MAY
7
2024

ICLR 2024 Test of Time Award

YIZHOU SUN / ICLR 2024

ICLR is in its 12th year! For the inaugural ICLR Test of Time award, the Program Chairs examined papers from ICLR 2013 & 2014, and looked for ones with long-lasting impact.

Congratulations to the authors of the Test of Time winner and runner up!

Test of Time

Auto-Encoding Variational Bayes

Diederik Kingma, Max Welling

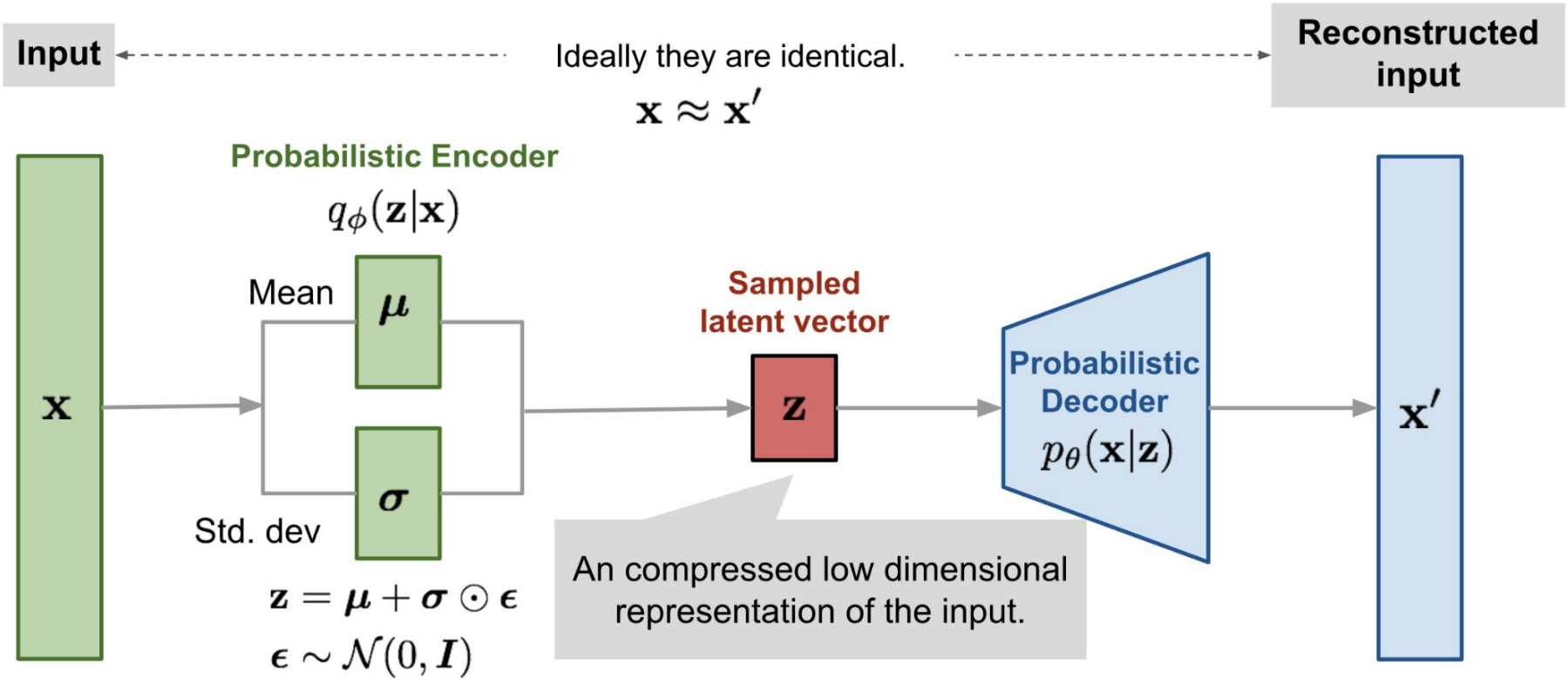
https://arxiv.org/abs/1312.6114

Probabilistic modeling is one of the most fundamental ways in which we reason about the world. This paper spearheaded the integration of deep learning with scalable probabilistic inference (amortized mean-field variational inference via a so-called reparameterization trick), giving rise to the Variational Autoencoder (VAE). The lasting value of this work is rooted in its elegance. The principles used to develop VAEs deepened our understanding of the interplay between deep learning and probabilistic modeling, and sparked the development of many subsequent interesting probabilistic models and encoding approaches.

https://blog.iclr.cc/2024/05/07/iclr-2024-test-of-time-award/

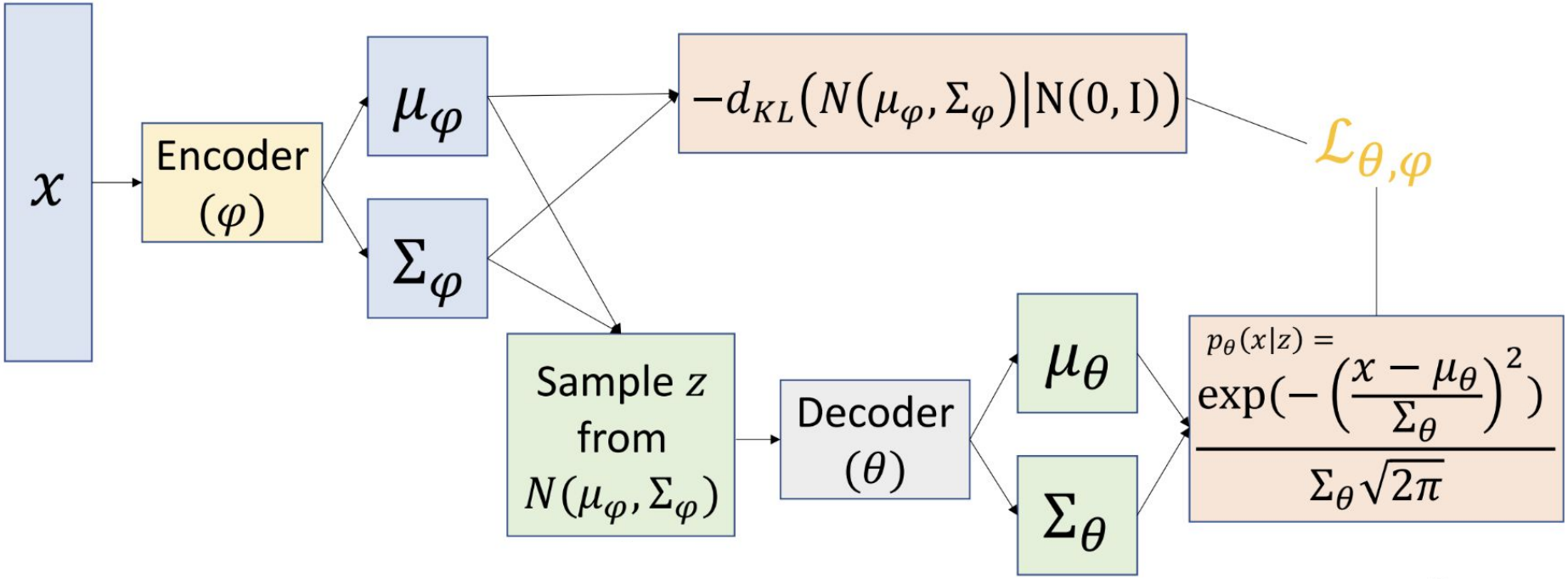
VAE original paper: https://arxiv.org/abs/1312.6114

VAE tutorial: https://arxiv.org/abs/1606.05908


Input
Ideally they are identical.
$\mathbf{x} \approx \mathbf{x}'$
Reconstructed input
Probabilistic Encoder
$q_\phi(\mathbf{z}|\mathbf{x})$
Mean
$\boldsymbol{\mu}$
$\boldsymbol{\sigma}$
Std. dev
$\mathbf{z} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon}$
$\boldsymbol{\epsilon} \sim \mathcal{N}(0, \boldsymbol{I})$
$\mathbf{x}$
Sampled latent vector
$\mathbf{z}$
An compressed low dimensional representation of the input.
Probabilistic Decoder
$p_\theta(\mathbf{x}|\mathbf{z})$
$\mathbf{x}'$

$$\mathcal{L}_{\theta,\varphi}(x) = \mathbb{E}_{z\sim q_\varphi}[\log p_\theta(x|z)] \; - \; d_{KL}(q_\varphi(z|x)|p(z))$$

Training VAEs



$$d_{KL}(q_\varphi(z|x)\,|\,p_\theta(z|x)) = \mathbb{E}_{z\sim q_\varphi}[\log q_\varphi(z|x) - \log\, p_\theta(z|x)]$$
$$= \mathbb{E}_{z\sim q_\varphi}[\log q_\varphi(z|x) - \log\, p_\theta(x|z) - \log p(z) + \log\, p_\theta(x)]$$
$$= \mathbb{E}_{z\sim q_\varphi}[\log q_\varphi(z|x) - \log\, p_\theta(x|z) - \log p(z)] + \log\, p_\theta(x)$$

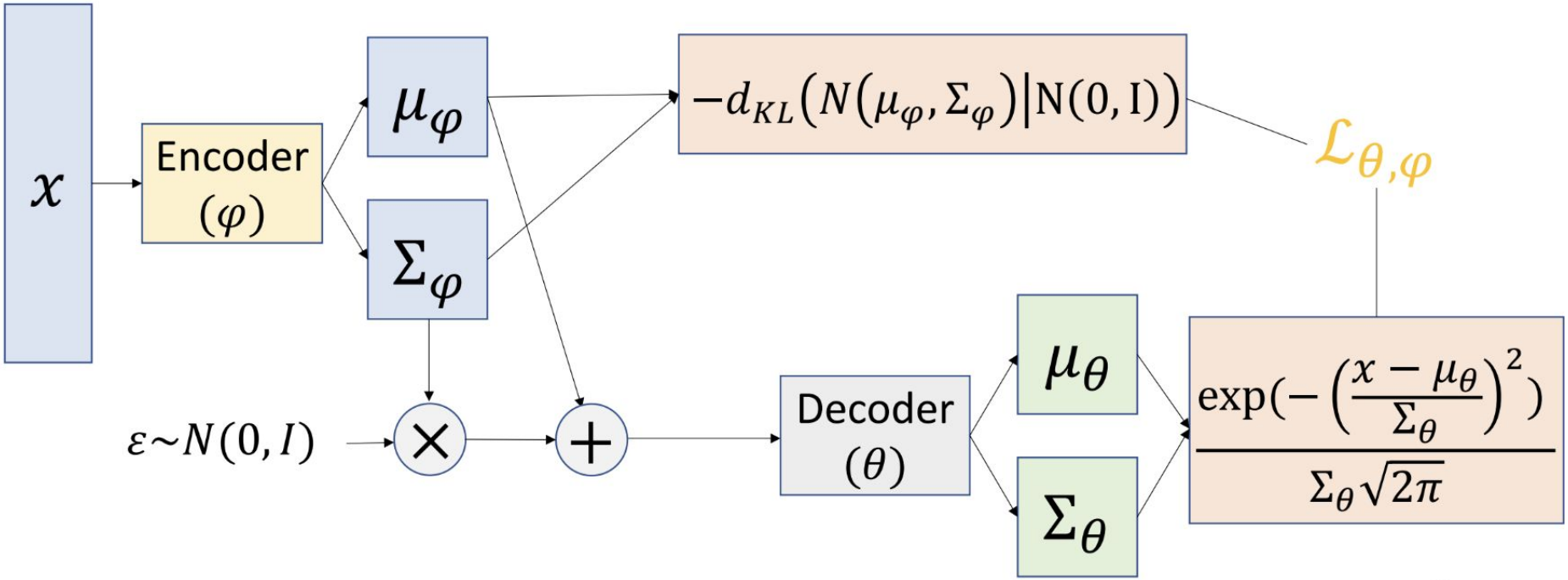
$\Rightarrow$

$$\log\, p_\theta(x) - d_{KL}(q_\varphi(z|x)\,|\,p_\theta(z|x)) = \underbrace{\mathbb{E}_{z\sim q_\varphi}[\log p_\theta(x|z)] - d_{KL}(q_\varphi(z|x)|p(z))}$$

It's called the Evidence Lower Bound (ELBO)

Training VAEs

Reparameterization trick



We want 3 things from generative models

- *Density estimation*: Given a datapoint x, what is the probability assigned by the model, i.e. $p_\theta(x)$
- *Sampling*: How can we generate novel data from the model distribution, i.e. $x_{new} \sim p_\theta(x)$
- *Unsupervised representation learning*: How can we learn meaningful feature representations for a datapoint x?

We want 3 things from generative models

- *Density estimation*: Given a datapoint x, what is the probability assigned by the model, i.e. $p_\theta(x)$

Roughly! You can estimate the joint $p_\theta(x, z)$ or a lower bound on $p_\theta(x)$.

Optimizing the lower bound is perhaps why VAE images are often blurrier than the ones generated with autoregressive models.

We want 3 things from generative models

- *Density estimation*: Given a datapoint x, what is the probability assigned by the model, i.e. $p_\theta(x)$

- *Sampling*: How can we generate novel data from the model distribution, i.e. $x_{new} \sim p_\theta(x)$

We can sample new data fast with one forward pass through the decoder!

We want 3 things from generative models

- *Density estimation*: Given a datapoint x, what is the probability assigned by the model, i.e. $p_\theta(x)$
- *Sampling*: How can we generate novel data from the model distribution, i.e. $x_{new} \sim p_\theta(x)$
- *Unsupervised representation learning*: How can we learn meaningful feature representations for a datapoint x?

Yassss! The pro of VAEs is that you learn a latent code z which is produced by the encoder.

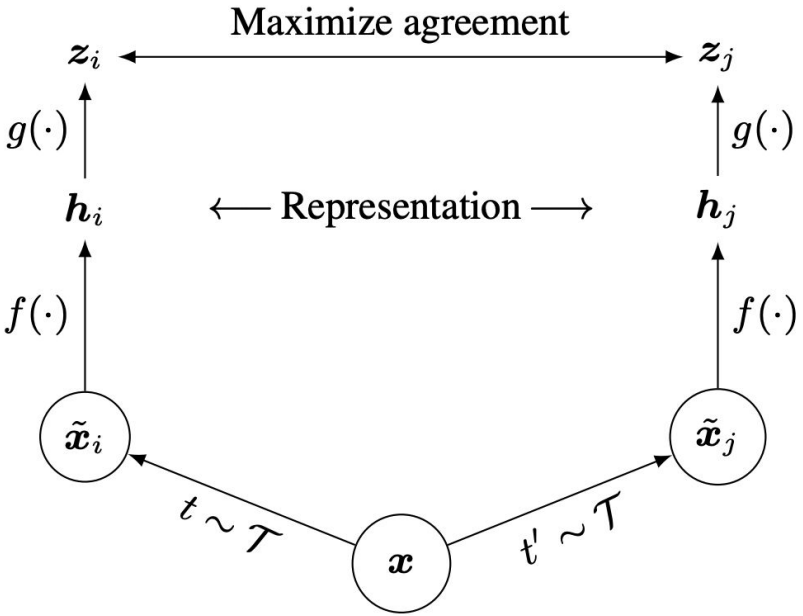
Contrastive Learning

Modern SSL | The Contrastive Paradigm

The Contrastive Paradigm:

Train a classifier where each instance in the training set is its own class.

