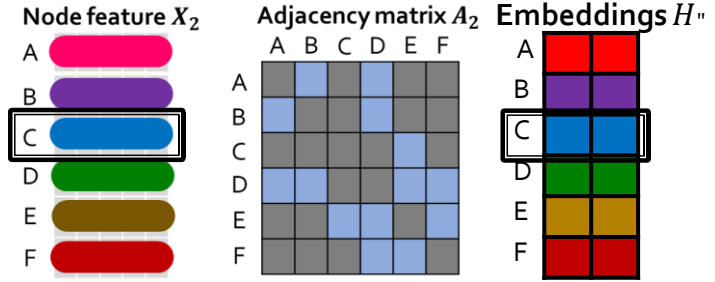
Detailed reasoning:

1. The rows of **input node features** and **output embeddings** are **aligned**
2. We know computing the embedding of **a given node** with GCN is **invariant.**
3. So, after permutation, the **location** of **a given node** in the **input node feature** matrix is changed, and the **the output embedding of a given node stays the same** (the colors of node feature and embedding are **matched**)

This is permutation equivariant

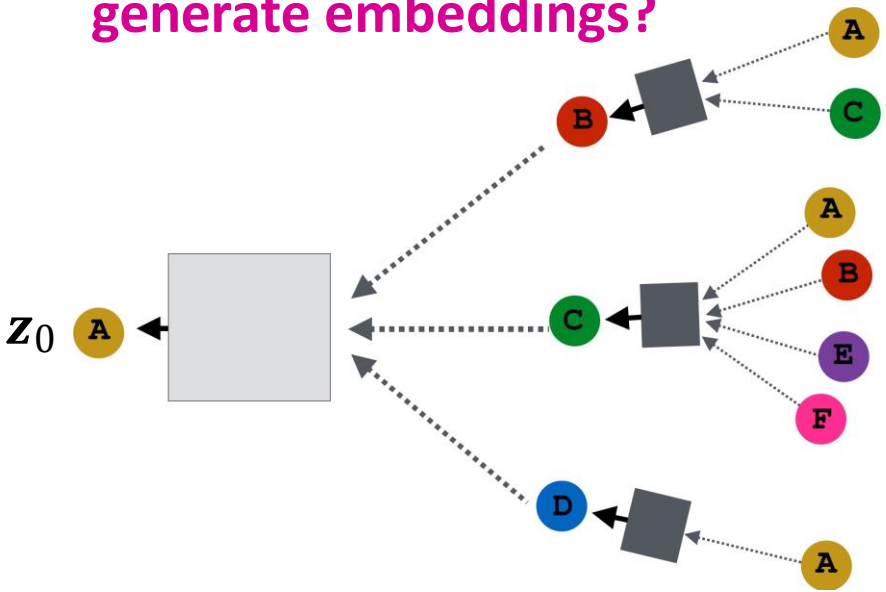


Permute the input, the output also permutes accordingly - permutation equivariant



Training the Model

How do we train the GCN to generate embeddings?



Need to define a loss function on the embeddings.

Model Parameters

Trainable weight matrices
(i.e., what we learn)

$$h_v^{(0)} = x_v$$

$$h_v^{(k+1)} = \sigma\left(W_k \sum_{u \in N(v)} \frac{h_u^{(k)}}{|N(v)|} + B_k h_v^{(k)}\right), \forall k \in \{0..K-1\}$$

$$z_v = h_v^{(K)}$$

Final node embedding

We can feed these **embeddings into any loss function** and run SGD to **train the weight parameters**

h_v^k: the hidden representation of node v at layer k

- W_k: weight matrix for neighborhood aggregation
- B_k: weight matrix for transforming hidden vector of self

How to Train A GNN

- Node embedding $\boldsymbol{z}_0$ is a function of input graph
- **Supervised setting**: We want to minimize loss $\mathcal{L}$:

