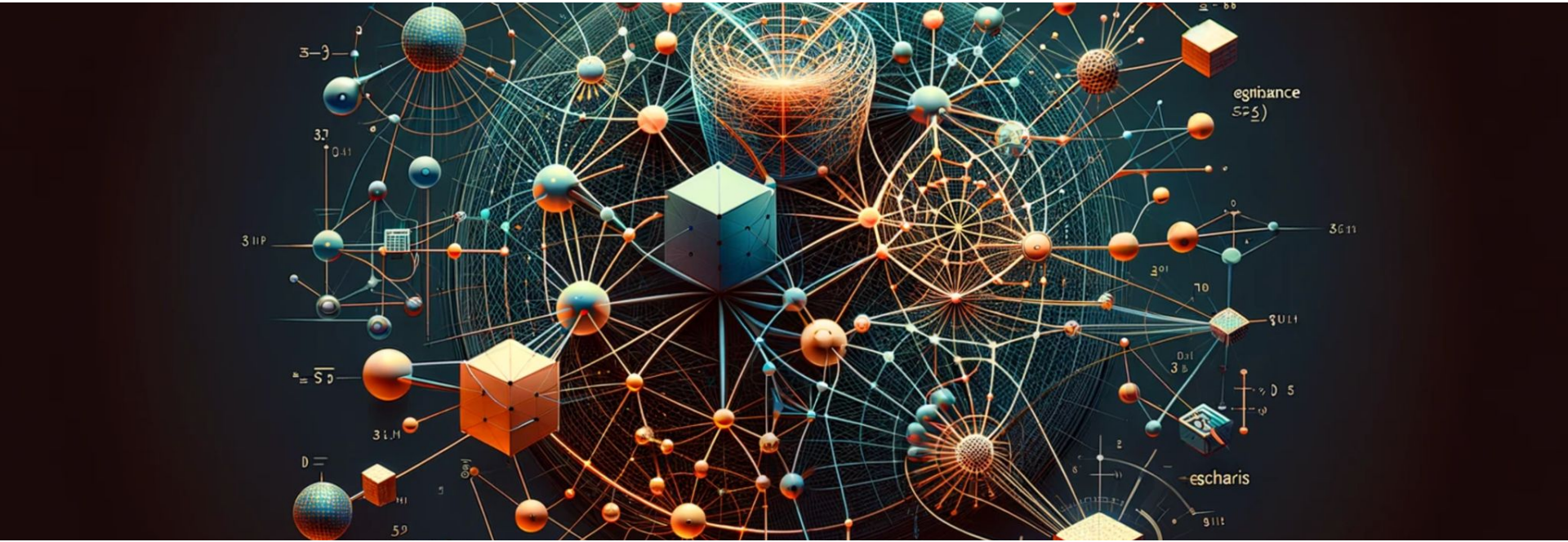
# CSE7850/CX4803 Machine Learning in Computational Biology

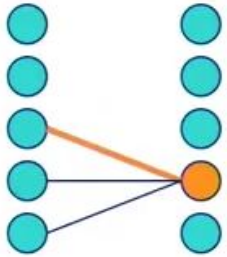



## Lecture 14: Graph Neural Networks

Yunan Luo

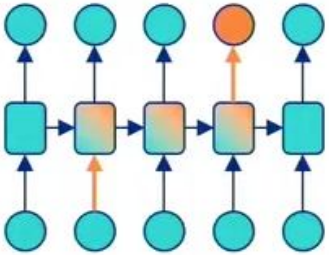
Acknowledgements: Some of the slides were adapted from CS224W, Jure Leskovec

# Graph Neural Networks (GNN)



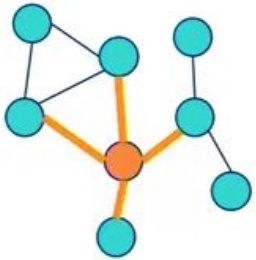
Convolutional Networks
(e.g. computer vision)

- data in regular grid
- information flow to local neighbours
- AlphaFold 1



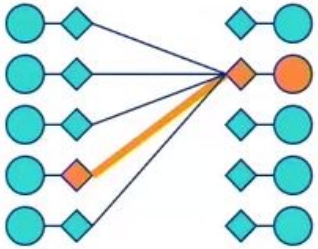
Recurrent Networks
(e.g. language)

- data in ordered sequence
- information flow sequentially



Graph Networks (e.g. recommender systems or molecules)

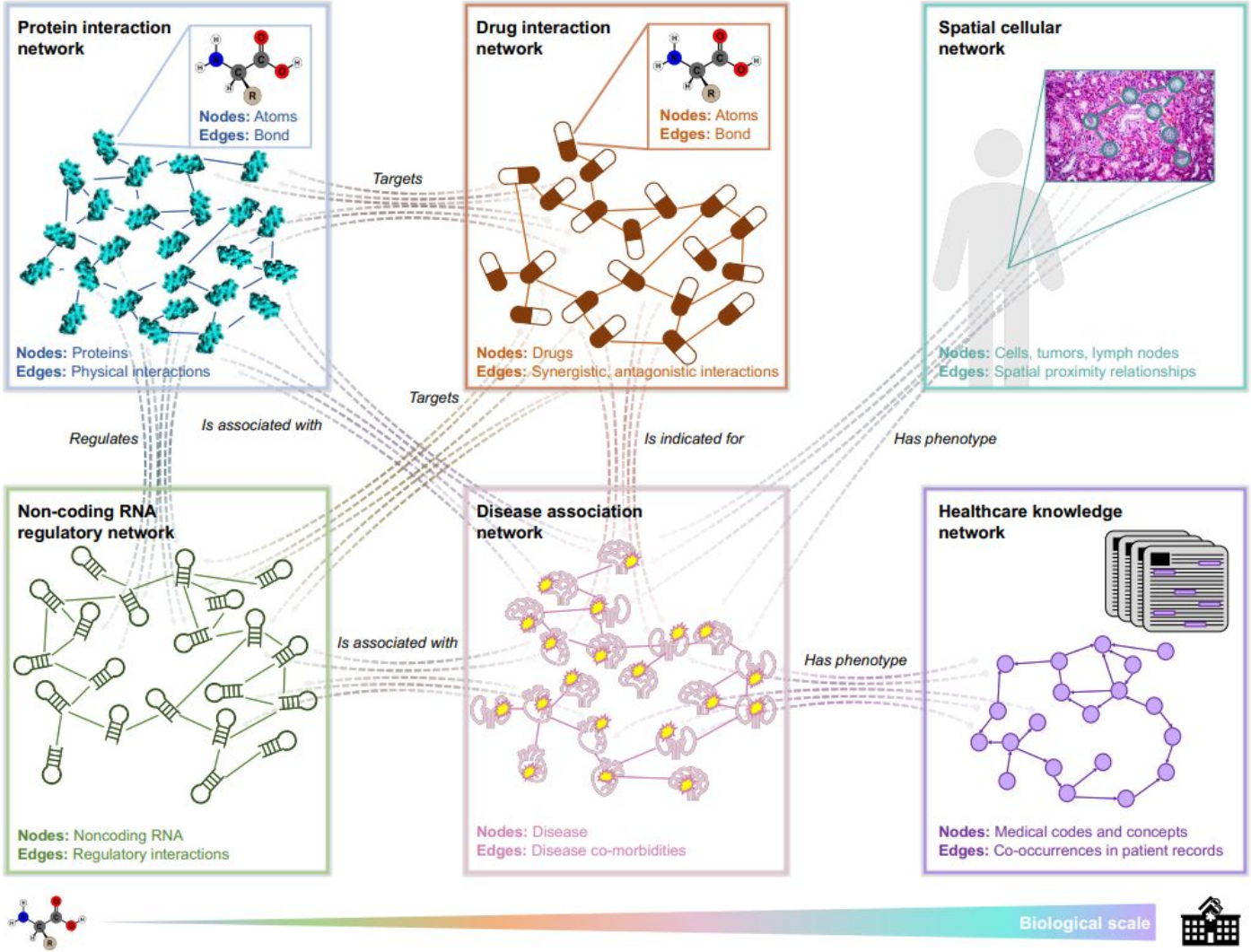
- data in fixed graph structure
- information flow along fixed edges