AKA *instance discrimination*

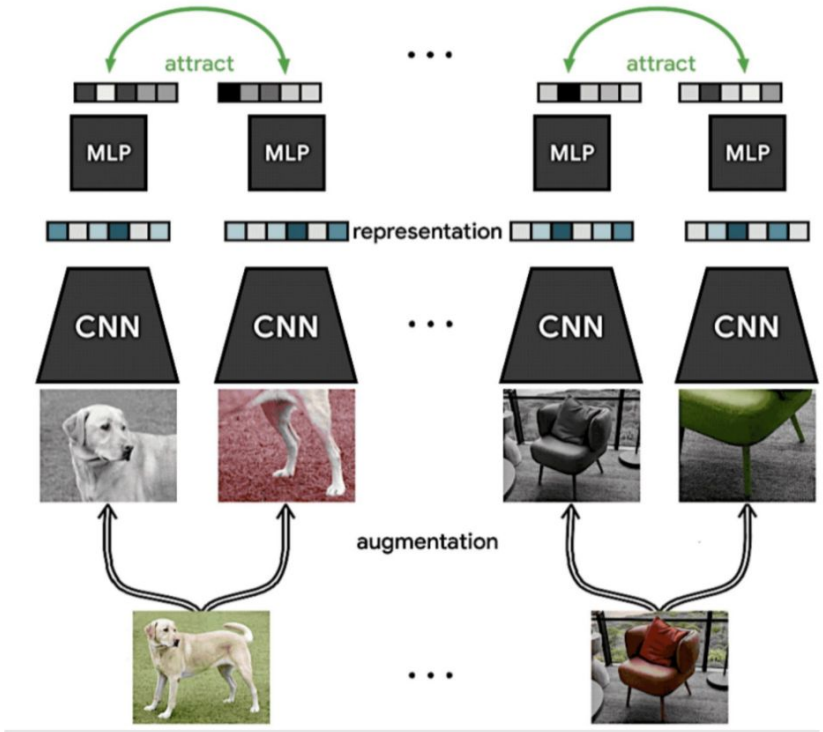


SimCLR: A Simple Framework for Contrastive Learning of Visual Representations

https://arxiv.org/pdf/2002.05709

$$\ell_{i,j} = -\log \frac{\exp(\text{sim}(\boldsymbol{z}_i, \boldsymbol{z}_j)/\tau)}{\sum_{k=1}^{2N} \mathbb{1}_{[k \neq i]} \exp(\text{sim}(\boldsymbol{z}_i, \boldsymbol{z}_k)/\tau)}$$

Modern SSL | The Contrastive Paradigm

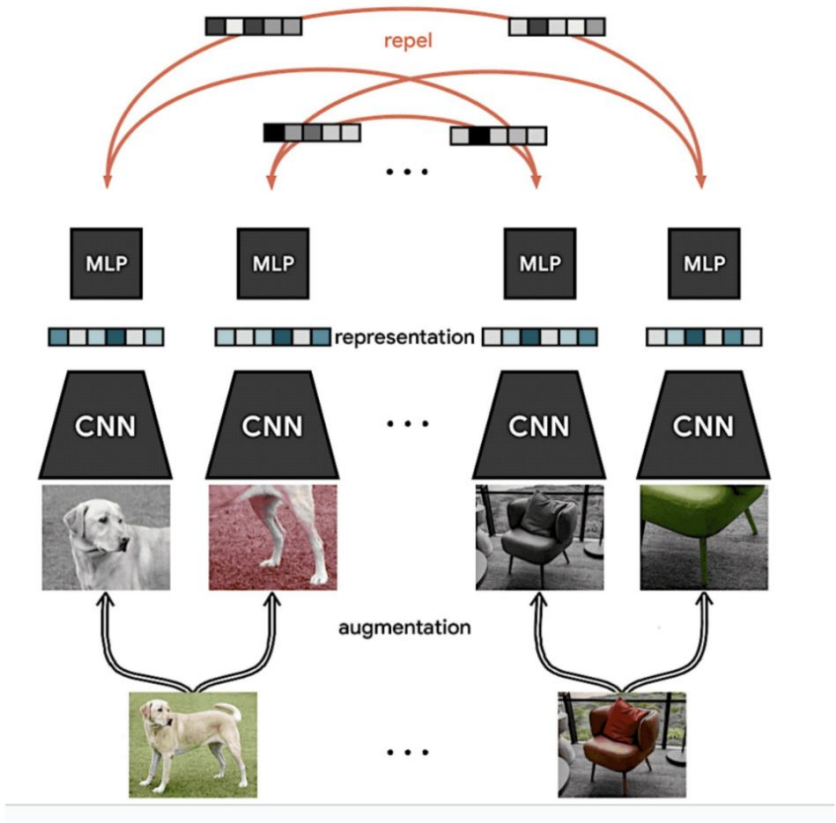


Algorithm 1 SimCLR's main learning algorithm.

input: batch size N, constant τ, structure of $f, g, \mathcal{T}$.
for sampled minibatch $\{\boldsymbol{x}_k\}_{k=1}^{N}$ **do**
  for all $k \in \{1, \ldots, N\}$ **do**
    draw two augmentation functions $t \sim \mathcal{T}$, $t' \sim \mathcal{T}$
    # the first augmentation
    $\tilde{\boldsymbol{x}}_{2k-1} = t(\boldsymbol{x}_k)$
    $\boldsymbol{h}_{2k-1} = f(\tilde{\boldsymbol{x}}_{2k-1})$ # representation
    $\boldsymbol{z}_{2k-1} = g(\boldsymbol{h}_{2k-1})$ # projection
    # the second augmentation
    $\tilde{\boldsymbol{x}}_{2k} = t'(\boldsymbol{x}_k)$
    $\boldsymbol{h}_{2k} = f(\tilde{\boldsymbol{x}}_{2k})$ # representation
    $\boldsymbol{z}_{2k} = g(\boldsymbol{h}_{2k})$ # projection
  end for
  for all $i \in \{1, \ldots, 2N\}$ and $j \in \{1, \ldots, 2N\}$ **do**
    $s_{i,j} = \boldsymbol{z}_i^\top \boldsymbol{z}_j / (\|\boldsymbol{z}_i\| \|\boldsymbol{z}_j\|)$ # pairwise similarity
  end for
  define $\ell(i, j)$ **as** $\ell(i, j) = -\log \frac{\exp(s_{i,j}/\tau)}{\sum_{k=1}^{2N} \mathbb{1}_{[k \neq i]} \exp(s_{i,k}/\tau)}$

end for
return encoder network $f(\cdot)$, and throw away $g(\cdot)$

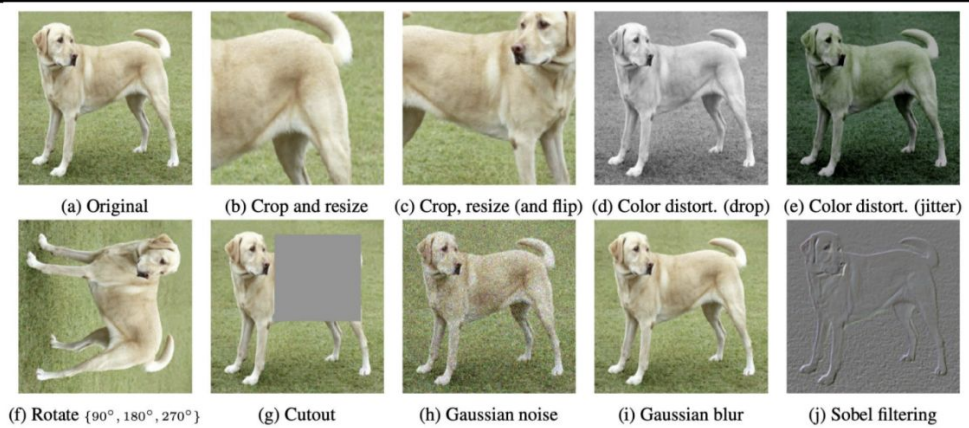
Modern SSL | The Contrastive Paradigm



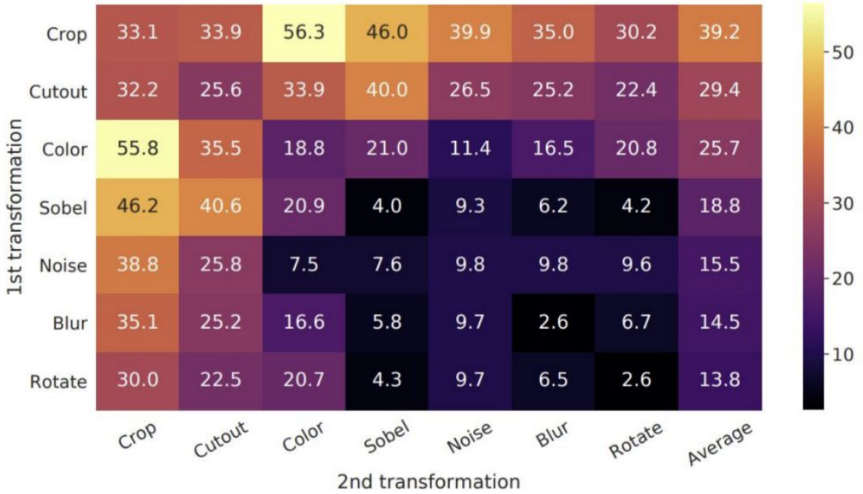
Algorithm 1 SimCLR's main learning algorithm.

input: batch size N, constant τ, structure of $f, g, \mathcal{T}$.
for sampled minibatch $\{\boldsymbol{x}_k\}_{k=1}^{N}$ **do**
  for all $k \in \{1, \ldots, N\}$ **do**
    draw two augmentation functions $t \sim \mathcal{T}$, $t' \sim \mathcal{T}$
    # the first augmentation
    $\tilde{\boldsymbol{x}}_{2k-1} = t(\boldsymbol{x}_k)$
    $\boldsymbol{h}_{2k-1} = f(\tilde{\boldsymbol{x}}_{2k-1})$ # representation
    $\boldsymbol{z}_{2k-1} = g(\boldsymbol{h}_{2k-1})$ # projection
    # the second augmentation
    $\tilde{\boldsymbol{x}}_{2k} = t'(\boldsymbol{x}_k)$
    $\boldsymbol{h}_{2k} = f(\tilde{\boldsymbol{x}}_{2k})$ # representation
    $\boldsymbol{z}_{2k} = g(\boldsymbol{h}_{2k})$ # projection
  end for
  for all $i \in \{1, \ldots, 2N\}$ and $j \in \{1, \ldots, 2N\}$ **do**
    $s_{i,j} = \boldsymbol{z}_i^\top \boldsymbol{z}_j / (\|\boldsymbol{z}_i\| \|\boldsymbol{z}_j\|)$ # pairwise similarity
  end for
  define $\ell(i, j)$ **as** $\ell(i, j) = -\log \frac{\exp(s_{i,j}/\tau)}{\sum_{k=1}^{2N} \mathbb{1}_{[k \neq i]} \exp(s_{i,k}/\tau)}$
  $\mathcal{L} = \frac{1}{2N} \sum_{k=1}^{N} [\ell(2k-1, 2k) + \ell(2k, 2k-1)]$
  update networks f and g to minimize $\mathcal{L}$
end for
return encoder network $f(\cdot)$, and throw away $g(\cdot)$

Modern SSL | The Contrastive Paradigm

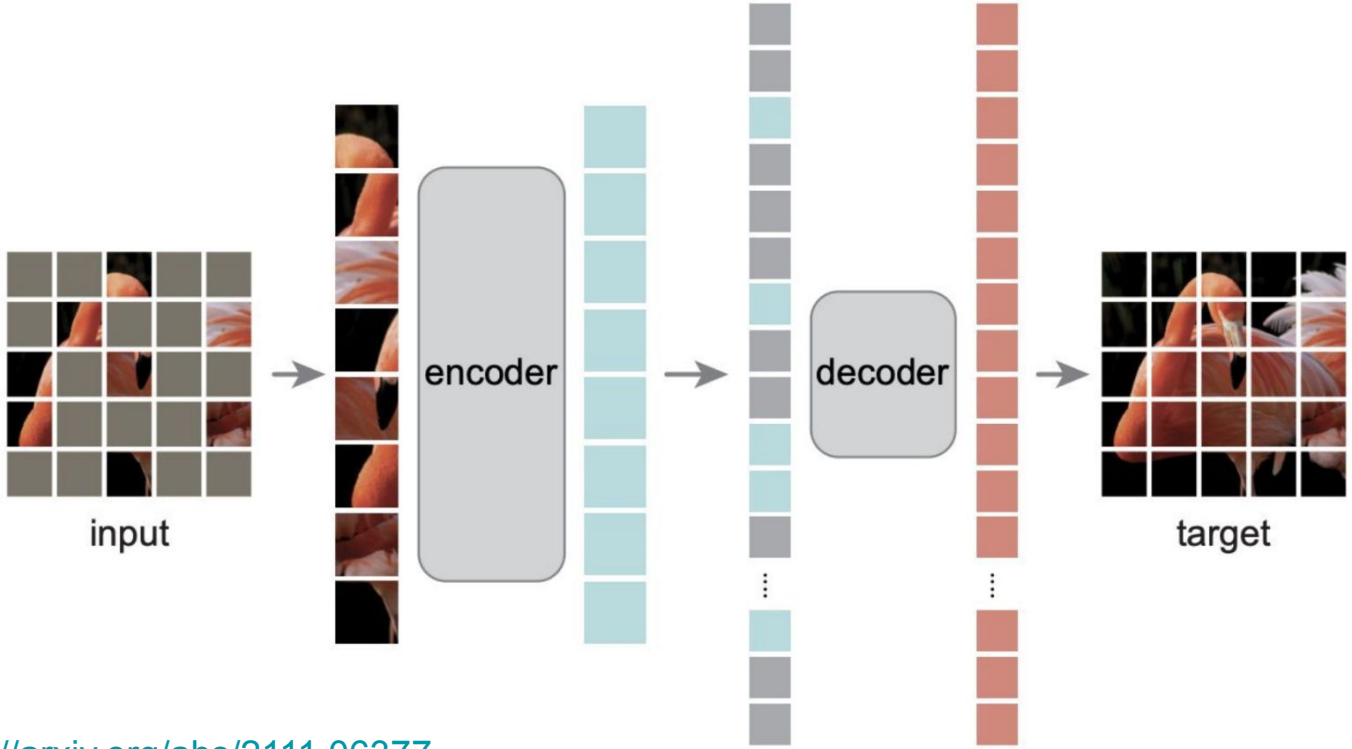


Composite Augmentation is the key!



Modern SSL | The Masking Paradigm | MAE

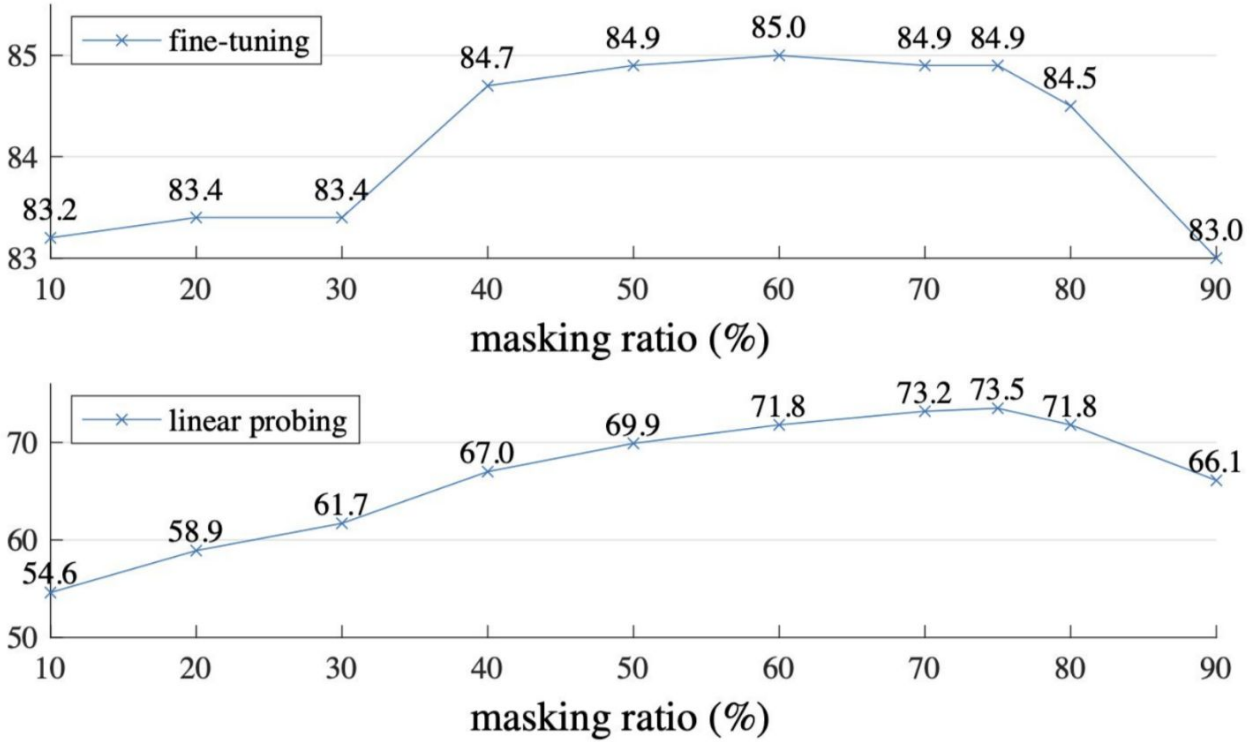
Masked Autoencoders Are Scalable Vision Learners
He et al. CVPR 2022



https://arxiv.org/abs/2111.06377

Modern SSL | The Masking Paradigm | MAE

Modern SSL | The Masking Paradigm | MAE



Intuition: Masking a large portion of the image to "create a challenging self supervisory task that requires holistic understanding beyond low-level image statistics."