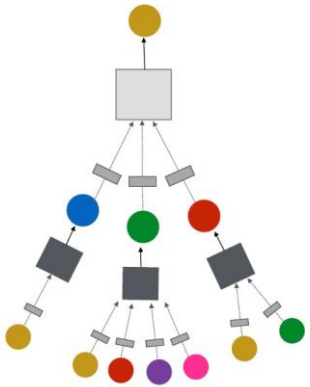
$$\min_{\Theta} \mathcal{L}(\boldsymbol{y}, f_{\Theta}(\boldsymbol{z}_v))$$

  - $\boldsymbol{y}$: node label
  - $\mathcal{L}$ could be L2 if $\boldsymbol{y}$ is real number, or cross entropy if $\boldsymbol{y}$ is categorical
- **Unsupervised setting:**
  - No node label available
  - **Use the graph structure as the supervision!**

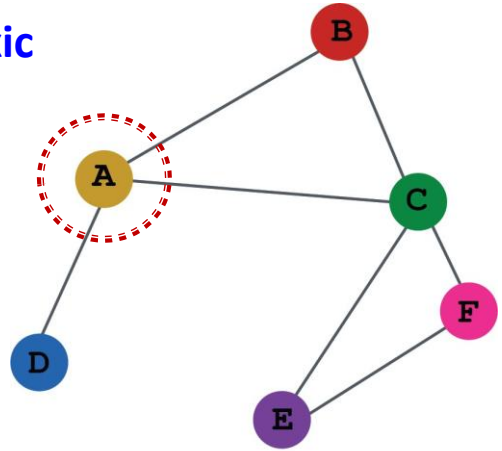
Supervised Training

Directly train the model for a supervised task (e.g., **node classification**)

Safe or toxic drug?



Safe or toxic drug?



E.g., a drug-drug interaction network

Supervised Training

Directly train the model for a supervised task (e.g., **node classification**)

- Use cross entropy loss

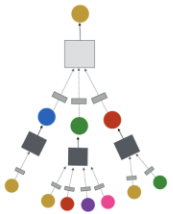
$$\mathcal{L} = -\sum_{v \in V} y_v \log(\sigma(z_v^{\mathrm{T}} \theta)) + (1 - y_v) \log(1 - \sigma(z_v^{\mathrm{T}} \theta))$$

Encoder output: node embedding

Classification weights

Node class label

Safe or toxic drug?



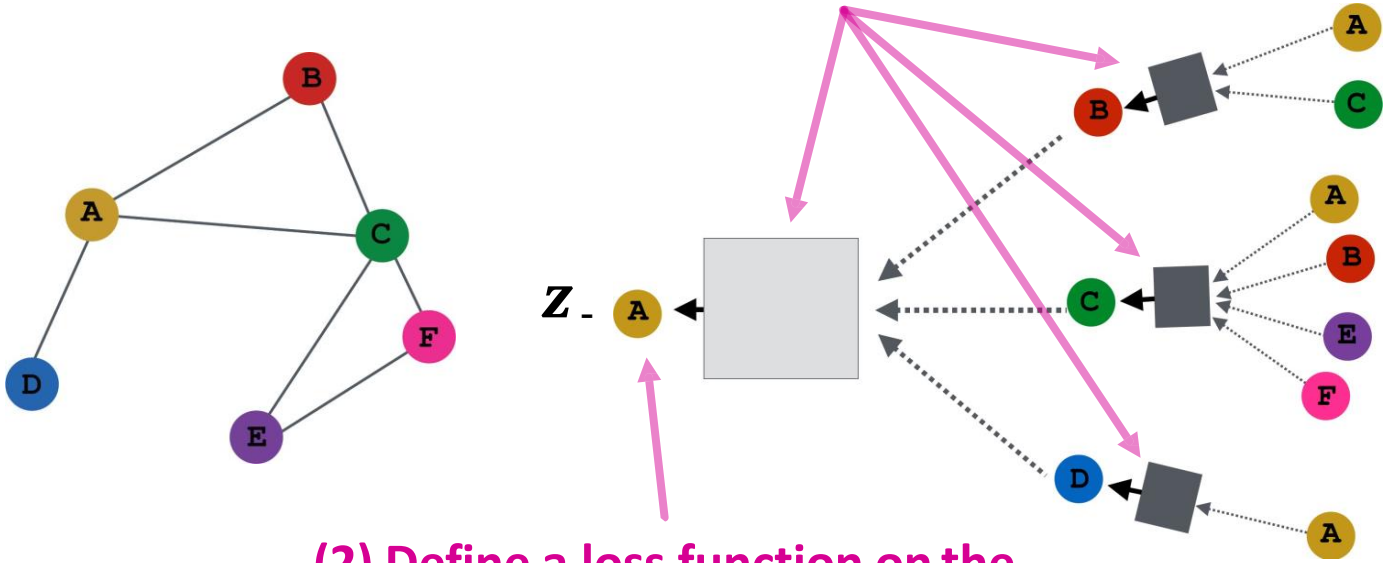
True probability distribution (one-shot)