**Transformers** (e.g. language)

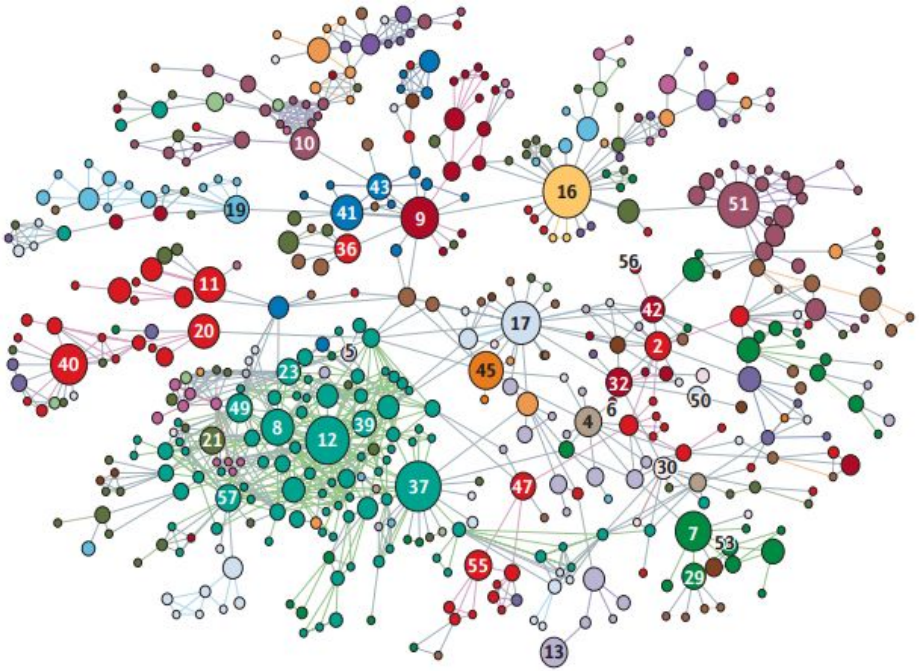
- data in unordered set
- information flow dynamically controlled by the network (via keys and queries)

# Why networks in biology?




Li et al, “Graph Representation Learning in Biomedicine”, 2021

# Human disease network



Node: protein
Edge: protein-protein interaction

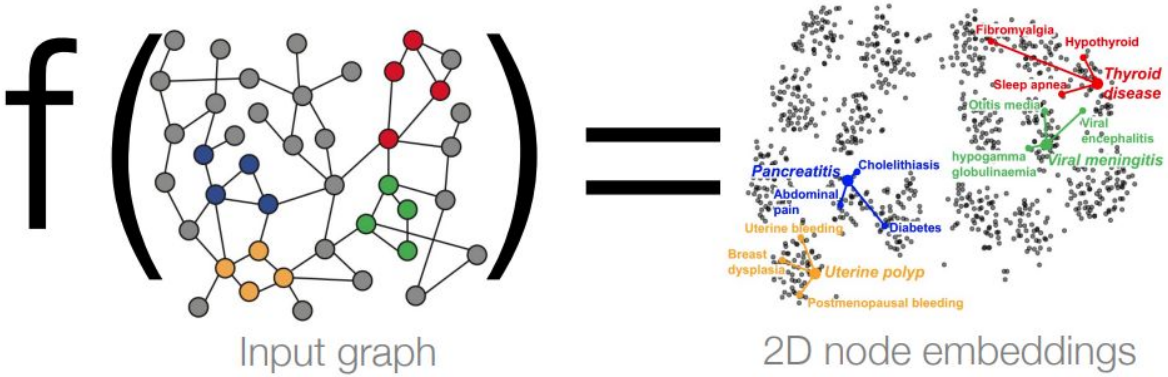
1. Aldosteronism
2. Alzheimer's disease
3. Anaemia, congenital deserythropoietic
4. Asthma
5. Ataxia-telangiectasia
6. Atherosclerosis
7. Blood group
8. Breast cancer
9. Cardiomyopathy
10. Cataract
11. Charcot–Marie–Tooth disease
12. Colon cancer
13. Complement component deficiency
14. Coronary artery disease
15. Coronary spasm
16. Deafness
17. Diabetes mellitus
18. Enolase-β deficiency
19. Epidermolysis bullosa
20. Epilepsy
21. Fanconi's anaemia
22. Fatty liver
23. Gastric cancer
24. Gilbert's syndrome
25. Glaucoma 1A
26. Goitre congenital
27. HARP syndrome
28. HELLP syndrome
29. Haemolytic anaemia
30. Hirschprung disease
31. Hyperbilirubinaemia
32. Hypertension
33. Hypertension diastolic
34. Hyperthyroidism
35. Hypoaldosteronism
36. Leigh syndrome
37. Leukaemia
38. Low renin hypertension
39. Lymphoma
40. Mental retardation
41. Muscular dystrophy
42. Myocardial infarction
43. Myopathy
44. Nucleoside phosphorylase deficiency
45. Obesity
46. Paraganglioma
47. Parkinson's disease
48. Pheochromocytoma
49. Prostate cancer
50. Pseudohypoaldosteronism
51. Retinitis pigmentosa
52. Schizoaffective disorder
53. Spherocytosis
54. Spina bifida
55. Spinocerebellar ataxia
56. Stroke
57. Thyroid carcinoma
58. Total iodide organification defect
59. Trifunctional protein deficiency
60. Unipolar depression

Proteins involved in the same disease have an increased tendency to interact with each other

Barabási el al, "Network medicine: a network-based approach to human disease", 2011