Introduction to Data Mining

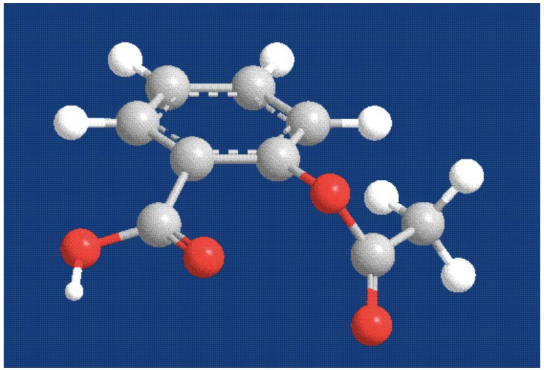
CS 145

Lecture 11:

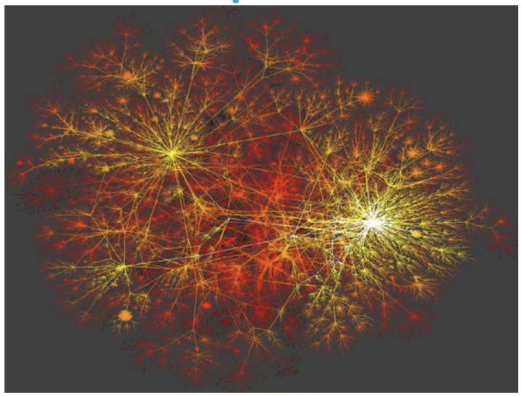
Graph and Networks:

Random Walk and PageRank

Graph, Graph, Everywhere



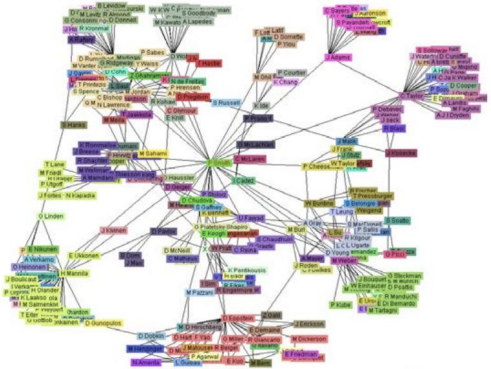
Aspirin



Internet

from H. Jeong et al Nature 411, 41 (2001)

Yeast protein interaction network



Co-author network

Many types of Data are Graphs

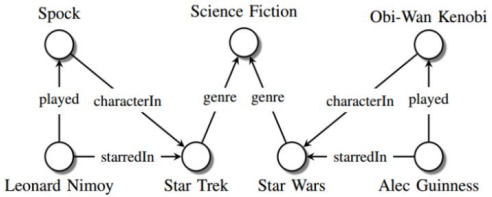


Image credit: Maximilian Nickel et al

Knowledge Graphs

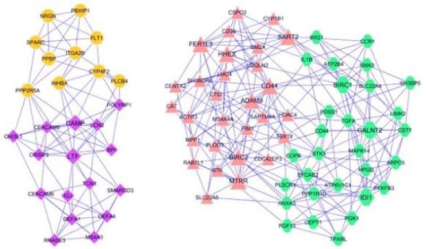


Image credit: ese.wustl.edu

Regulatory Networks

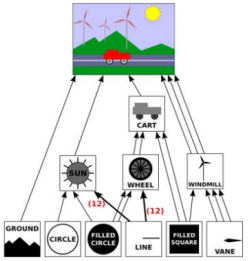


Image credit: math.hws.edu

Scene Graphs

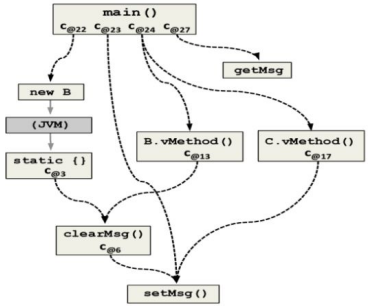


Image credit: ResearchGate

Code Graphs

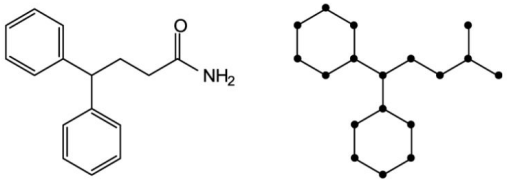


Image credit: MDPI

Molecules

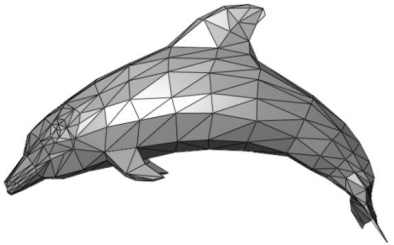


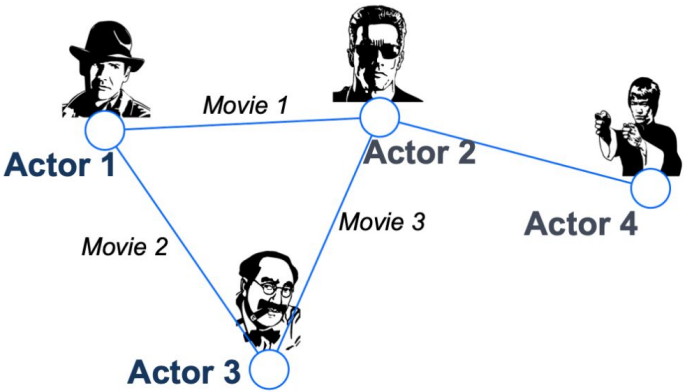
Image credit: Wikipedia

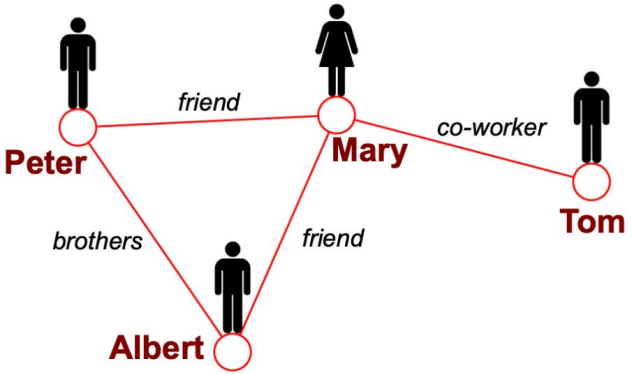
3D Shapes

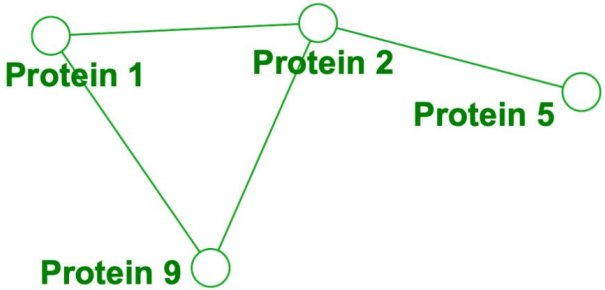
Why Graphs?

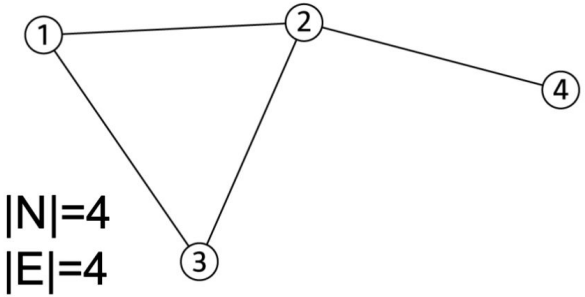
Graphs are a general language for describing and analyzing entities with relations/interactions

Graphs: A Common Language





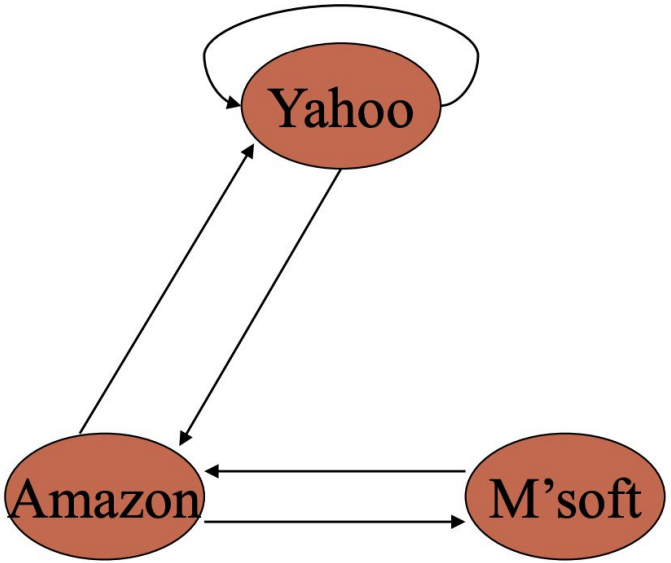




Representation of Graph

- $G =< V, E >$
  - $V = \{u_1, \ldots, u_n\}$: node set
  - $E \subseteq V \times V$: edge set
- Adjacency matrix
  - $A = \{a_{ij}\}, i, j = 1, \ldots, N$
    - $a_{ij} = 1, if \ < u_i, u_j > \in E$
    - $a_{ij} = 0, if \ < u_i, u_j > \notin E$
  - Undirected graph vs. Directed graph
    - $A = A^{\mathrm{T}} \ vs. \ A \neq A^{\mathrm{T}}$
  - Weighted graph
    - Use *W* instead of *A*, where w_{ij} represents the weight of edge $< u_i, u_j >$

Example



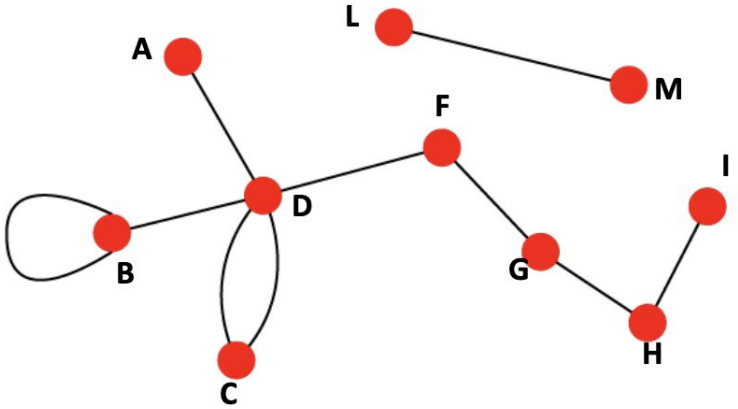
	y	a	m
y	1	1	0
a	1	0	1
m	0	1	0

Adjacency matrix A

Directed vs UnDirected Graphs

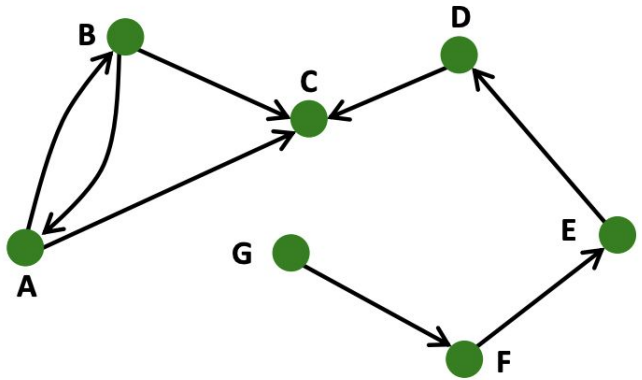
Undirected

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

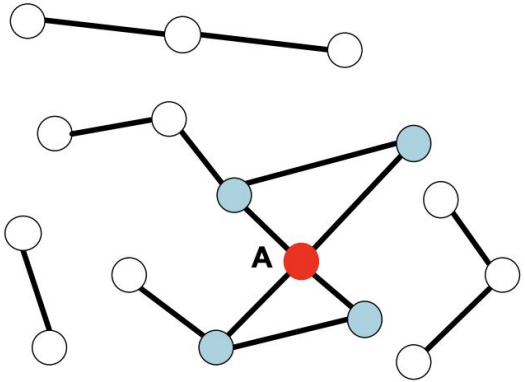


Directed

- **Links:** directed



Undirected

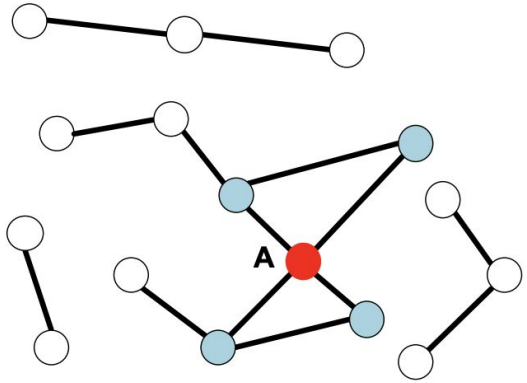


Node degree, k_i: the number of edges adjacent to node ***i***

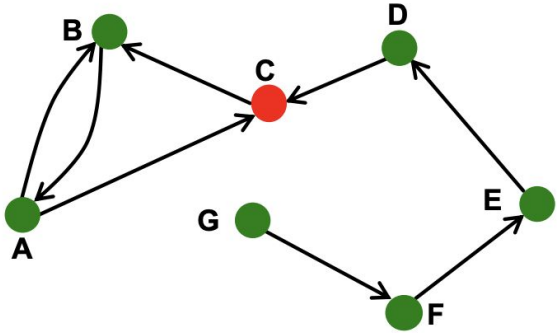
$$k_A = 4$$

Avg. degree: $\bar{k} = \langle k \rangle = \frac{1}{N}\sum_{i=1}^{N} k_i = \frac{2E}{N}$

Undirected



Directed



Source: Node with $k^{in} = 0$
Sink: Node with $k^{out} = 0$

Node degree, k_i: the number of edges adjacent to node ***i***

$$k_A = 4$$

Avg. degree: $\bar{k} = \langle k \rangle = \frac{1}{N}\sum_{i=1}^{N} k_i = \frac{2E}{N}$

In directed networks we define an **in-degree** and **out-degree.**

The (total) degree of a node is the sum of in- and out-degrees.