$$H(p, q) = -\sum_{x \in classes} p(x) \log q(x)$$

Your model's predicted probability distribution

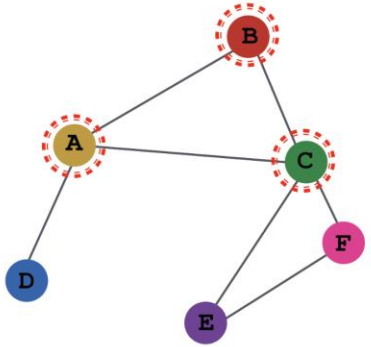
Model Design: Overview

(1) Define a neighborhood aggregation function



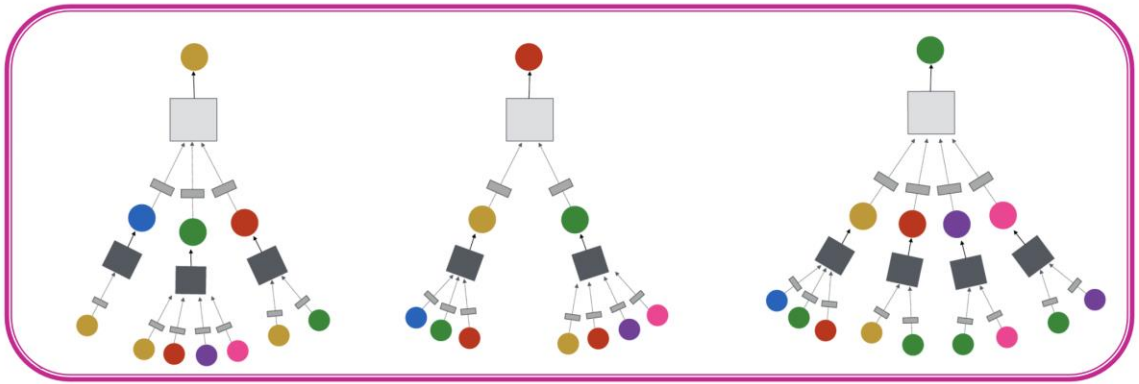
(2) Define a loss function on the embeddings