# Recap: Network embeddings

- **Idea**: Map nodes to d-dimensional embeddings such that similar nodes in the graph are embedded close together




**How to <u>learn</u> mapping function $f$?**

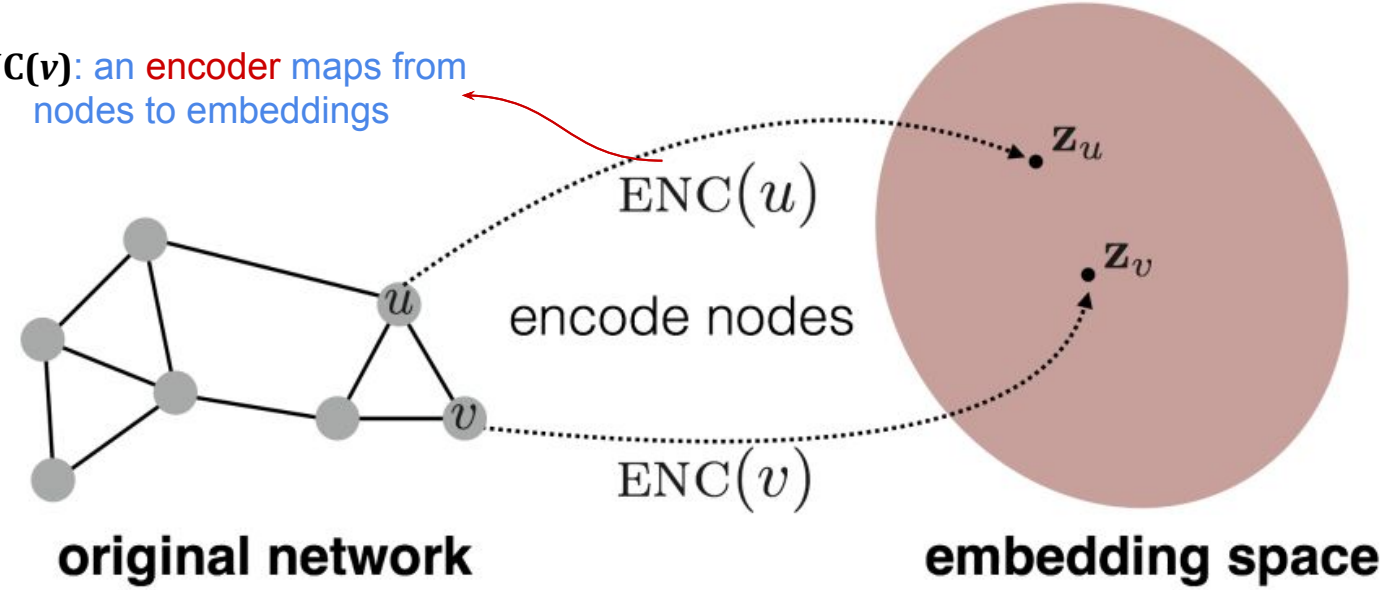
# Recap: Embedding nodes

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

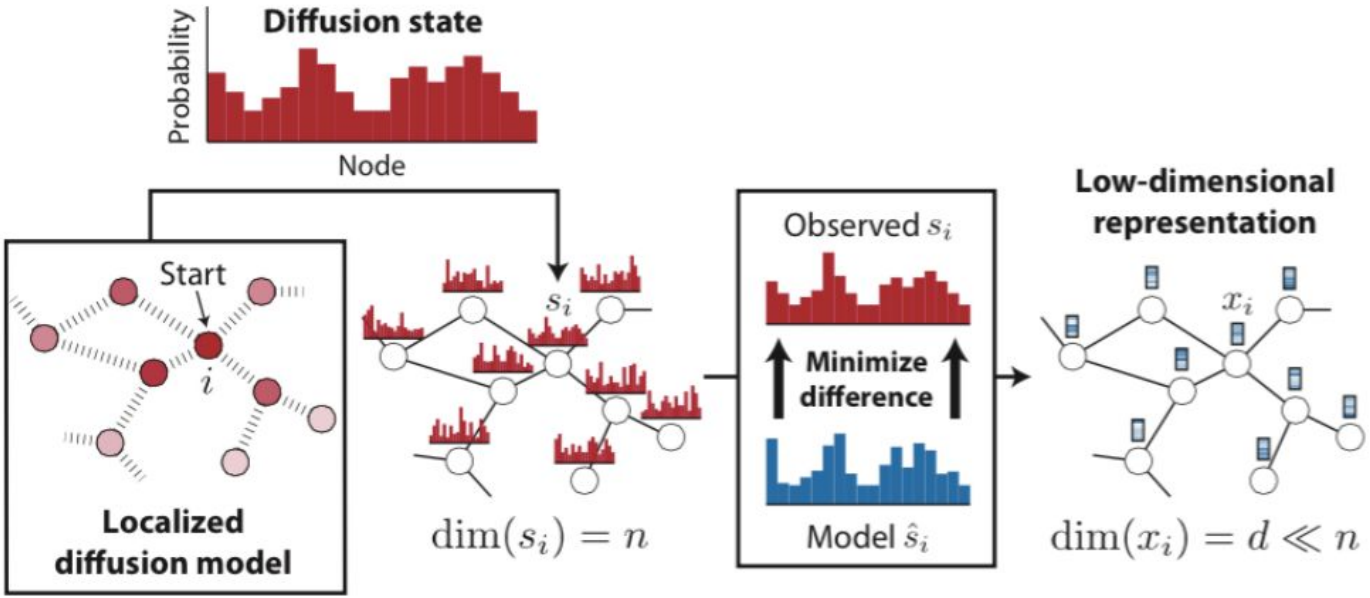
In the original network

Similarity of the embeddings

$\mathbf{ENC}(v)$: an encoder maps from nodes to embeddings



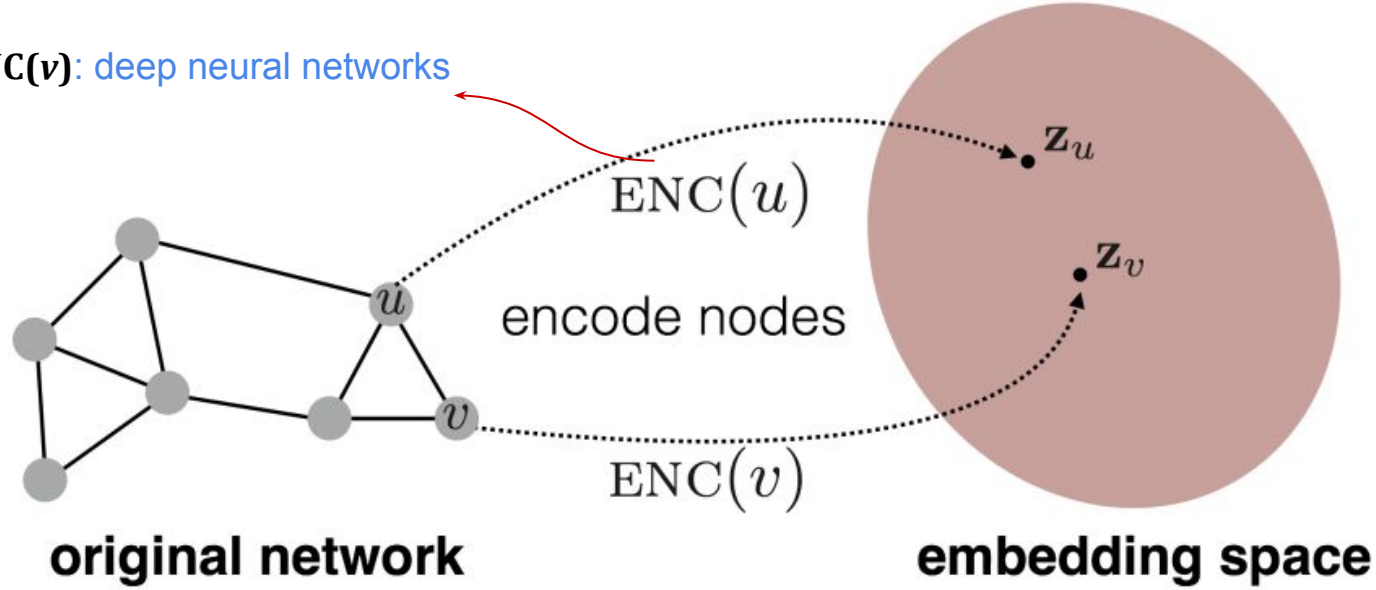
# Recap: Diffusion-based approaches



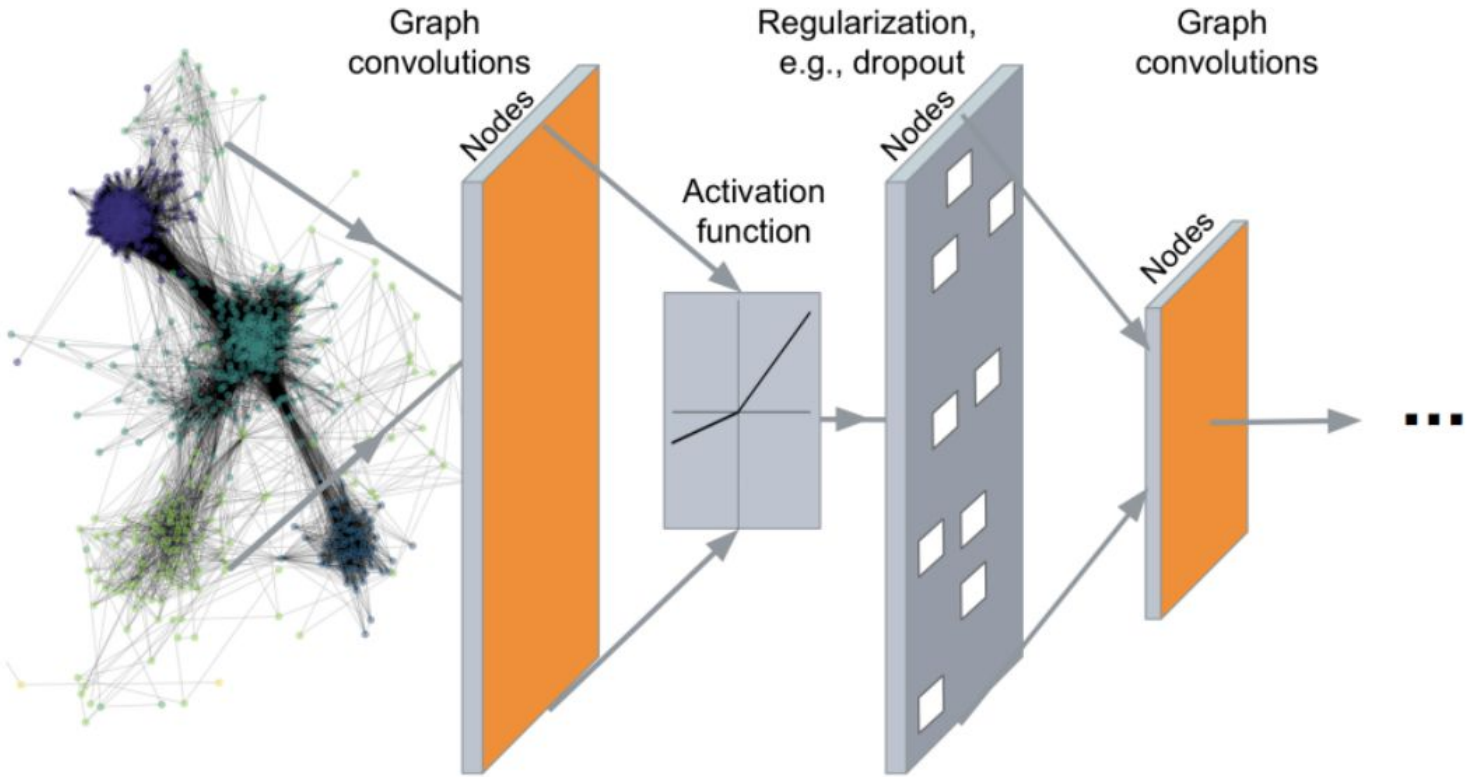

Cho, Hyunghoon, Bonnie Berger, and Jian Peng. "Diffusion component analysis: unraveling functional topology in biological networks." *International Conference on Research in Computational Molecular Biology*. Springer, Cham, 2015.

# Today: Deep learning for graphs

$\mathbf{ENC}(v)$: deep neural networks



# Graph Neural Network (GNN)



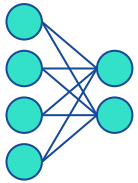

**Output**
Node embeddings
Node-level prediction
Edge-level prediction
Graph-level prediction

# Problem setup

- Given a graph ***G***
  - ***V*** is the **node set**
  - ***A*** is the **adjacency matrix** (assume binary)
  - $\boldsymbol{X} \in \mathbb{R}^{m \times |V|}$ 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]

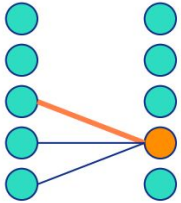
# Inductive bias in neural networks

- Popular neural networks in modern ML toolbox:



**Fully connected networks**

- tabular data
- model all possible interactions



**Convolutional networks**

- data in regular grid
- information flow to local neighbours

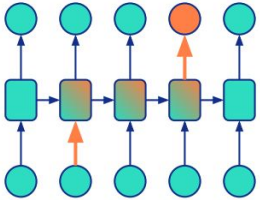
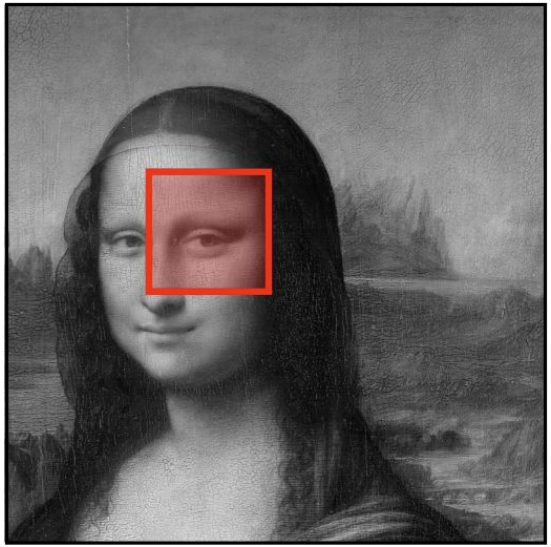


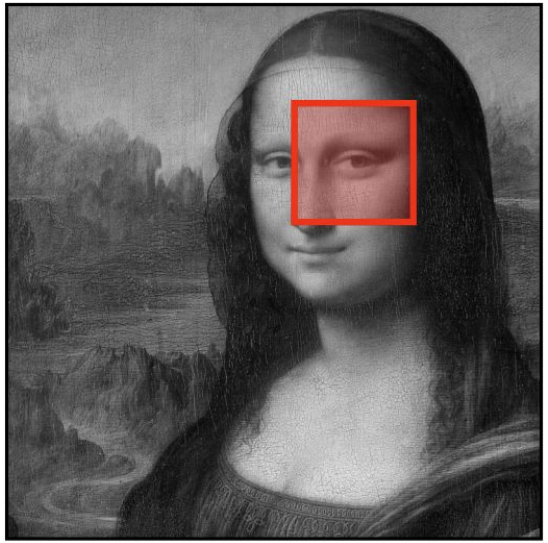
Image source

**Recurrent networks**

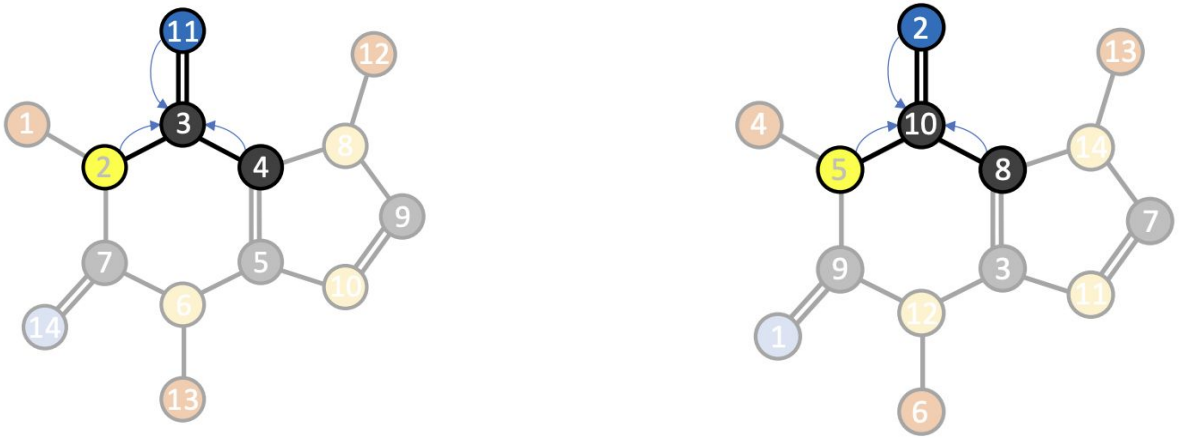
- data in ordered sequence
- information flow sequentially

# Image data: translation equivariant





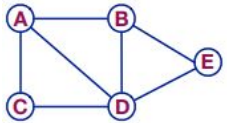
# Graph data: permutation equivariant



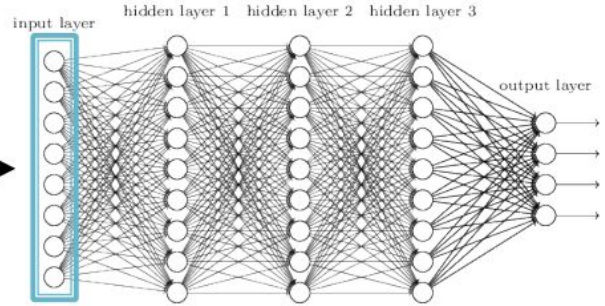

How to incorporate the inductive bias of graph data into neural networks?

# Idea 1: Fully-connected neural network

- Concatenate adjacency matrix and features
- Feed them into a deep neural network




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

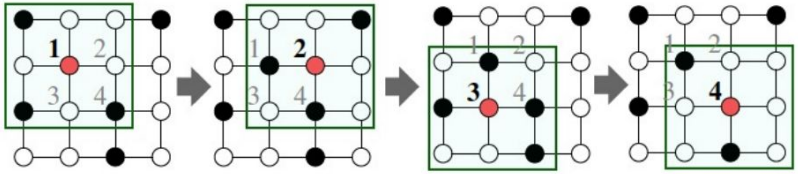


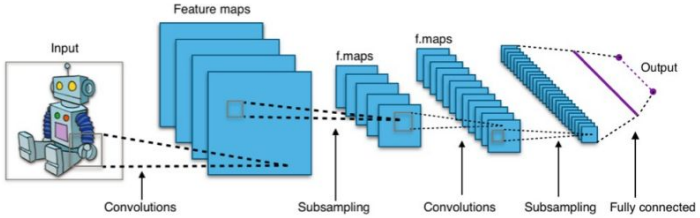

?

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

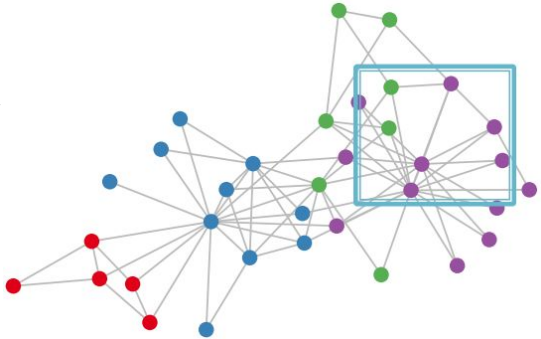
# Idea 2: Convolutional neural network

- CNN for image data




- CNN for Graph data?
  - No fixed notion of locality or sliding window on the graph
  - **Graph is permutation invariant**