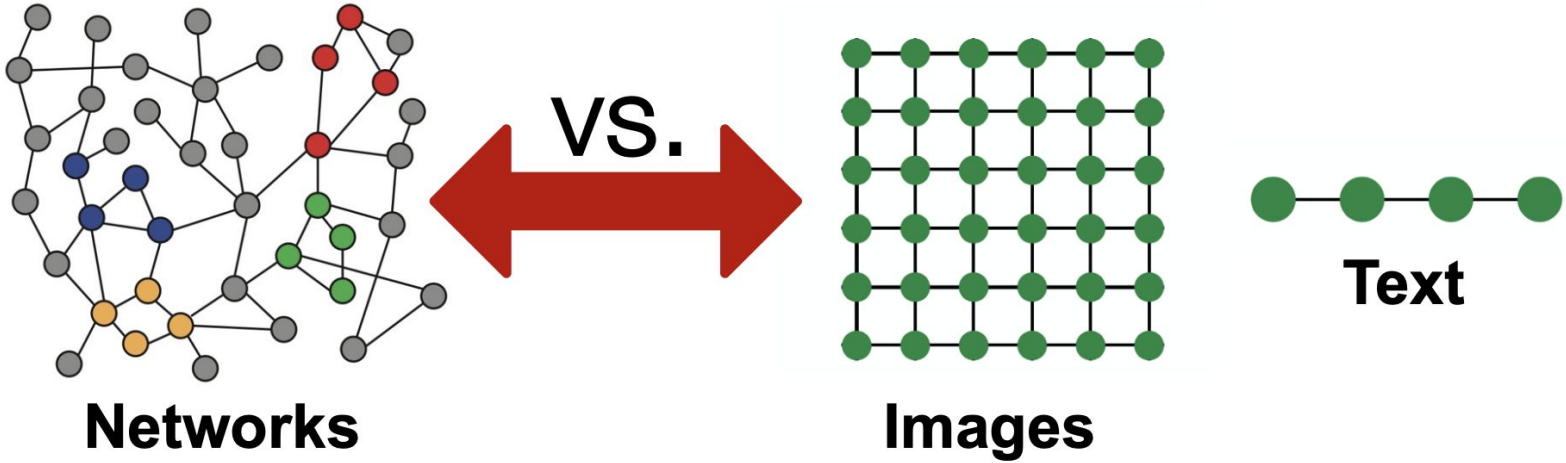
$$k_C^{in} = 2 \qquad k_C^{out} = 1 \qquad k_C = 3$$

$$\bar{k} = \frac{E}{N}$$

$$\overline{k^{in}} = \overline{k^{out}}$$

Networks are complex.

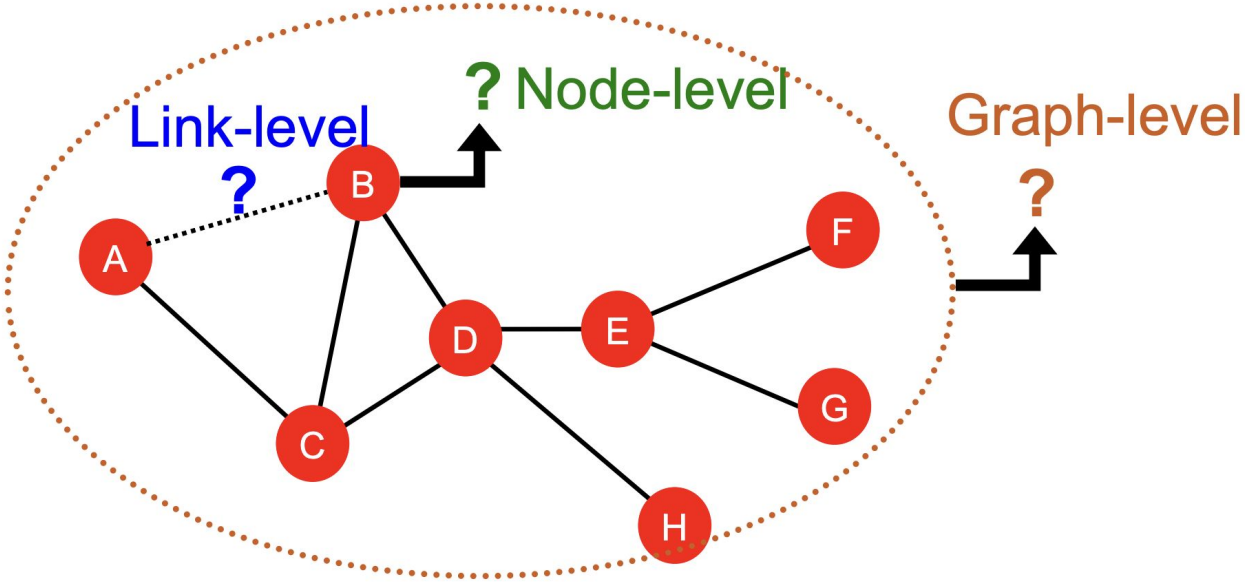
- Arbitrary size and complex topological structure (*i.e.*, no spatial locality like grids)



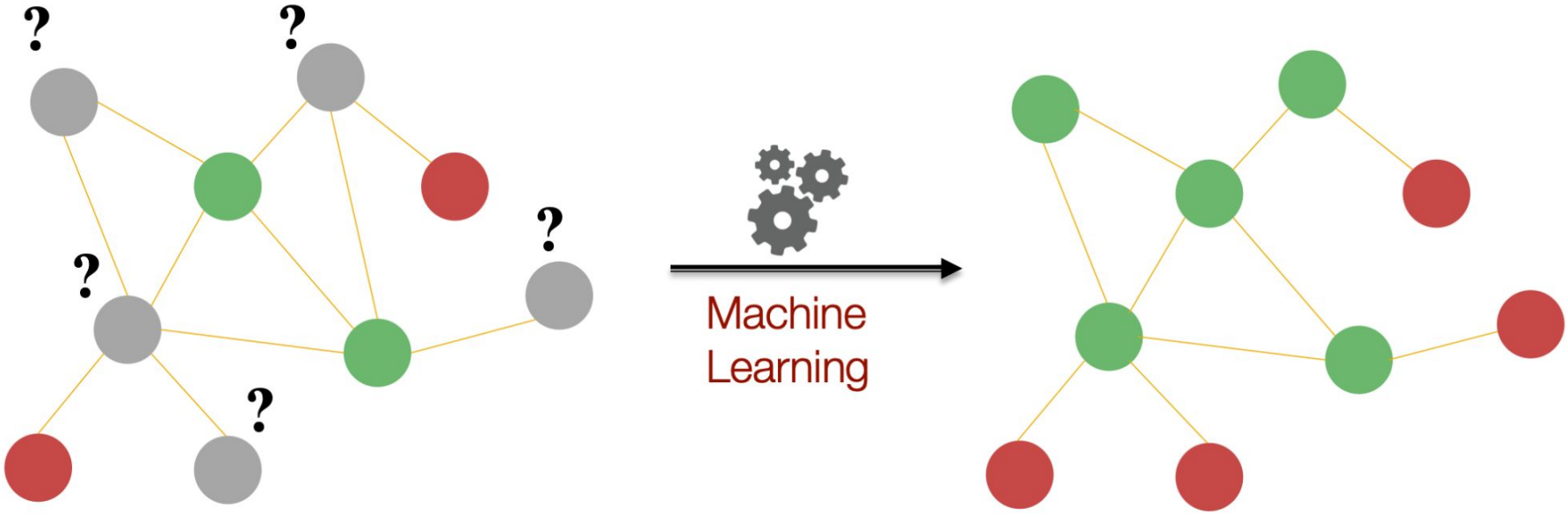
- No fixed node ordering or reference point
- Often dynamic and have multimodal features

Different Types of Tasks on Graphs

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



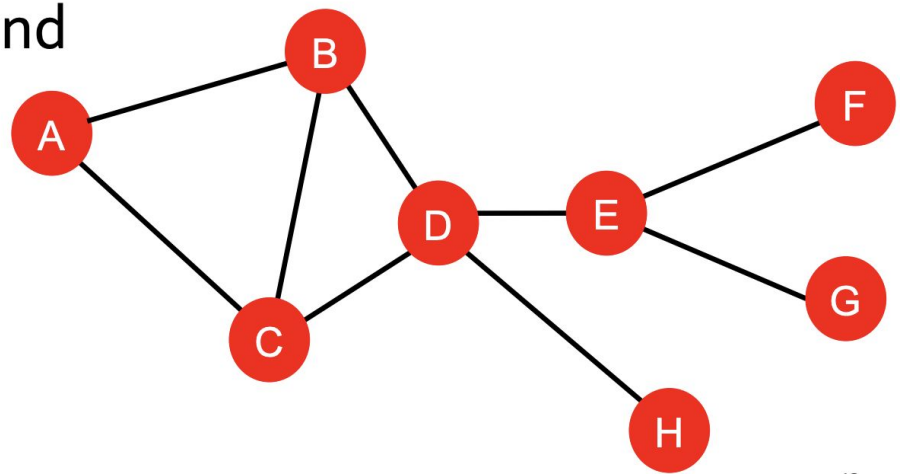
Node-Level Prediction



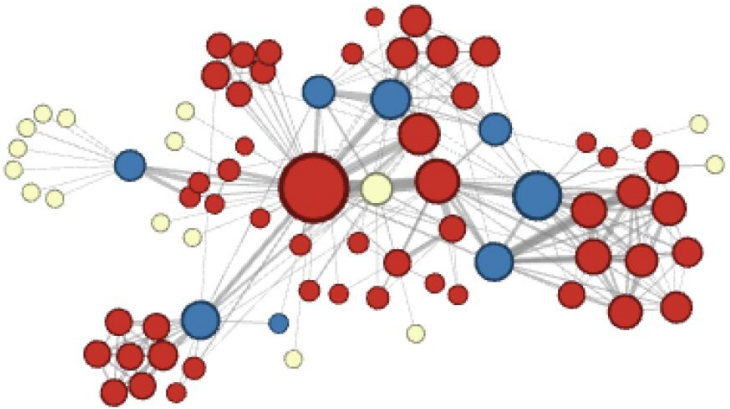
Node classification

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

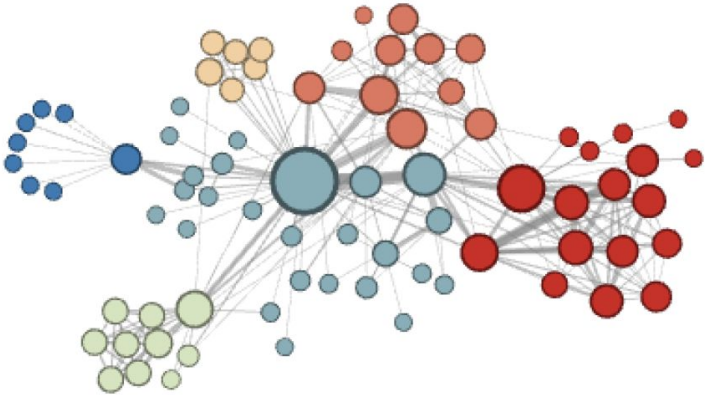
- Node degree
- Node importance & position
  - E.g., Number of shortest paths passing through a node
  - E.g., Avg. shortest path length to other nodes
- Substructures around the node



Different ways to label nodes of the network:



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

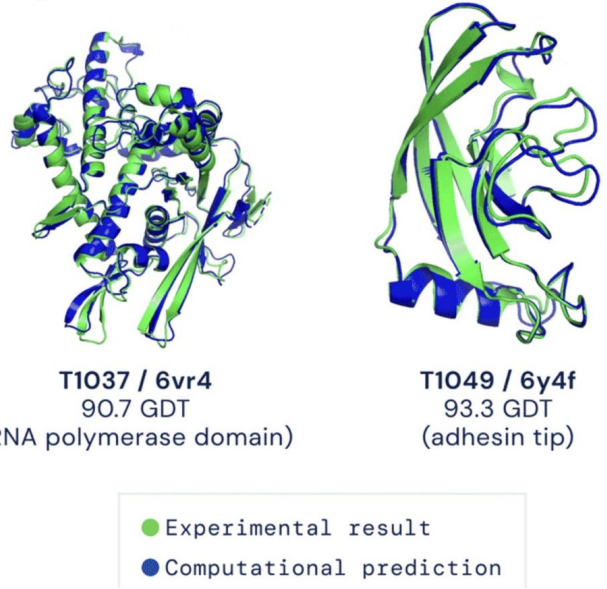


However, the features defines so far would not allow for distinguishing the above node labelling

Example: Protein Folding (Alphafold)

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

For each node predict its 3D coordinates



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

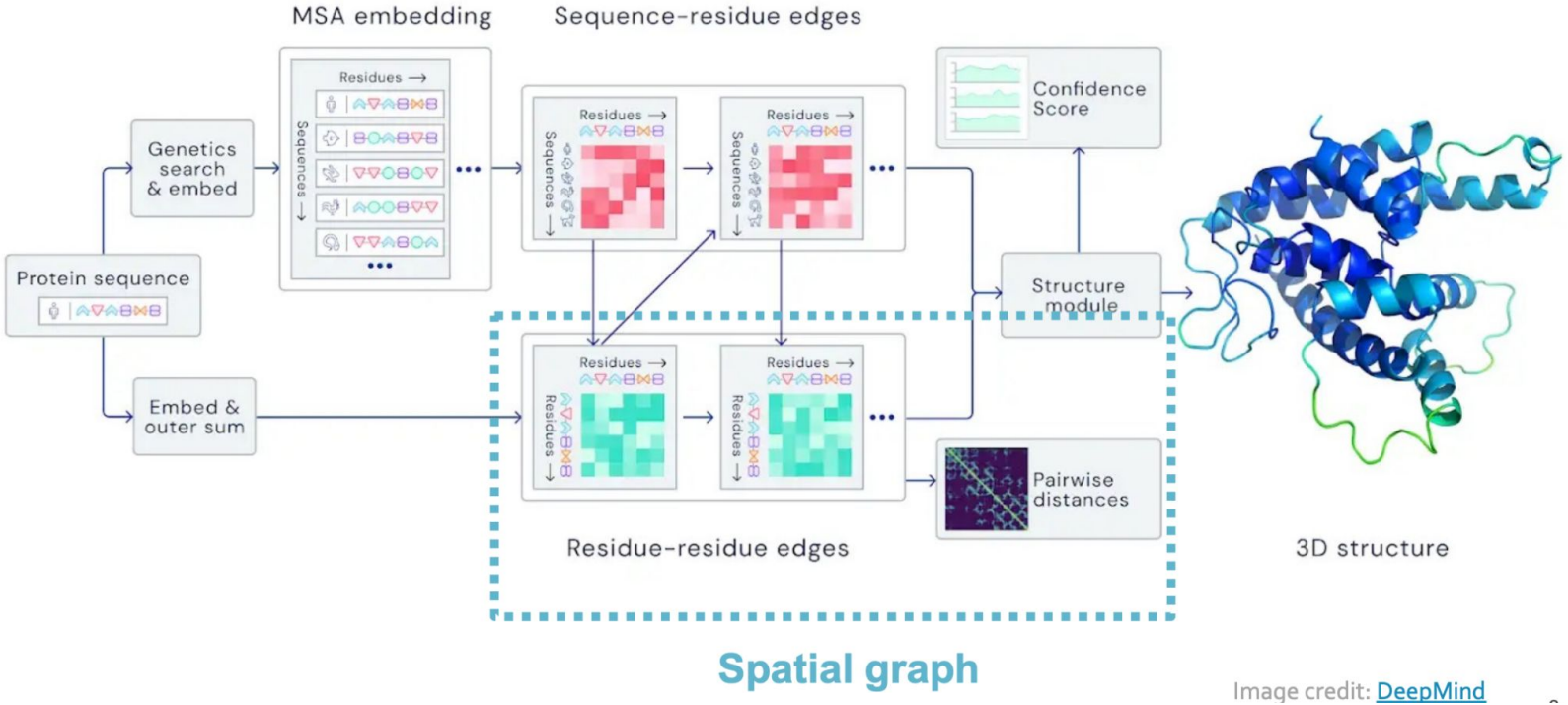
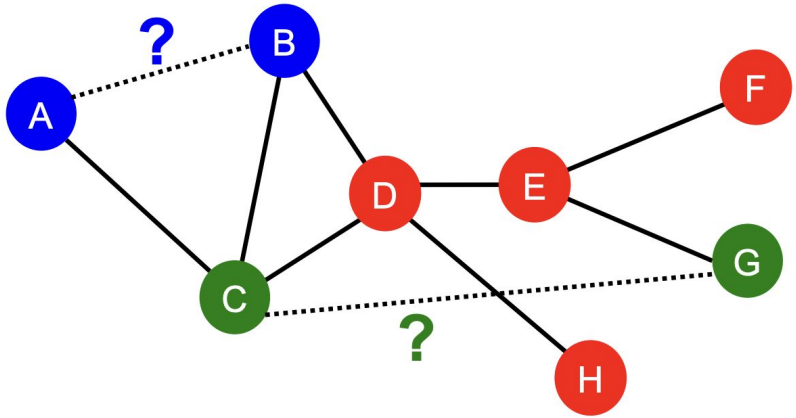


Image credit: DeepMind

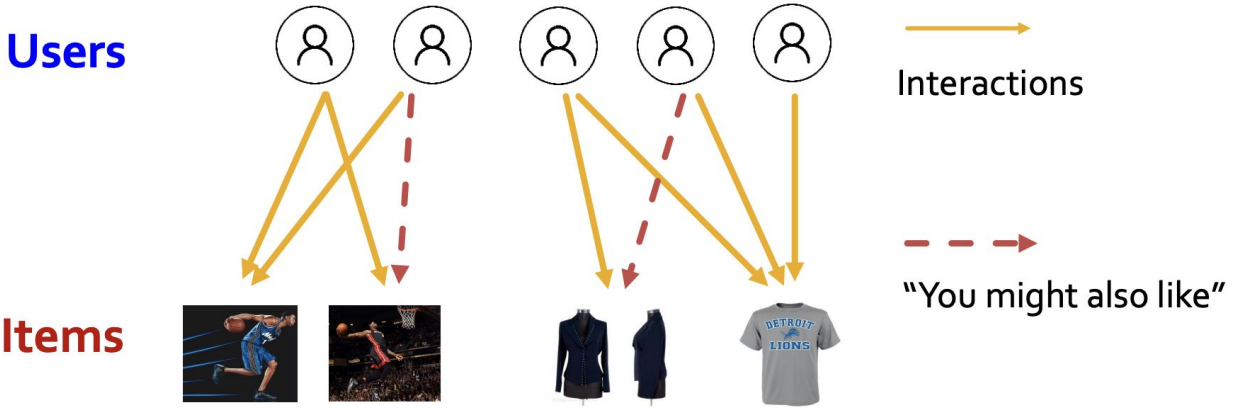
Link-Level Prediction

- 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**.