Model Design: Overview

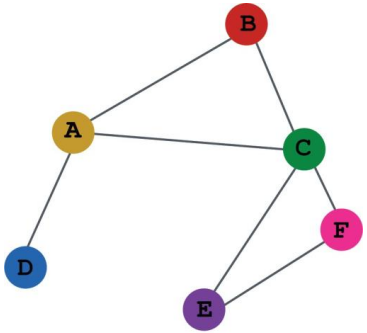


INPUT GRAPH

(3) Train on a set of nodes, i.e., a batch of compute graphs



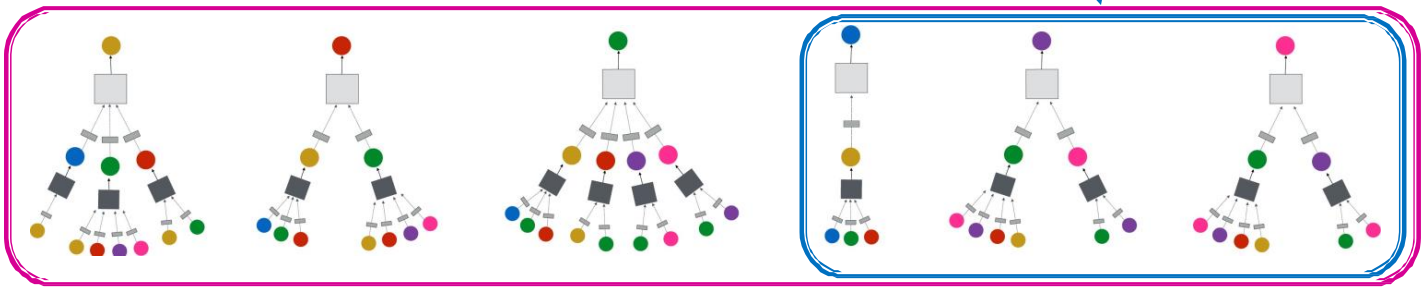
Model Design: Overview



INPUT GRAPH

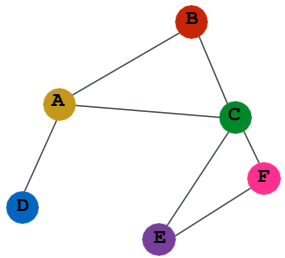
(4) Generate embeddings for nodes as needed

Even for nodes we never trained on!

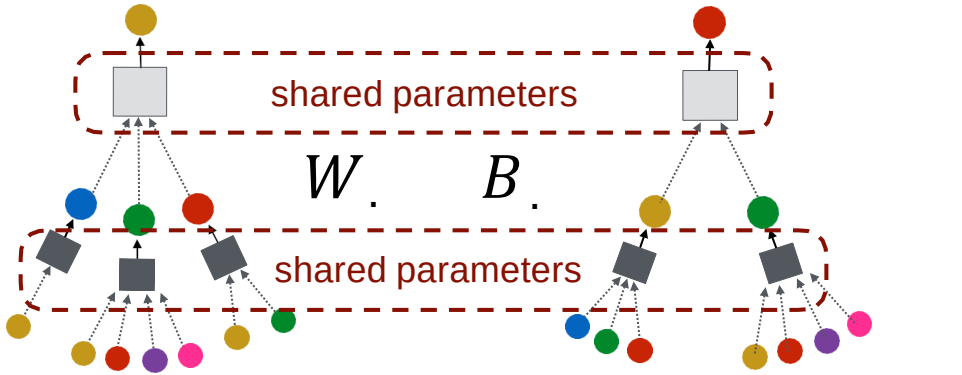


Inductive Capability

- **The same aggregation parameters are shared for all nodes:**
  - The number of model parameters is sublinear in $|V|$ and we can **generalize to unseen nodes**!