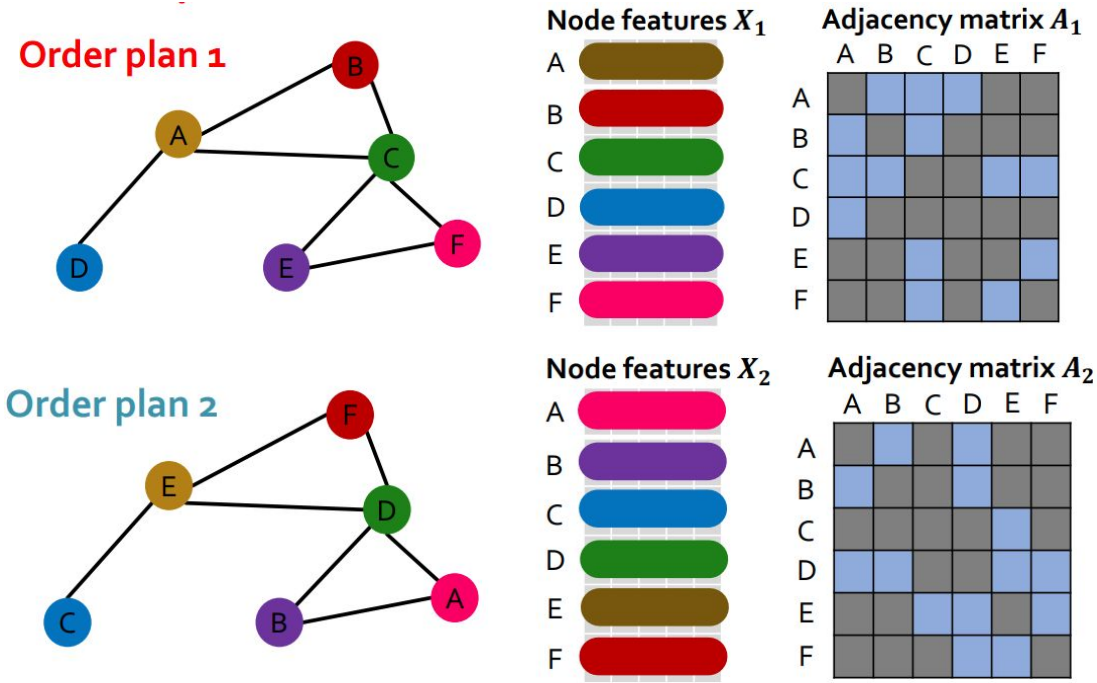


We need a new model with two important properties

- Invariance
- Equivariance

# Permutation invariance

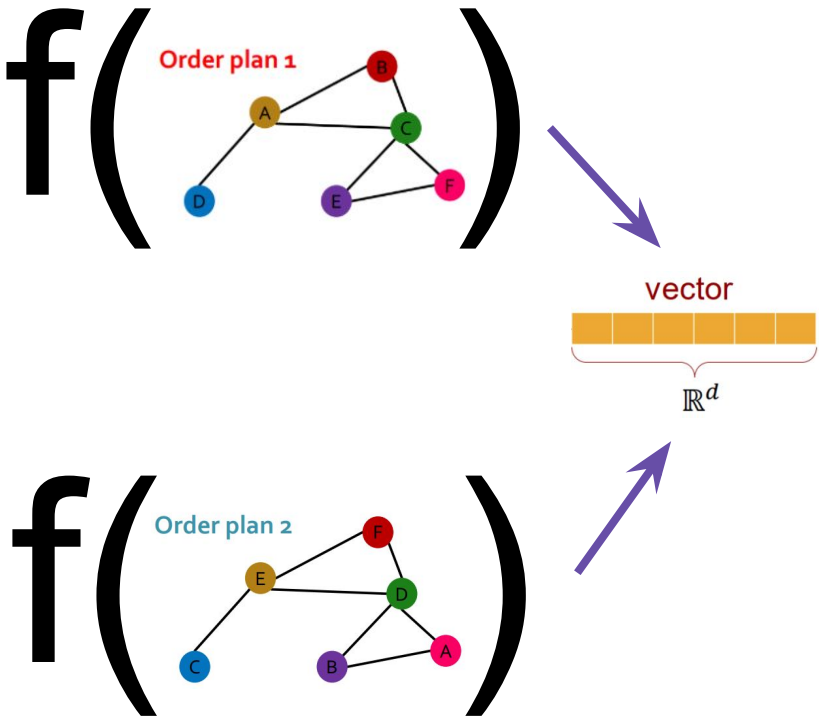
- Graph does not have a canonical order of the nodes




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

# Permutation invariance

- Consider **graph** representation
- **Goal**: learn a function ***f*** that maps a graph $\boldsymbol{G} = (\boldsymbol{A}, \boldsymbol{X})$ to a vector $R^d$
- If $f(\boldsymbol{A}_i, \boldsymbol{X}_i) = f(\boldsymbol{A}_j, \boldsymbol{X}_j)$ for any order plan i and j, we say f is a **permutation invariant function**

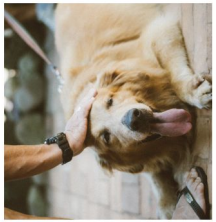


# Permutation invariance

- Want: functions f(**X**) over sets that will not depend on the order
- Equivalently: applying a permutation matrix shouldn't modify result!
- We arrive at a very useful notion of **permutation invariance**.

  f(**X**) is permutation invariant if, for all permutation matrices **P**:

$$f(\mathbf{PX}) = f(\mathbf{X})$$

$f($  $) = f($  $) = \text{“dog”}$

# The Problem with Invariance in Deep Learning

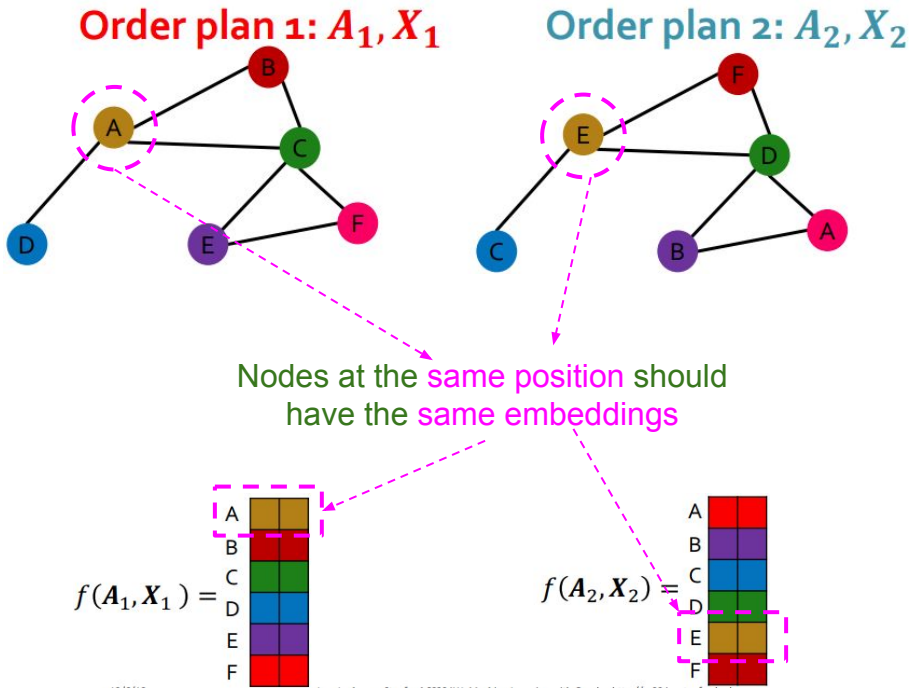
- To recognize whole objects, we need to first recognize parts
  - This is why neural networks should be deep
- If we make the intermediate representations invariant, we lose critical information:


- The relative pose of object parts contains critical information [Hinton]

Hinton et al., Transforming Auto-Encoders, 2011

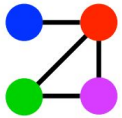
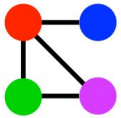
# Permutation equivariance

- Consider **node** representation
- Goal learn a function ***f*** that maps a graph ***G*** = (***A***, ***X***) to a vector $R^{m*d}$
  - m: #nodes, each row is the embedding of a node
- If every pair of nodes at the same position have the same embedding, we say f is a **permutation equivariant function**



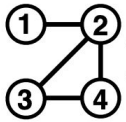
# Invariant and equivariant functions

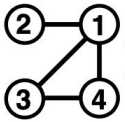
Invariant function

$f($  $) = a,$ $\qquad$ $f($  $) = a$

permutation of indices/nodes does not change the output

Equivariant function

$g($ 

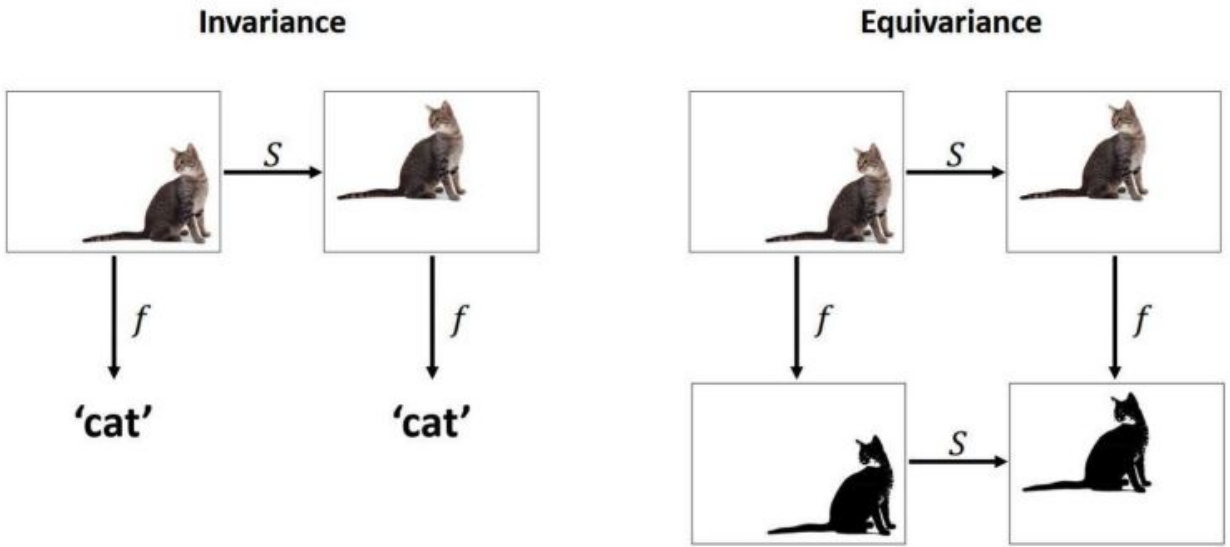
$) = \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix},$ $\qquad$ $g($ 

$) = \begin{bmatrix} b \\ a \\ c \\ d \end{bmatrix}$

permutation of indices/nodes changes the order of the output accordingly

# Invariance & Equivariance

- The analogy in image domain
  - Classification: invariant label
  - Segmentation: equivariant pixel coordinates




https://www.doc.ic.ac.uk/~bkainz/teaching/DL/notes/equivariance.pdf

# Review: Invariant and equivariant functions

A function $f : \mathbb{R}^n \to \mathbb{R}$ is permutation invariant if

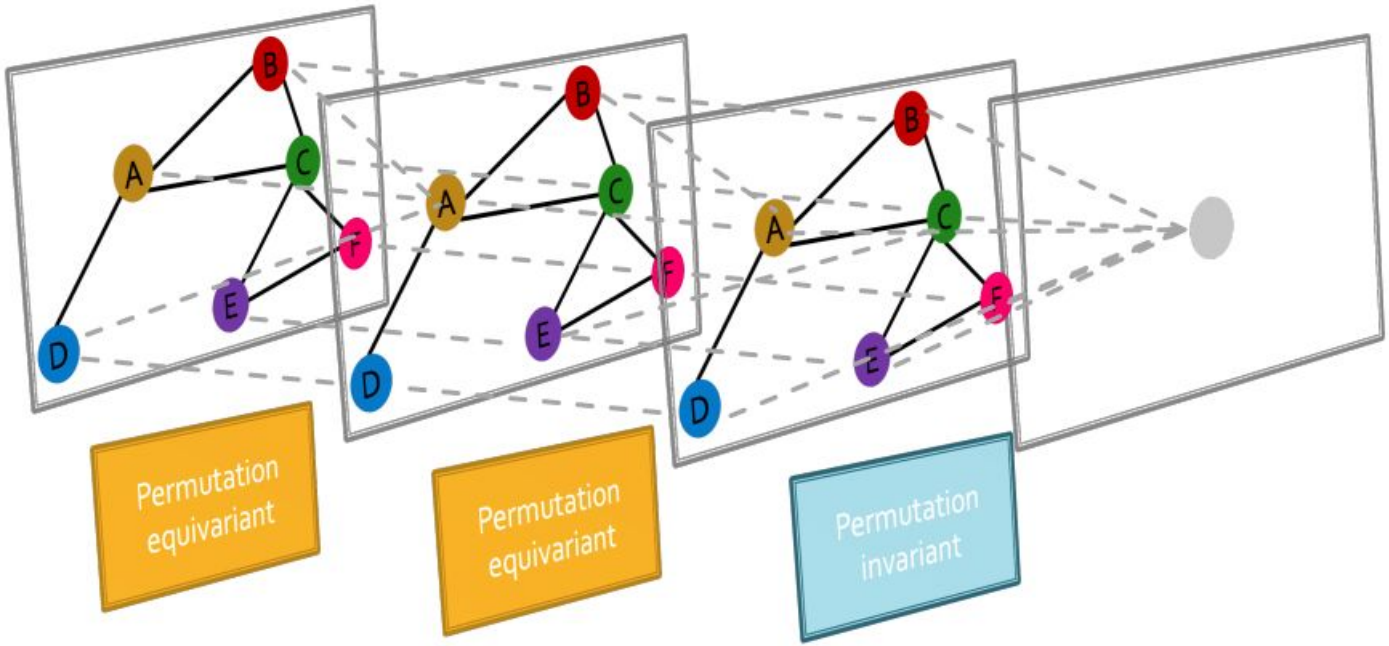
$$f(Px) = f(x) \text{ for all permutation matrices } P \in \mathbb{R}^{n \times n} \text{ and for all } x \in \mathbb{R}^n$$