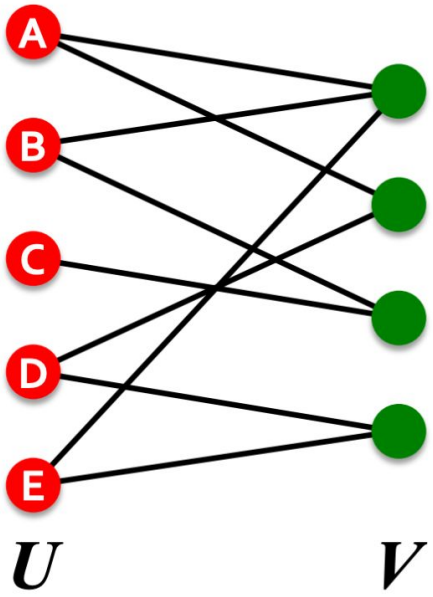


Example: Recommender System

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



- **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**



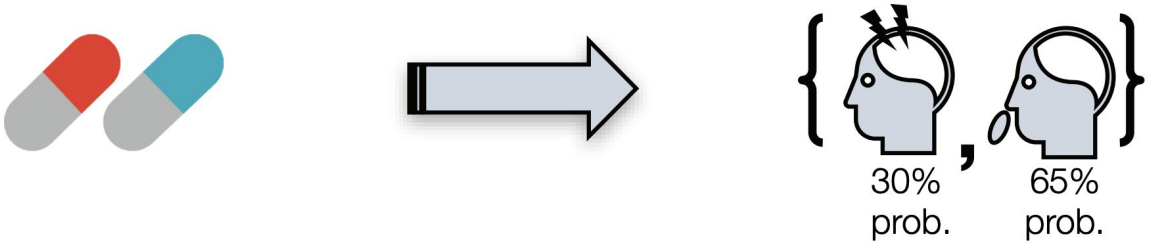
- **Examples:**
  - Authors-to-Papers (they authored)
  - Actors-to-Movies (they appeared in)
  - Users-to-Movies (they rated)
  - Recipes-to-Ingredients (they contain)
- **"Folded" networks:**
  - Author collaboration networks
  - Movie co-rating networks

Example: Drug Side Effect

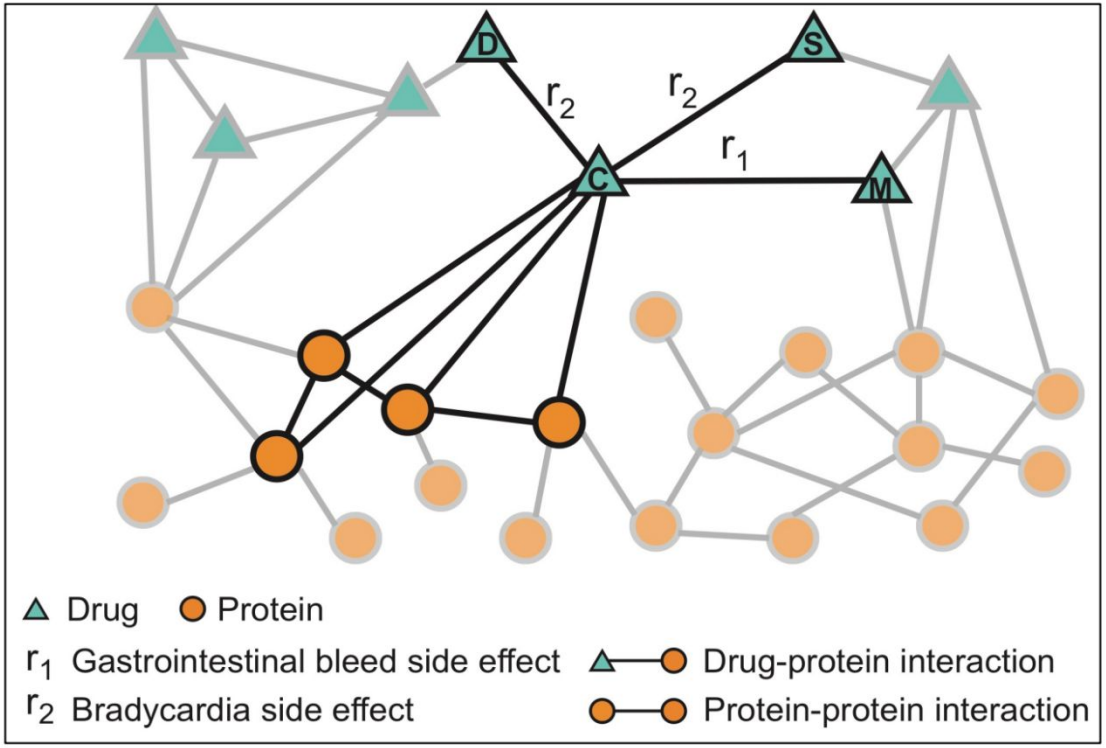
Many patients take multiple drugs to treat complex or co-existing diseases:

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

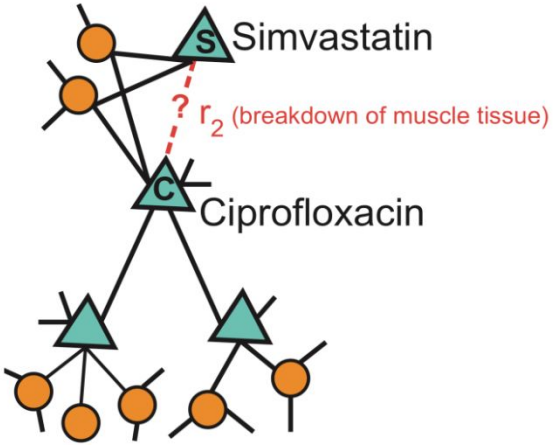
Task: Given a pair of drugs predict adverse side effects



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



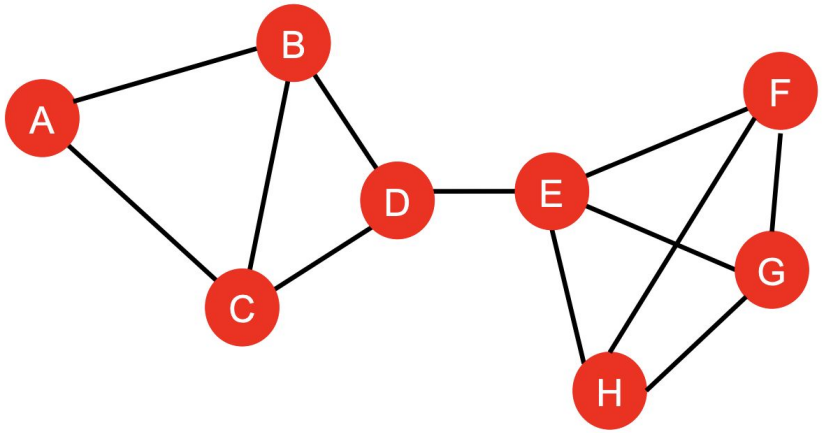
Query: How likely will Simvastatin and Ciprofloxacin, when taken together, break down muscle tissue?



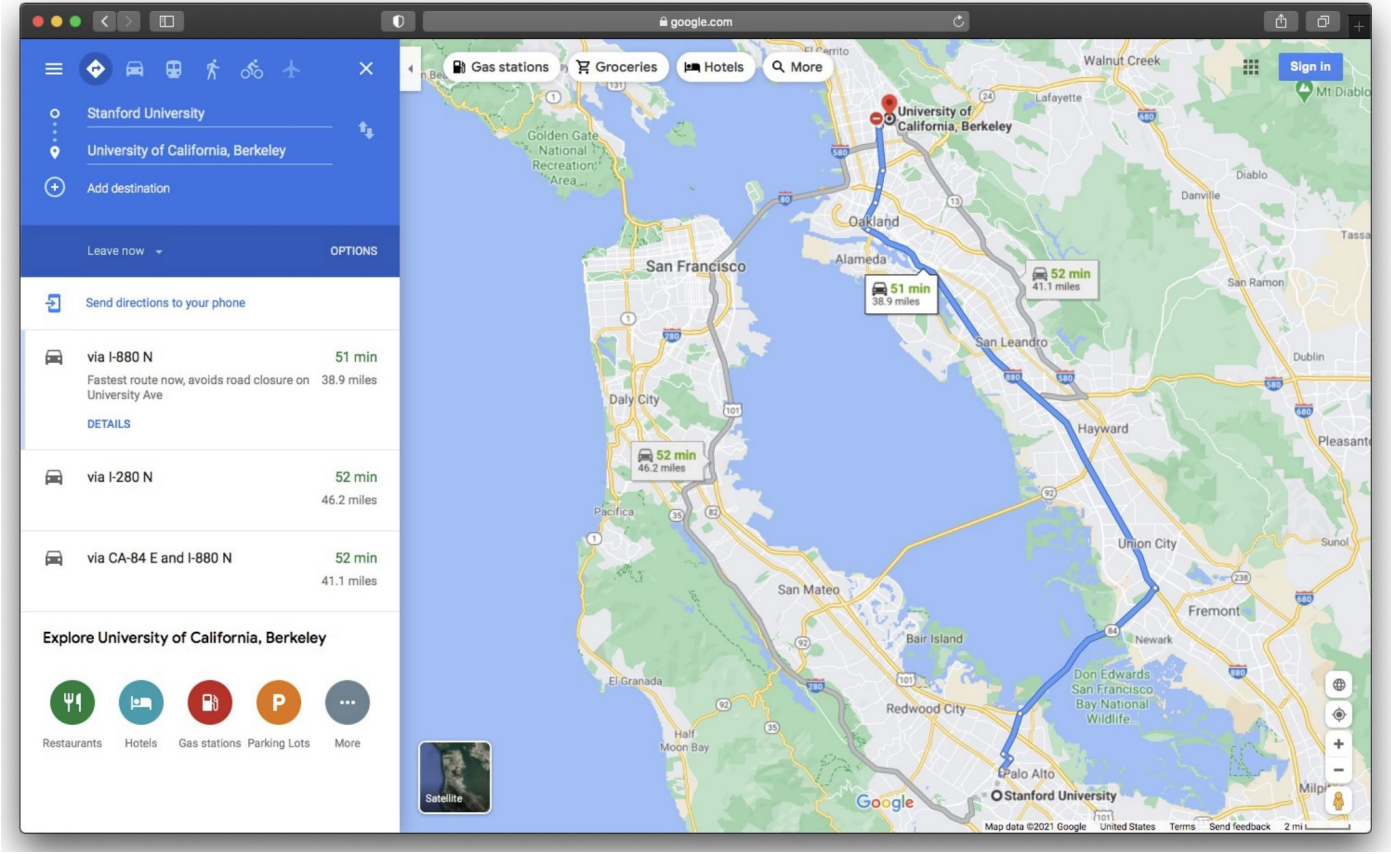
Graph-Level Prediction

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

- **For example:**



Example: Traffic Prediction



Example: Traffic Prediction

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

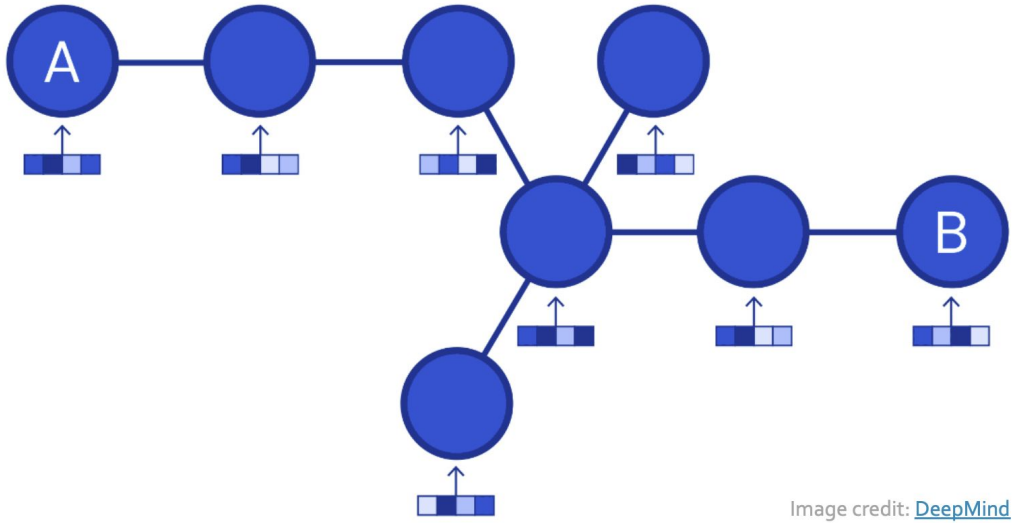
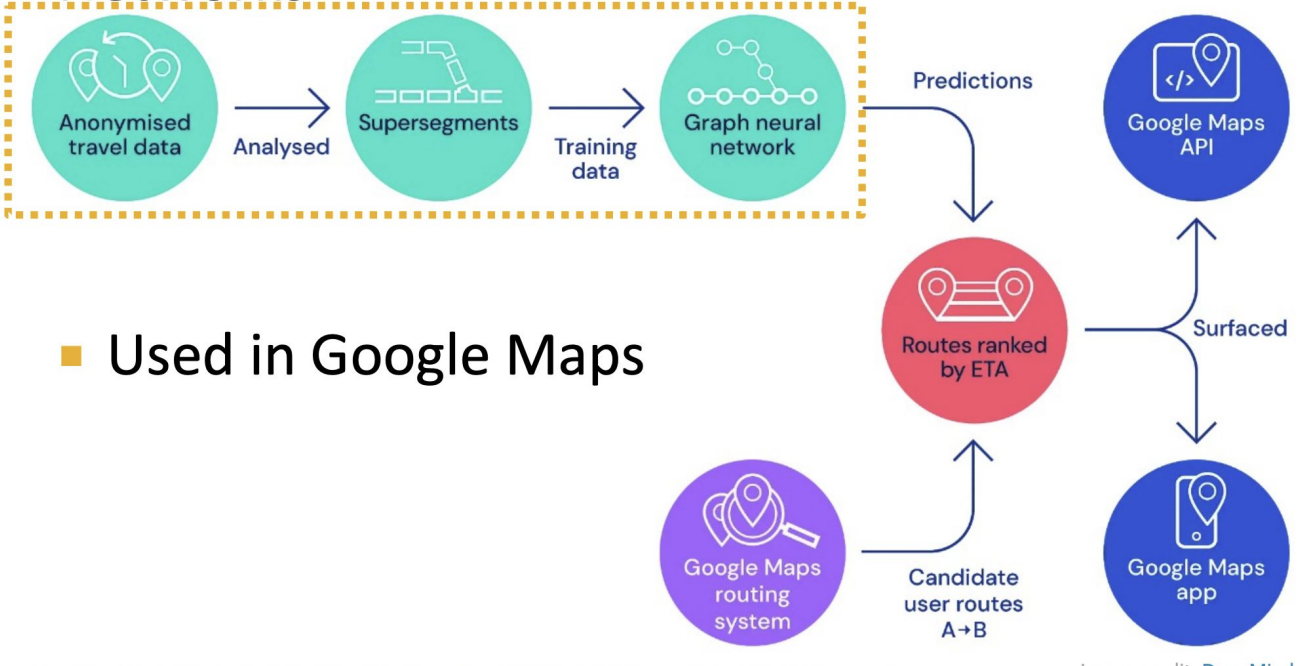


Image credit: DeepMind

Example: Traffic Prediction

Predicting Time of Arrival with Graph Neural Networks



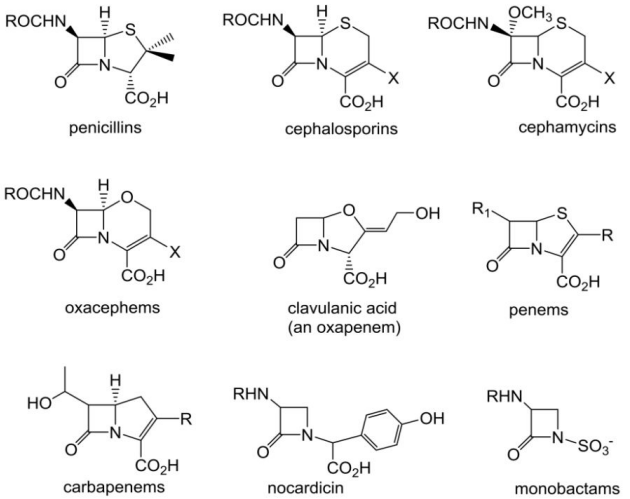
- Used in Google Maps

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

Image credit: DeepMind

Example: 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.