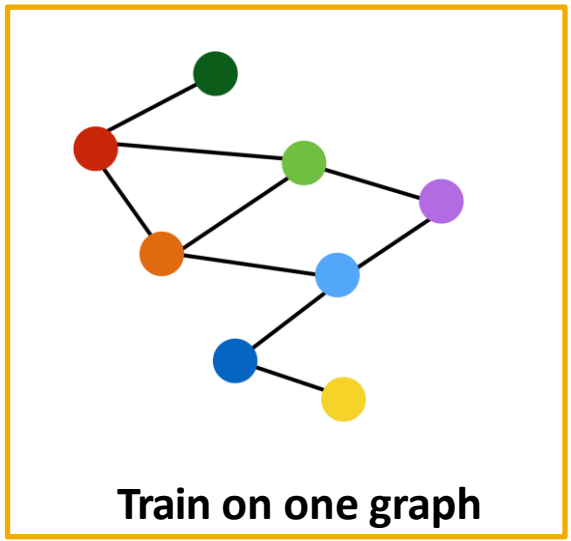
INPUT GRAPH

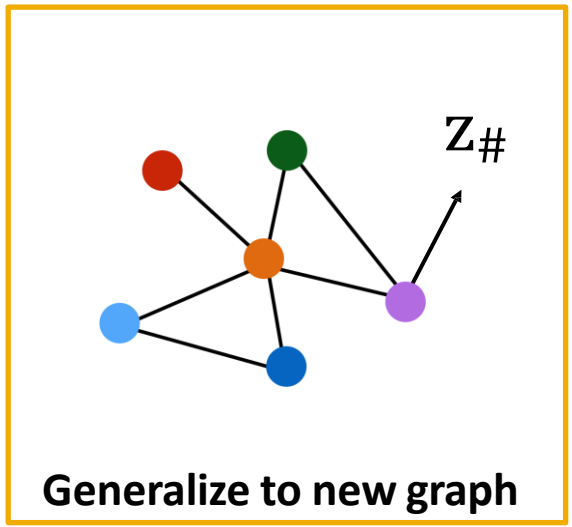


Compute graph for node A

Compute graph for node B

Inductive Capability: New Graphs

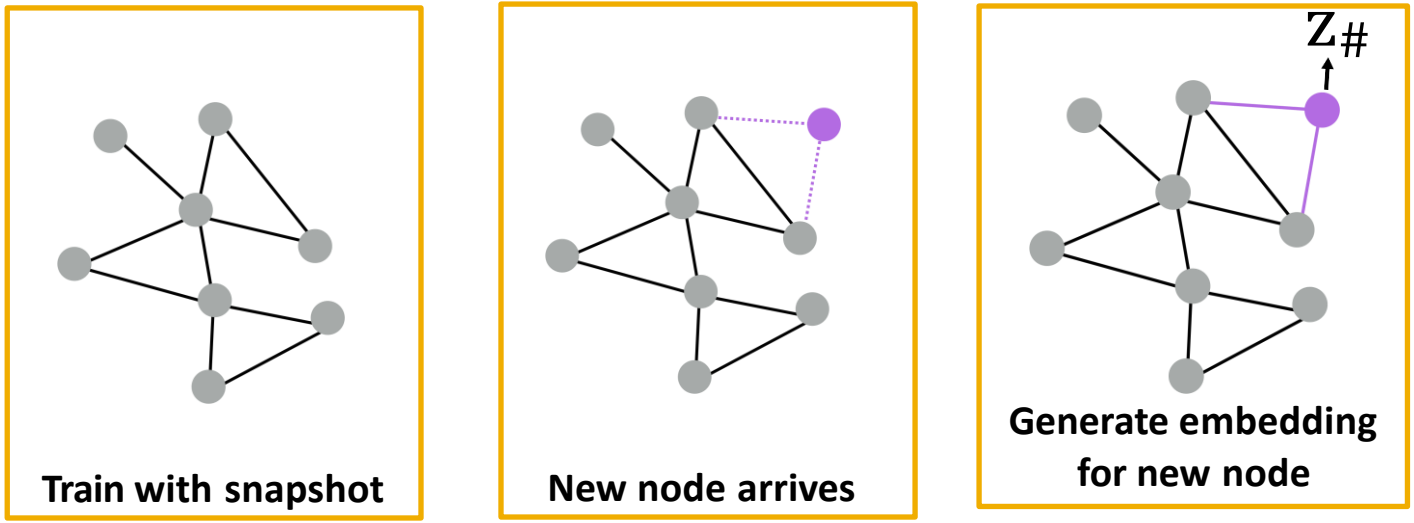




Inductive node embedding → Generalize to entirely unseen graphs

E.g., train on protein interaction graph from model organism A and generate embeddings on newly collected data about organism B