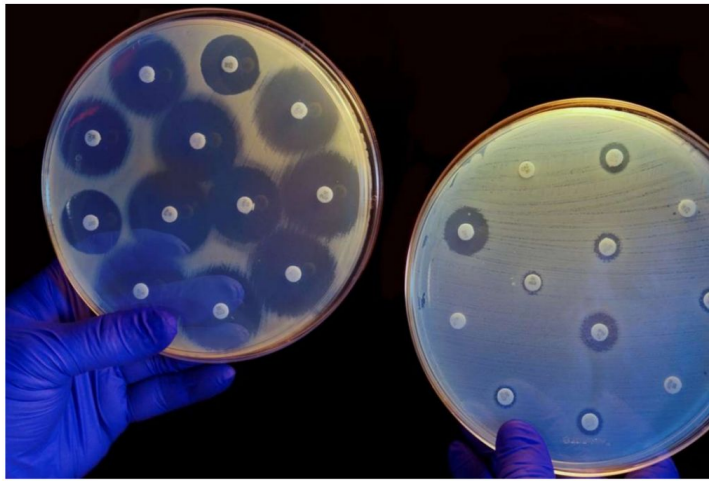
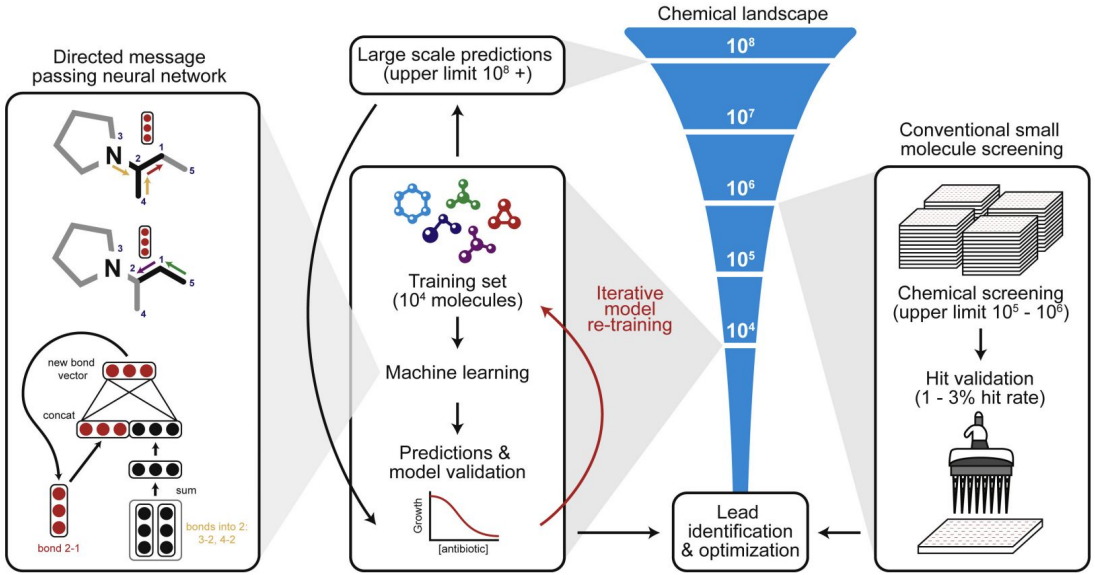


Image credit: CNN

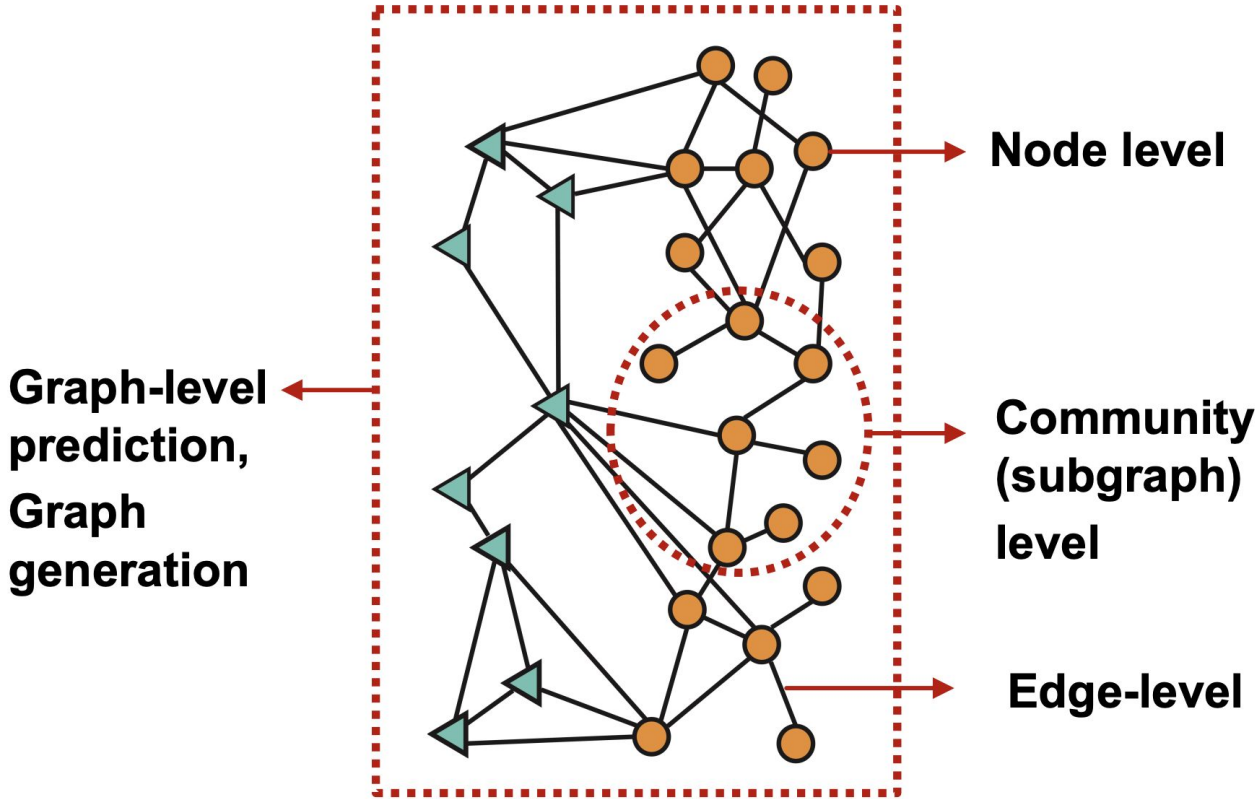
Example: Drug Discovery

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



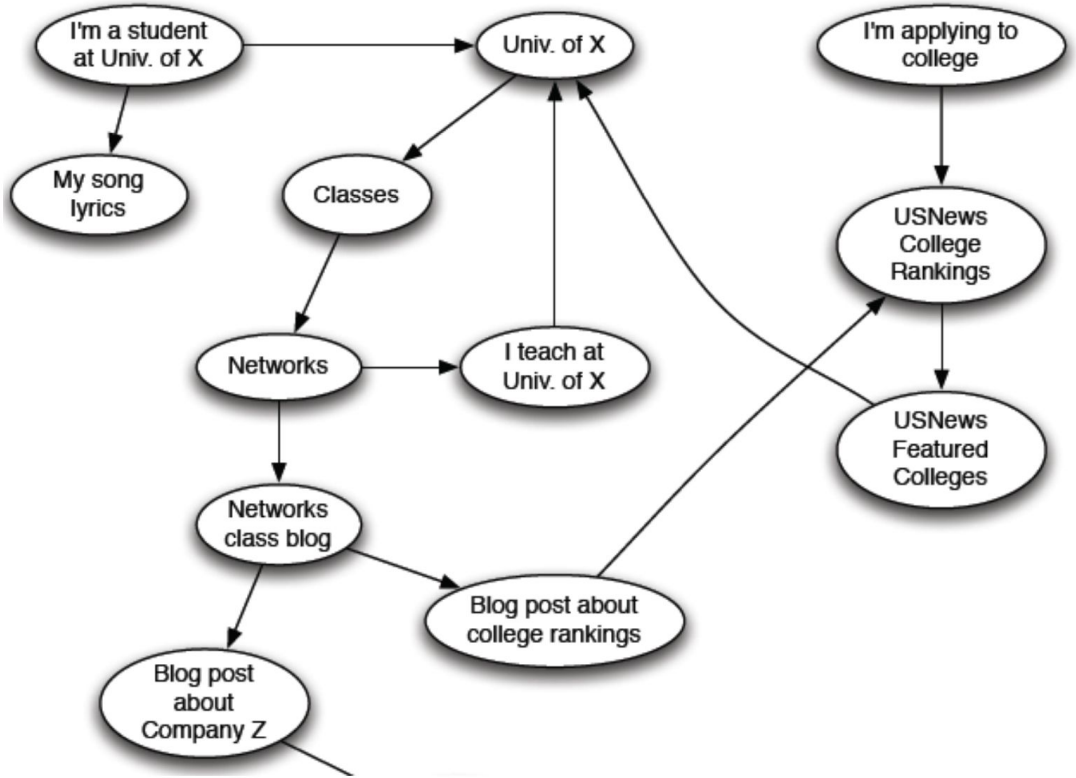
Stokes, Jonathan M., et al. "A deep learning approach to antibiotic discovery." Cell 180.4 (2020): 688-702.

Summary


Node level
Graph-level prediction, Graph generation
Community (subgraph) level
Edge-level

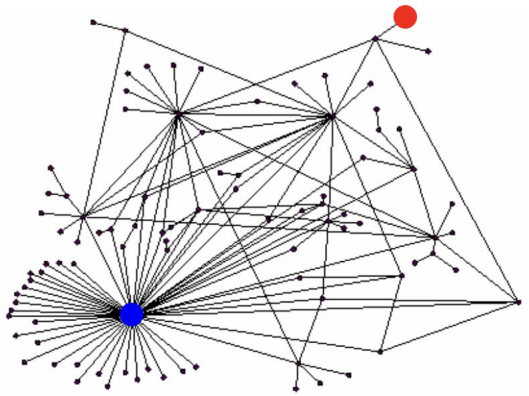
Random Walk and PageRank

The Web as a Directed Graph



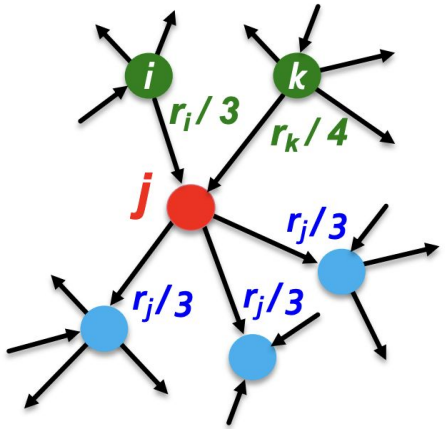
Rank Node Importance on Web

- Web pages are not equally “important”
  - www.cnn.com vs. a personal webpage
- Inlinks as votes
  - The more inlinks, the more important
- Are all inlinks equal?
  - Higher ranked inlink should play a more important role
  - Recursive question!



PageRank: Recursive Formulation

- **A "vote" from an important page is worth more:**
  - Each link's vote is proportional to the **importance** of its source page
  - If page *i* with importance r_i has d_i out-links, each link gets r_i / d_i votes
  - Page *j*'s own importance r_j is the sum of the votes on its in-links

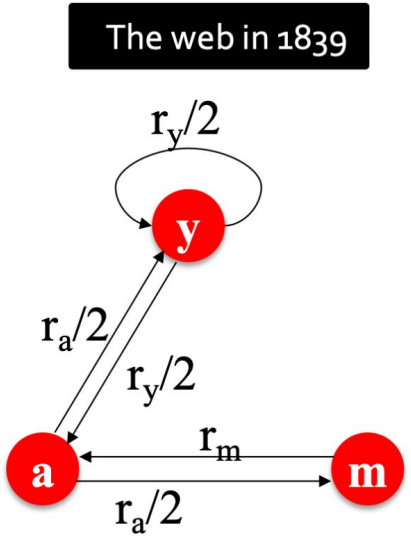


PageRank: Recursive Formulation

- **A page is important if it is pointed to by other important pages**
- **Define "rank" r_j for node j**

$$r_j = \sum_{i \to j} \frac{r_i}{d_i}$$

d_i … out-degree of node i



"Flow" equations:

$r_y = r_y/2 + r_a/2$

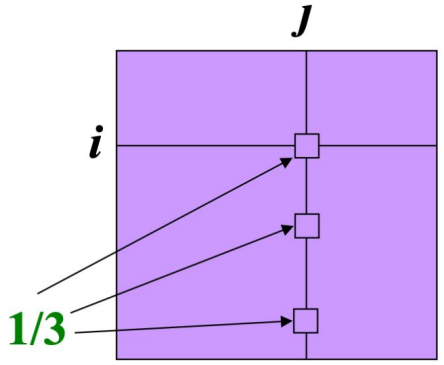
$r_a = r_y/2 + r_m$

$r_m = r_a/2$

You might wonder: Let's just use Gaussian elimination to solve this system of linear equations. Bad idea!

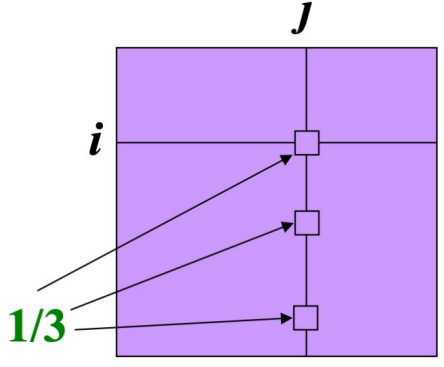
PageRank: Recursive Formulation

- **Stochastic adjacency matrix M**
  - Let page j have d_j out-links
  - If $j \rightarrow i$, then $M_{ij} = \frac{1}{d_j}$
    - M is a **column stochastic matrix**
      - **Columns** sum to **1**



PageRank: Recursive Formulation

- **Stochastic adjacency matrix M**
  - Let page j have d_j out-links
  - If $j \rightarrow i$, then $M_{ij} = \frac{1}{d_j}$
    - M is a **column stochastic matrix**
      - **Columns** sum to **1**



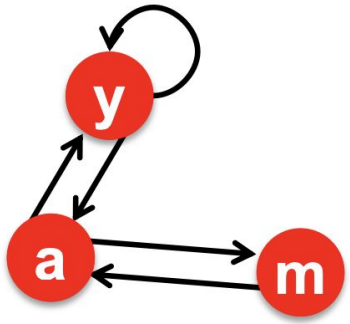
M

- **Rank vector r:** An entry per page
  - r_i is the importance score of page i
  - $\sum_i r_i = 1$
- **The flow equations can be written**

$$r = M \cdot r$$

$$r_j = \sum_{i \rightarrow j} \frac{r_i}{d_i}$$

Example



	r_y	r_a	r_m
r_y	½	½	0
r_a	½	0	1
r_m	0	½	0

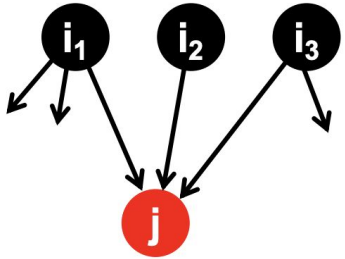
$$r_y = r_y/2 + r_a/2$$
$$r_a = r_y/2 + r_m$$
$$r_m = r_a/2$$

$$\begin{bmatrix} r_y \\ r_a \\ r_m \end{bmatrix} = \begin{bmatrix} ½ & ½ & 0 \\ ½ & 0 & 1 \\ 0 & ½ & 0 \end{bmatrix} \begin{bmatrix} r_y \\ r_a \\ r_m \end{bmatrix}$$

$$r \qquad M \qquad r$$

Connection to Random Walk

- **Imagine a random web surfer:**
  - At any time t, surfer is on some page i
  - At time $t+1$, the surfer follows an out-link from i uniformly at random
  - Ends up on some page j linked from i
  - Process repeats indefinitely

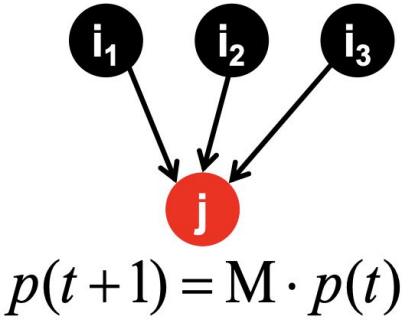


- **Let:**
  - $p(t)$ … vector whose i^{th} coordinate is the prob. that the surfer is at page i at time t
  - So, $p(t)$ is a probability distribution over pages

Stationary Distribution of Random Walk

- **Where is the surfer at time $t+1$?**
  - Follow a link uniformly at random

$$\boldsymbol{p(t+1) = M \cdot p(t)}$$



- Suppose the random walk reaches a state

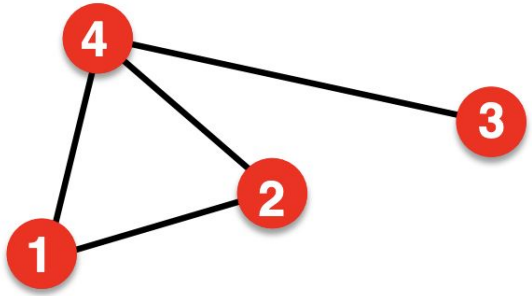
$$\boldsymbol{p(t+1) = M \cdot p(t) = p(t)}$$

then $\boldsymbol{p(t)}$ is **stationary distribution** of a random walk

- **Our original rank vector $\boldsymbol{r}$** satisfies $\boldsymbol{r = M \cdot r}$
  - **So, $\boldsymbol{r}$ is a stationary distribution for the random walk**

Recall Eigenvector of a Matrix

$\boldsymbol{A \in \{0,1\}^{n \times n}}$ be an adj. matrix of undir. graph:



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

- Eigenvector of adjacency matrix: vectors satisfying $\boldsymbol{\lambda c = Ac}$
- $\boldsymbol{c}$: eigenvector; $\boldsymbol{\lambda}$: eigenvalue

- **The flow equation:**

$$1 \cdot \boldsymbol{r} = \boldsymbol{M} \cdot \boldsymbol{r}$$

$$\begin{bmatrix} r_y \\ r_a \\ r_m \end{bmatrix} = \begin{bmatrix} ½ & ½ & 0 \\ ½ & 0 & 1 \\ 0 & ½ & 0 \end{bmatrix} \begin{bmatrix} r_y \\ r_a \\ r_m \end{bmatrix}$$

$\boldsymbol{r}$ $\boldsymbol{M}$ $\boldsymbol{r}$

- So the **rank vector** $\boldsymbol{r}$ is an **eigenvector** of the stochastic ajd. matrix $\boldsymbol{M}$ (with eigenvalue 1)

- **The flow equation:**

$$1 \cdot \boldsymbol{r} = \boldsymbol{M} \cdot \boldsymbol{r}$$

$$\begin{bmatrix} r_y \\ r_a \\ r_m \end{bmatrix} = \begin{bmatrix} ½ & ½ & 0 \\ ½ & 0 & 1 \\ 0 & ½ & 0 \end{bmatrix} \begin{bmatrix} r_y \\ r_a \\ r_m \end{bmatrix}$$

$$\boldsymbol{r} \qquad \boldsymbol{M} \qquad \boldsymbol{r}$$

- So the **rank vector** $\boldsymbol{r}$ is an **eigenvector** of the stochastic ajd. matrix $\boldsymbol{M}$ (with eigenvalue 1)
  - Starting from any vector $\boldsymbol{u}$, the limit $\boldsymbol{M}(\boldsymbol{M}(\ldots \boldsymbol{M}(\boldsymbol{M}\,\boldsymbol{u})))$ is the **long-term distribution** of the surfers.
    - **PageRank** = Limiting distribution = **principal eigenvector** of M
    - Note: If $\boldsymbol{r}$ is the limit of the product $\boldsymbol{MM} \ldots \boldsymbol{Mu}$, then $\boldsymbol{r}$ satisfies the **flow equation** $1 \cdot \boldsymbol{r} = \boldsymbol{Mr}$
    - So $\boldsymbol{r}$ is the **principal eigenvector** of $\boldsymbol{M}$ with eigenvalue 1
- **We can now efficiently solve for *r*!**
  - The method is called **Power iteration**

Solve PageRank: Power Iteration

Given a graph with *n* nodes, we use an iterative procedure:

- Assign each node an initial page rank
- Repeat until convergence ($\sum_i \left| r_i^{t+1} - r_i^t \right| < \epsilon$)
  - Calculate the page rank of each node

$$r_j^{(t+1)} = \sum_{i \to j} \frac{r_i^{(t)}}{d_i}$$

d_i …. out-degree of node i

- **Given a web graph with *N* nodes, where the nodes are pages and edges are hyperlinks**
- **Power iteration:** a simple iterative scheme
  - Initialize: $\boldsymbol{r}^0 = [1/N, \ldots, 1/N]^T$
  - Iterate: $\boldsymbol{r}^{(t+1)} = \boldsymbol{M} \cdot \boldsymbol{r}^t$
  - Stop when $|\boldsymbol{r}^{(t+1)} - \boldsymbol{r}^t|_1 < \varepsilon$

$$r_j^{(t+1)} = \sum_{i \to j} \frac{r_i^{(t)}}{d_i}$$

d_i …. out-degree of node i

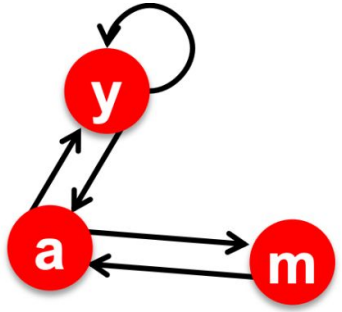
$|x|_1 = \sum_1^N |x_1|$ is the **L1** norm
Can use any other vector norm, e.g., Euclidean

About 50 iterations is sufficient to estimate the limiting solution.

- **Power Iteration:**
  - Set $r_j \leftarrow 1/N$
  - **1:** $r'_j \leftarrow \sum_{i \to j} \frac{r_i}{d_i}$
  - **2:** If $|r - r'| > \varepsilon$:
    - $r \leftarrow r'$
  - **3:** go to **1**
- **Example:**

$$\begin{bmatrix} r_y \\ r_a \\ r_m \end{bmatrix} = \begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix}$$

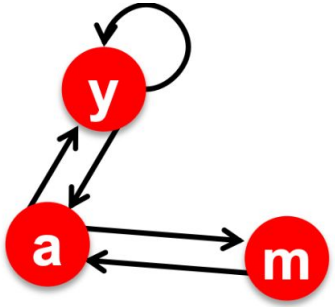
Iteration 0, 1, 2, …



	y	a	m
y	½	½	0
a	½	0	1
m	0	½	0

$$r_y = r_y/2 + r_a/2$$
$$r_a = r_y/2 + rm$$
$$r_m = r_a/2$$

- **Power Iteration:**
  - Set $r_j \leftarrow 1/N$
  - **1:** $r'_j \leftarrow \sum_{i \to j} \frac{r_i}{d_i}$
  - **2:** If $|r - r'| > \varepsilon$:
    - $r \leftarrow r'$
  - **3:** go to **1**



	y	a	m
y	½	½	0
a	½	0	1
m	0	½	0

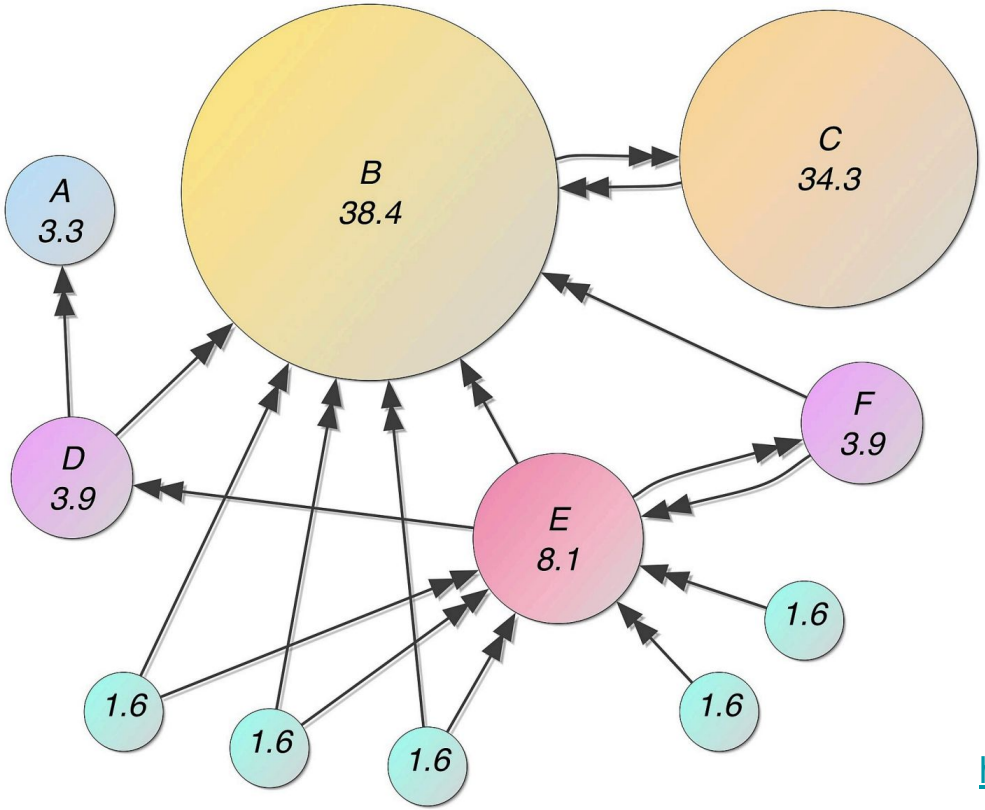
$$r_y = r_y/2 + r_a/2$$
$$r_a = r_y/2 + rm$$
$$r_m = r_a/2$$

- **Example:**

$$\begin{pmatrix} r_y \\ r_a \\ r_m \end{pmatrix} = \begin{pmatrix} 1/3 \\ 1/3 \\ 1/3 \end{pmatrix} \begin{pmatrix} 1/3 \\ 3/6 \\ 1/6 \end{pmatrix} \begin{pmatrix} 5/12 \\ 1/3 \\ 3/12 \end{pmatrix} \begin{pmatrix} 9/24 \\ 11/24 \\ 1/6 \end{pmatrix} \ldots \begin{pmatrix} 6/15 \\ 6/15 \\ 3/15 \end{pmatrix}$$

Iteration 0, 1, 2, …

PageRank Example



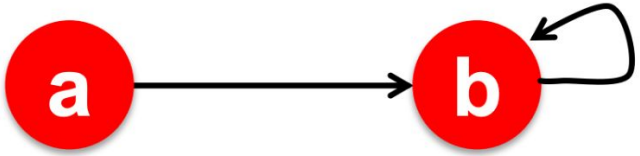
https://en.wikipedia.org/wiki/PageRank

Problem for PageRank

Two problems:

- **(1)** Some pages are **dead ends** (have no out-links)
  - Such pages cause importance to "leak out"
- **(2) Spider traps** (all out-links are within the group)
  - Eventually spider traps absorb all importance

- **The "Spider trap" problem:**

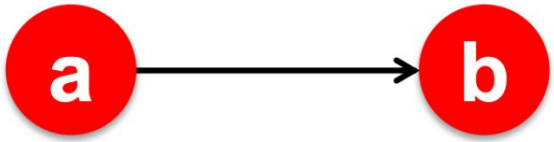


$$r_j^{(t+1)} = \sum_{i \to j} \frac{r_i^{(t)}}{d_i}$$

- **Example:**

Iteration:	0,	1,	2,	3…
r_a =	1	0	0	0
r_b =	0	1	1	1

- **The "Dead end" problem:**



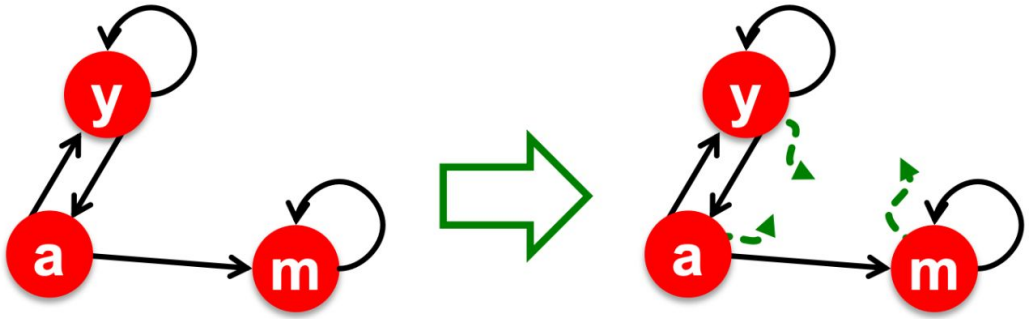
$$r_j^{(t+1)} = \sum_{i \to j} \frac{r_i^{(t)}}{d_i}$$

- **Example:**

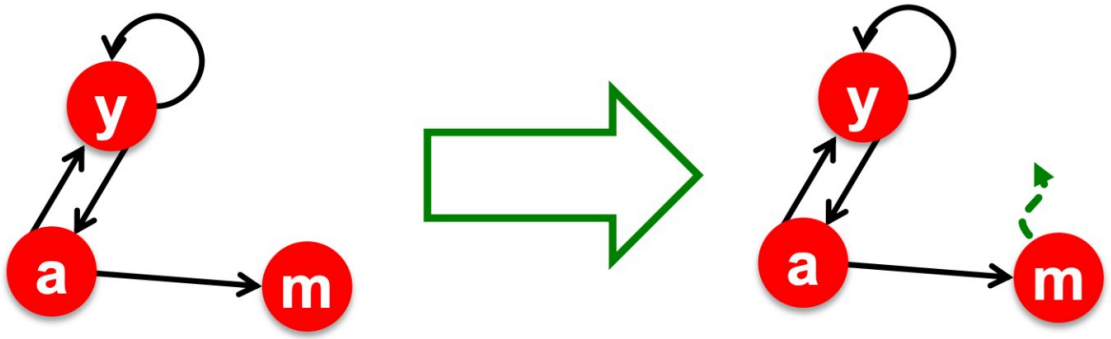
Iteration:	0,	1,	2,	3…
r_a	1	0	0	0
r_b	0	1	0	0

=

- **Solution for spider traps: At each time step, the random surfer has two options**
  - With prob. β, follow a link at random
  - With prob. **1-β,** jump to a random page
  - Common values for β are in the range 0.8 to 0.9
- **Surfer will teleport out of spider trap within a few time steps**



- **Teleports:** Follow random teleport links with total probability **1.0** from dead-ends
  - Adjust matrix accordingly



	y	a	m
y	½	½	0
a	½	0	0
m	0	½	0

	y	a	m
y	½	½	⅓
a	½	0	⅓
m	0	½	⅓

Why are dead-ends and spider traps a problem and why do teleports solve the problem?