Inductive Capability: New Nodes



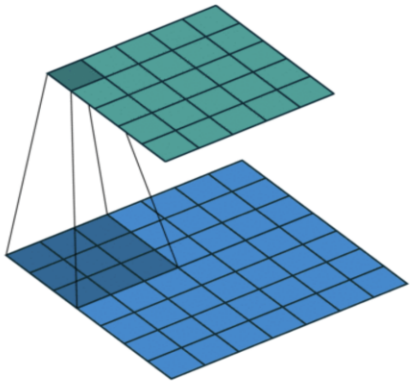
- Many application settings constantly encounter previously unseen nodes:
  - E.g., Reddit, YouTube, Google Scholar
- Need to generate new embeddings "on the fly"

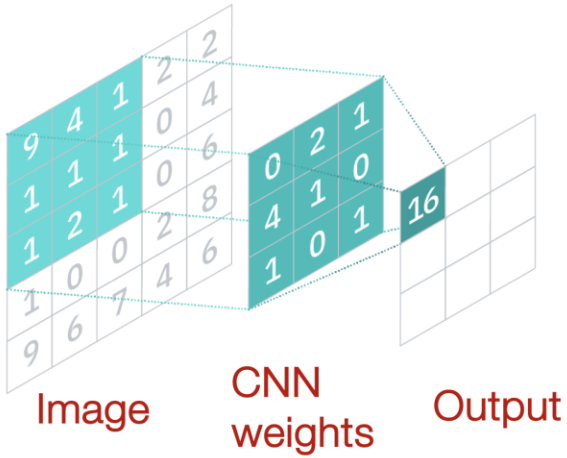
Architecture Comparison

- **How do GNNs compare to prominent architectures such as Convolutional Neural Nets?**

Convolutional Neural Network

Convolutional neural network (CNN) layer with 3x3 filter:



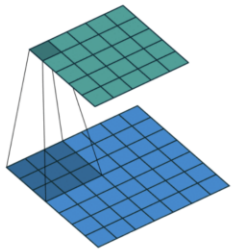


CNN formulation: $\mathrm{h}_v^{(l+1)} = \sigma(\sum_{u \in \mathrm{N}(v) \cup \{v\}} \mathrm{W}_l^u \mathrm{h}_u^{(l)}), \quad \forall l \in \{0, \ldots, L-1\}$

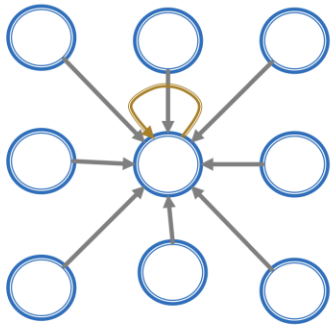
$N(v)$ represents the 8 neighbor pixels of v.

GNN vs. CNN

Convolutional neural network (CNN) layer with 3x3 filter:



Image



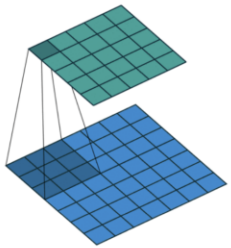
Graph

- GNN formulation: $\mathrm{h}_v^{(l+1)} = \sigma(\mathbf{W}_l \sum_{u \in \mathrm{N}(v)} \frac{\mathrm{h}_u^{(l)}}{|\mathrm{N}(v)|} + \mathrm{B}_l \mathrm{h}_v^{(l)}), \forall l \in \{0, \dots, L-1\}$
- CNN formulation: (previous slide) $\mathrm{h}_v^{(l+1)} = \sigma(\sum_{u \in \mathrm{N}(v) \cup \{v\}} \mathrm{W}_l^u \mathrm{h}_u^{(l)}), \forall l \in \{0, \dots, L-1\}$

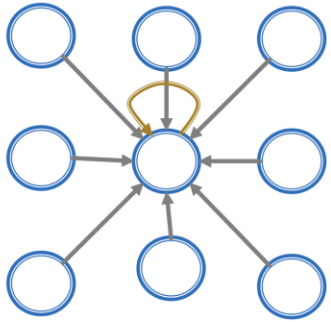
  if we rewrite: $\mathrm{h}_v^{(l+1)} = \sigma(\sum_{u \in \mathrm{N}(v)} \mathbf{W}_l^u \mathrm{h}_u^{(l)} + \mathrm{B}_l \mathrm{h}_v^{(l)}), \forall l \in \{0, \dots, L-1\}$

GNN vs. CNN

Convolutional neural network (CNN) layer with 3x3 filter:



Image



Graph

GNN formulation: $\mathrm{h}_v^{(l+1)} = \sigma(\mathbf{W}_l \sum_{u \in \mathrm{N}(v)} \frac{\mathrm{h}_u^{(l)}}{|\mathrm{N}(v)|} + \mathrm{B}_l \mathrm{h}_v^{(l)}), \forall l \in \{0, \ldots, L-1\}$

CNN formulation: $\mathrm{h}_v^{(l+1)} = \sigma(\sum_{u \in \mathrm{N}(v)} \mathbf{W}_l^u \mathrm{h}_u^{(l)} + \mathrm{B}_l \mathrm{h}_v^{(l)}), \forall l \in \{0, \ldots, L-1\}$

Key difference: We can learn different W_l^u for different "neighbor" u for pixel v on the image. The reason is we can pick an order for the 9 neighbors using **relative position** to the center pixel: {(-1,-1). (-1,0), (-1, 1), …, (1, 1)}

GNN vs. CNN

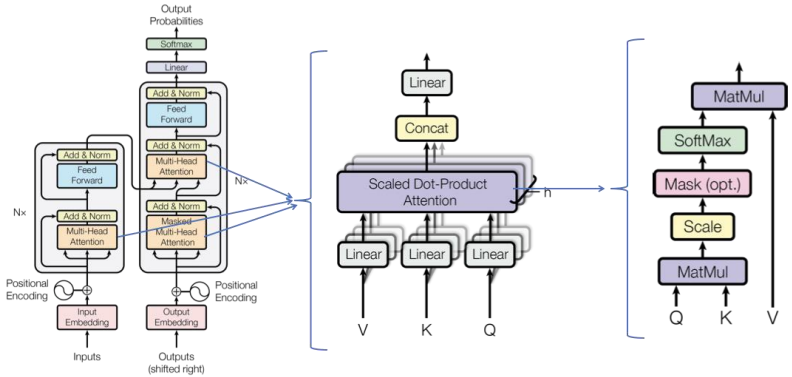
Convolutional neural network (CNN) layer with 3x3 filter:

- **CNN can be seen as a special GNN with fixed neighbor size and ordering:**
  - **The size of the filter is pre-defined for a CNN.**
  - **The advantage of GNN is it processes arbitrary graphs with different degrees for each node.**
- **CNN is not permutation invariant/equivariant.**
  - **Switching the order of pixels leads to different outputs.**

Key difference: We can learn different W^{μ} for different "neighbor" u for pixel v on the image. The reason is we can pick an order for the 9 neighbors using **relative position** to the center pixel: {(-1,-1). (-1,0), (-1, 1), …, (1, 1)}

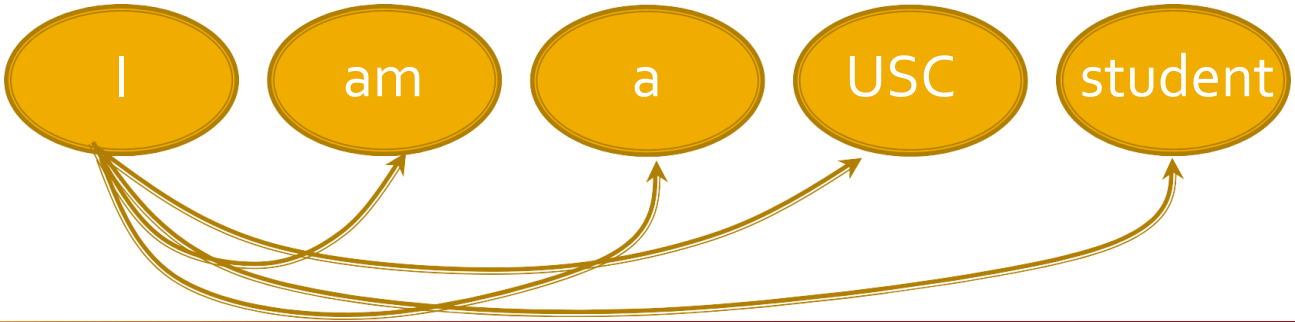
Transformer

Transformer is one of the most popular architectures that achieves great performance in many sequence modeling tasks.