- **Spider-traps** are not a problem, but with traps PageRank scores are **not** what we want
  - **Solution:** Never get stuck in a spider trap by teleporting out of it in a finite number of steps
- **Dead-ends** are a problem
  - The matrix is not column stochastic so our initial assumptions are not met
  - **Solution:** Make matrix column stochastic by always teleporting when there is nowhere else to go

Solution: Random Teleports

- **Google's solution that does it all:**
  At each step, random surfer has two options:
  - With probability β, follow a link at random
  - With probability $1\text{-}\beta$, jump to some random page

- **PageRank equation** [Brin-Page, 98]

$$r_j = \sum_{i \to j} \beta \frac{r_i}{d_i} + (1 - \beta)\frac{1}{N}$$

d_i … out-degree of node i

This formulation assumes that M has no dead ends. We can either preprocess matrix M to remove all dead ends or explicitly follow random teleport links with probability 1.0 from dead-ends.

The Google Matrix

- **PageRank equation** [Brin-Page, '98]

$$r_j = \sum_{i \to j} \beta \frac{r_i}{d_i} + (1 - \beta) \frac{1}{N}$$

- **The Google Matrix *G*:**

$[1/N]_{NxN}$…N by N matrix where all entries are 1/N

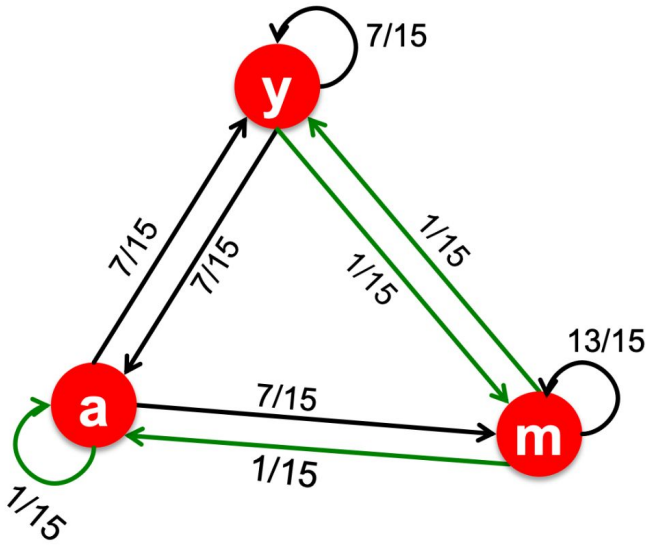
$$G = \beta\, M + (1 - \beta) \left[\frac{1}{N}\right]_{N \times N}$$

- **We have a recursive problem: $\boldsymbol{r = G \cdot r}$**

  And the Power method still works!

- **What is β?**
  - In practice $\beta = 0.8, 0.9$ (make 5 steps on avg., jump)

Random Teleports (beta = 0.8)



M

$$0.8 \begin{bmatrix} 1/2 & 1/2 & 0 \\ 1/2 & 0 & 0 \\ 0 & 1/2 & 1 \end{bmatrix}$$

$[1/N]_{NxN}$

$$+ 0.2 \begin{bmatrix} 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \end{bmatrix}$$

G

$$\begin{array}{c} \\ y \\ a \\ m \end{array} \begin{bmatrix} 7/15 & 7/15 & 1/15 \\ 7/15 & 1/15 & 1/15 \\ 1/15 & 7/15 & 13/15 \end{bmatrix}$$

$$\begin{array}{c} y \\ a \\ m \end{array} = \begin{array}{c} 1/3 \\ 1/3 \\ 1/3 \end{array} \begin{array}{c} 0.33 \\ 0.20 \\ 0.46 \end{array} \begin{array}{c} 0.24 \\ 0.20 \\ 0.52 \end{array} \begin{array}{c} 0.26 \\ 0.18 \\ 0.56 \end{array} \quad \ldots \quad \begin{array}{c} 7/33 \\ 5/33 \\ 21/33 \end{array}$$

Summary of PageRank

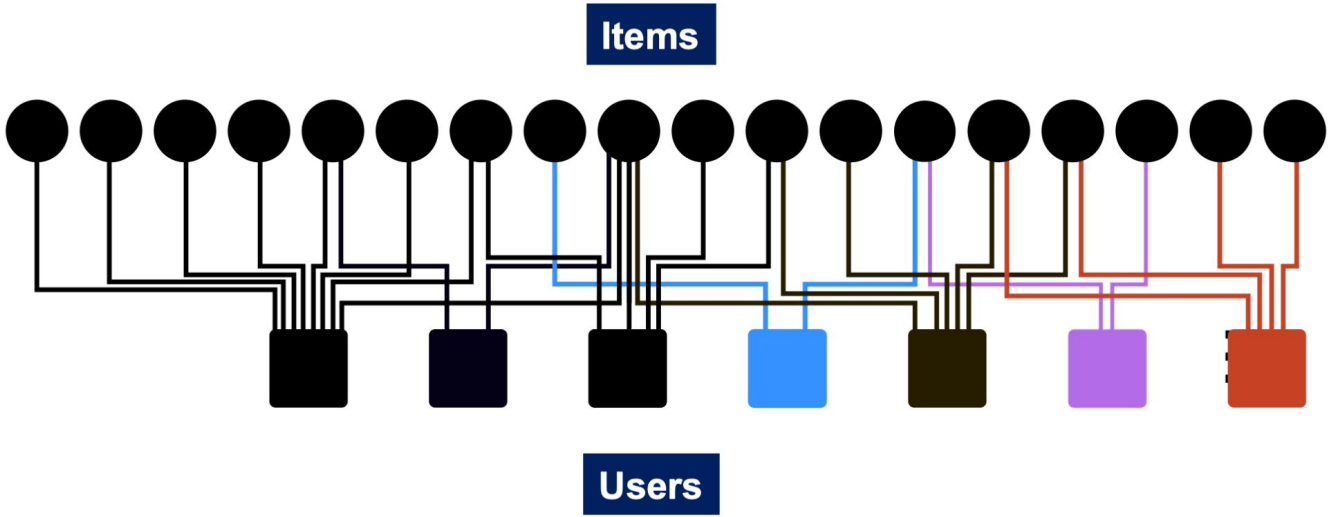
- PageRank solves for $\boldsymbol{r} = \boldsymbol{Gr}$ and can be efficiently computed by power iteration of the stochastic adjacency matrix ($\boldsymbol{G}$)
- Adding random uniform teleportation solves issues of dead-ends and spider-traps

Personalized PageRank

- Query-dependent Ranking
  - For a query webpage u, which webpages are most important to u?
    - We need a measure s(u,v)
  - The relative important webpages to different queries would be different

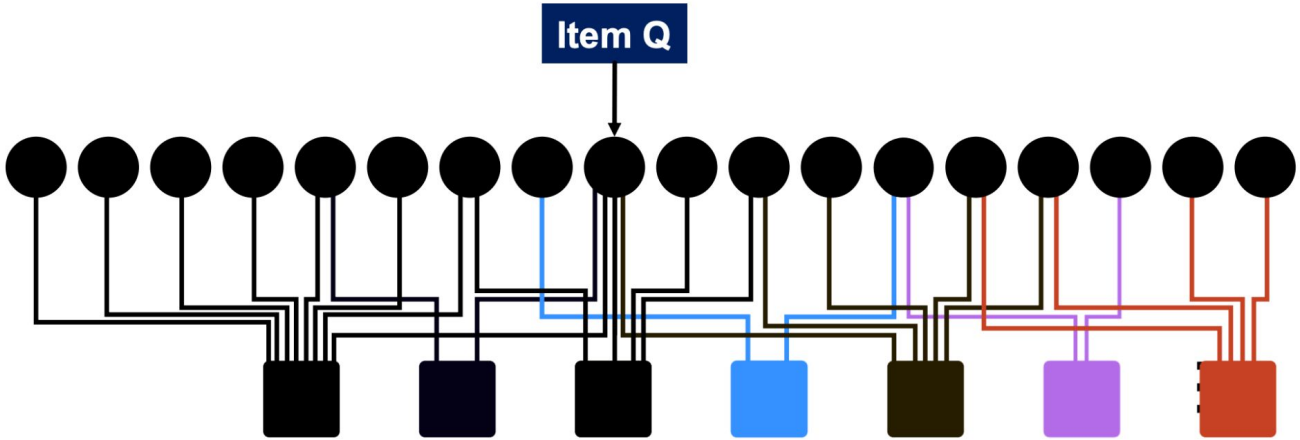
Example: Recommendation

- **Given:**
  A bipartite graph representing user and item interactions (e.g. purchase)



Example: Recommendation

- **Goal:** Proximity on graphs
  - **What items should we recommend to a user who interacts with item Q?**
  - **Intuition**: if items Q and P are interacted by similar users, recommend P when user interacts with Q



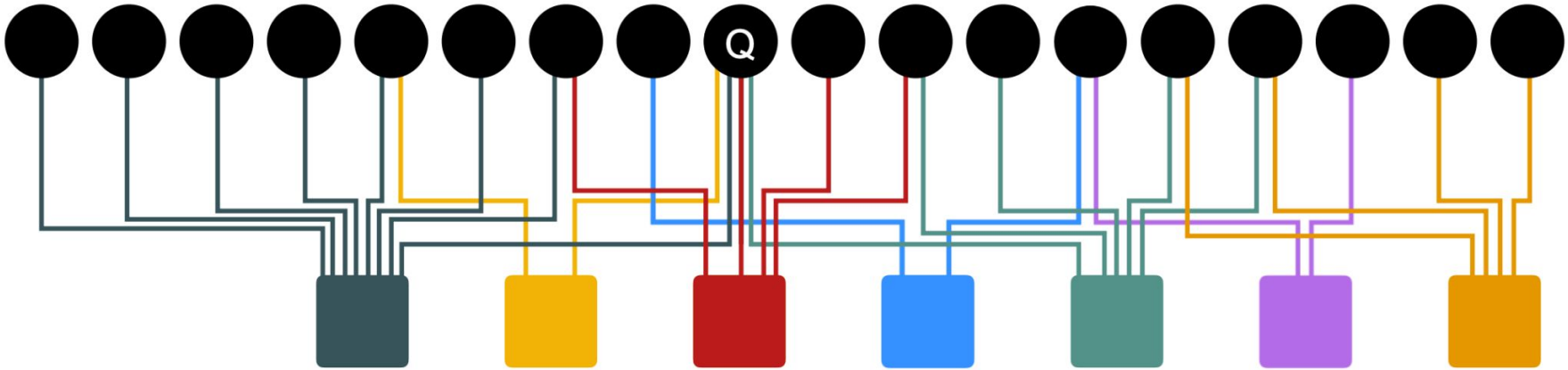
Random Walk with Restarts

- **Idea**
  - Every node has some importance
  - Importance gets evenly split among all edges and pushed to the neighbors:
- **Given a set of QUERY_NODES, we simulate a random walk:**
  - Make a step to a random neighbor and record the visit (visit count)
  - With probability ALPHA, restart the walk at one of the QUERY_NODES
  - The nodes with the highest visit count have highest proximity to the QUERY_NODES

- Proximity to query node(s) ***Q***:

```
ALPHA = 0.5
QUERY_NODES = { Q }
```

```
item = QUERY_NODES.sample_by_weight( )
for i in range( N_STEPS ):
    user = item.get_random_neighbor( )
    item = user.get_random_neighbor( )
    item.visit_count += 1
    if random( ) < ALPHA:
        item = QUERY_NODES.sample.by_weight ( )
```

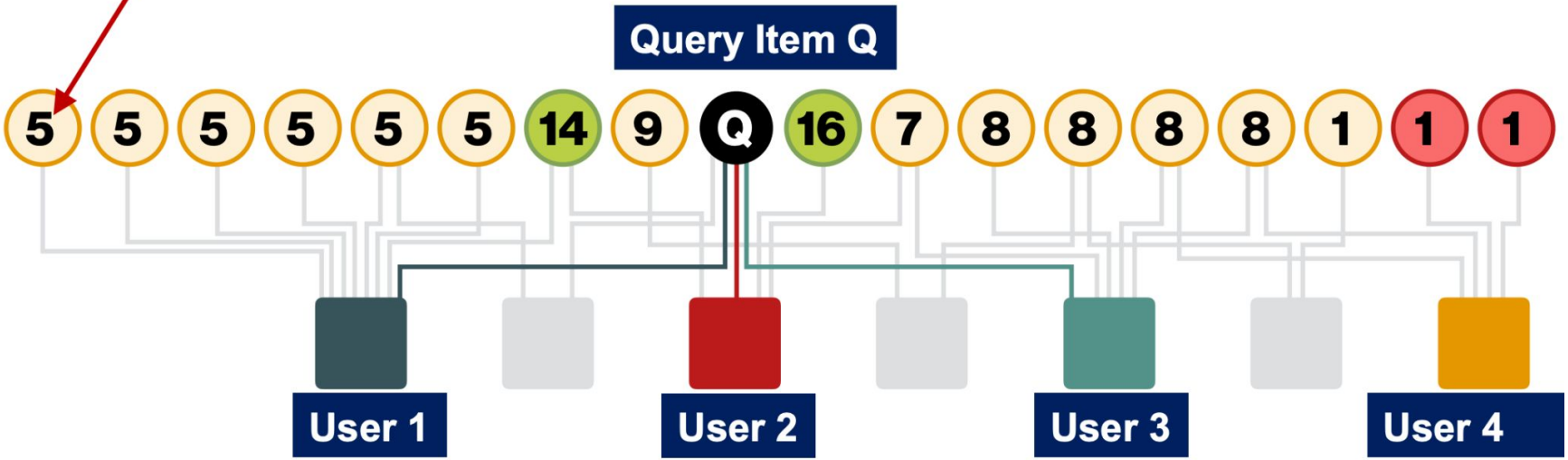


- Proximity to query node(s) ***Q***:

```
ALPHA = 0.5
QUERY_NODES = { Q }
```

```
item = QUERY_NODES.sample_by_weight( )
for i in range( N_STEPS ):
    user = item.get_random_neighbor( )
    item = user.get_random_neighbor( )
    item.visit_count += 1
    if random( ) < ALPHA:
        item = QUERY_NODES.sample.by_weight ( )
```

Number of visits by random walks starting at Q



Calculation of P-PageRank

- Recall PageRank calculation:
  - $\mathbf{r} = \beta\mathbf{M}\mathbf{r} + [(1-\beta)/N]_N$ or
  - $\mathbf{r} = \beta\mathbf{M}\mathbf{r} + (1-\beta)\ q$, where $q = \begin{pmatrix} 1/N \\ 1/N \\ \dots \\ 1/N \end{pmatrix}$
- For P-PageRank, $s(u,v) = \mathbf{r}_u(v)$, where $\mathbf{r}_u = \beta\mathbf{M}\ \mathbf{r}_u + (1-\beta)\ q$

  by replacing q with $q = \begin{pmatrix} 0 \\ 0 \\ \dots \\ 1 \\ \dots \\ 0 \end{pmatrix}$ ← uth webpage

  - Teleport to webpage u

Summary

- A graph is naturally represented as a matrix
- We defined a random walk process over the graph
  - Random surfer moving across the links and with random teleportation
  - Stochastic adjacency matrix M
- PageRank = Limiting distribution of the surfer location represented node importance
  - Corresponds to the leading eigenvector of transformed adjacency matrix M.