Key component: self-attention

- Every token/word attends to all the other tokens/words via matrix calculation.

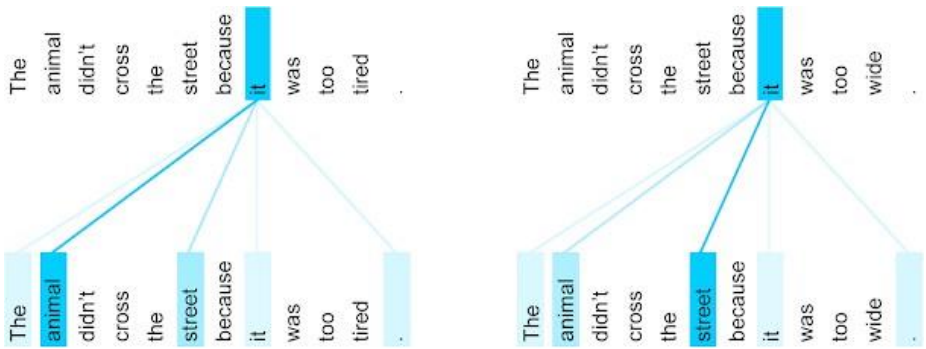


Transformer

A general definition of attention:

Given a set of vector *values*, and a vector *query*, **attention** is a technique to compute a weighted sum of the values, dependent on the query.

Each token/word has a **value vector** and a **query vector**. The value vector can be seen as the representation of the token/word. We use the query vector to calculate the attention score (weights in the weighted sum).



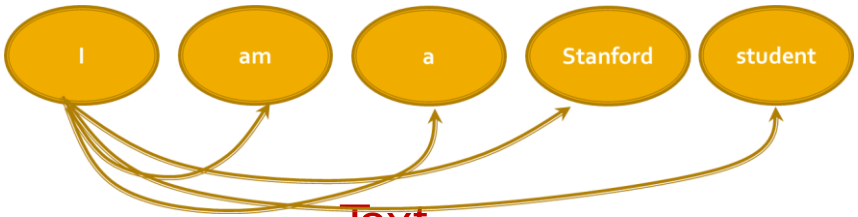
https://towardsdatascience.com/transformers-are-graph-neural-networks-bca9f75412aa

GNN vs. Transformer

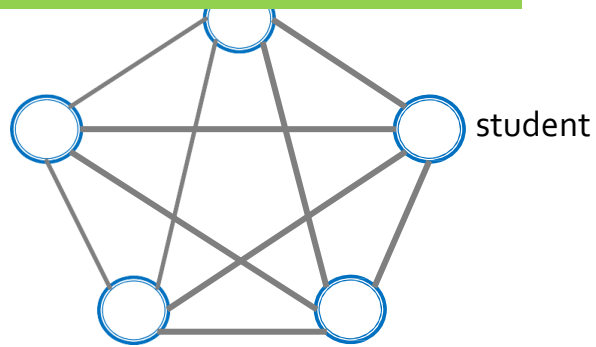
Transformer layer can be seen as a special GNN that runs on a fully- connected "word" graph!

Sinc

words, the computation graph of a transformer layer is identical to that of a GNN on the **fully-connected "word" graph**.



Text



(Complete) Graph

Summary

- **In this lecture, we introduced**
  - Idea for Deep Learning for Graphs
    - Multiple layers of embedding transformation
    - At every layer, use the embedding at previous layer as the input
    - Aggregation of neighbors and self-embeddings
  - Graph Convolutional Network
    - Mean aggregation (can be expressed in matrix form)
  - GNN is a general architecture
    - CNN can be viewed as a special GNN