A function $g : \mathbb{R}^n \to \mathbb{R}^n$ is permutation equivariant if

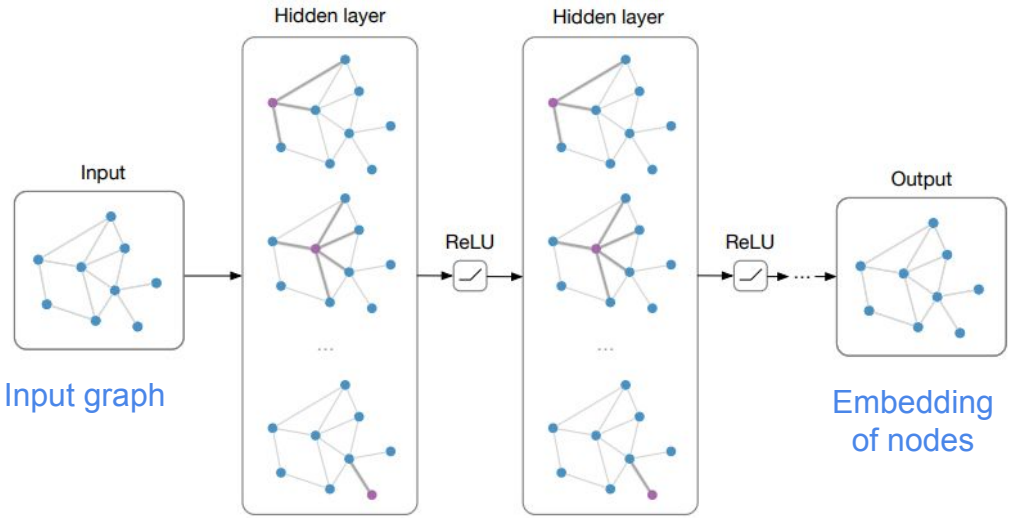
$$g(Px) = Pg(x) \text{ for all permutation matrices } P \in \mathbb{R}^{n \times n} \text{ and for all } x \in \mathbb{R}^n$$

# Graph neural networks overview

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



# Graph neural network

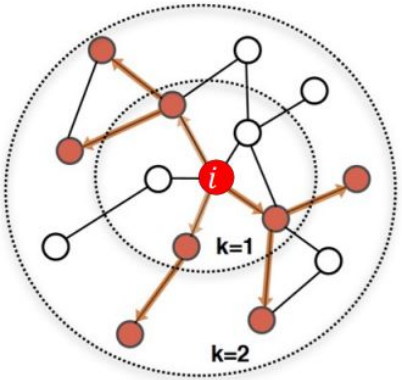



- **Main Idea**: Pass messages between pairs of nodes and then aggregate
- **Alternative Interpretation**: Pass messages between nodes to refine node (and possibly edge) representations

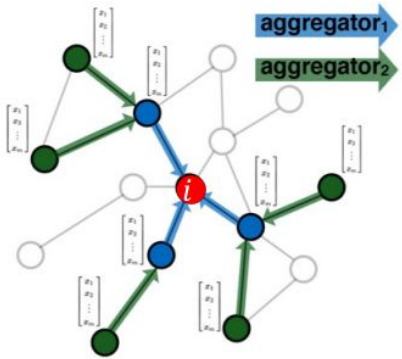
Image credit: Thomas Kipf

# Graph neural network

**Idea**: Node's neighborhood defines a **computation graph**




Determine node computation graph

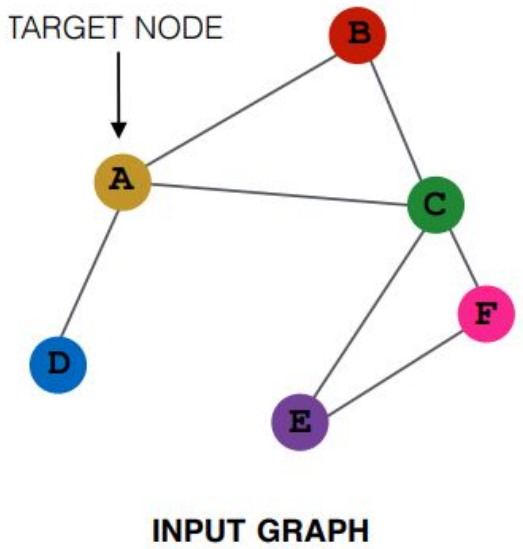


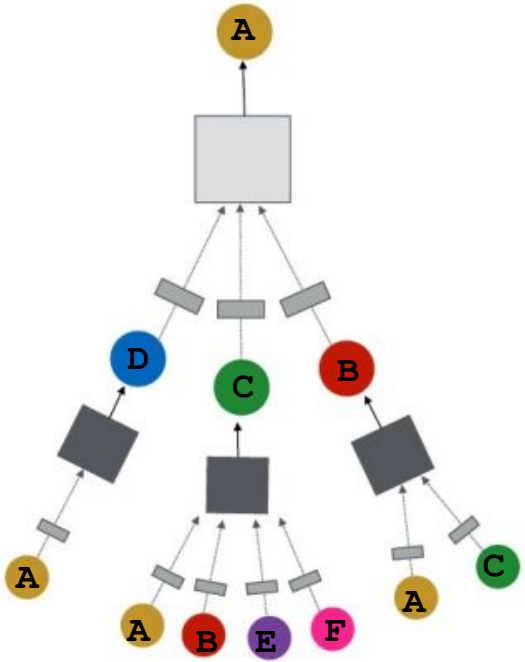

Propagate and transform information

**Goal**: Learn how to propagate information across the graph to compute node features

# Idea: aggregate information from neighbors

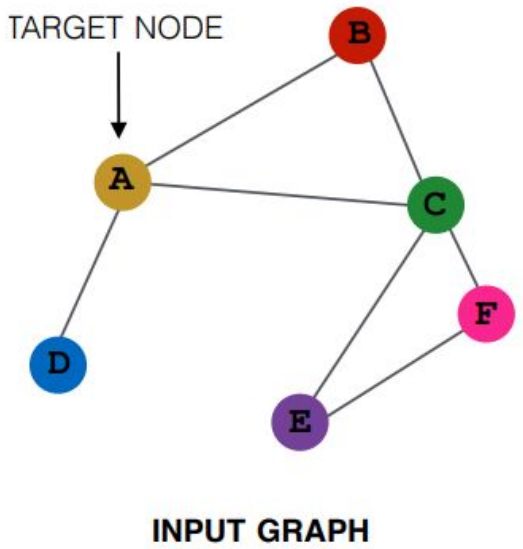
- **Key idea**: generate node embeddings based on **local network neighborhoods**

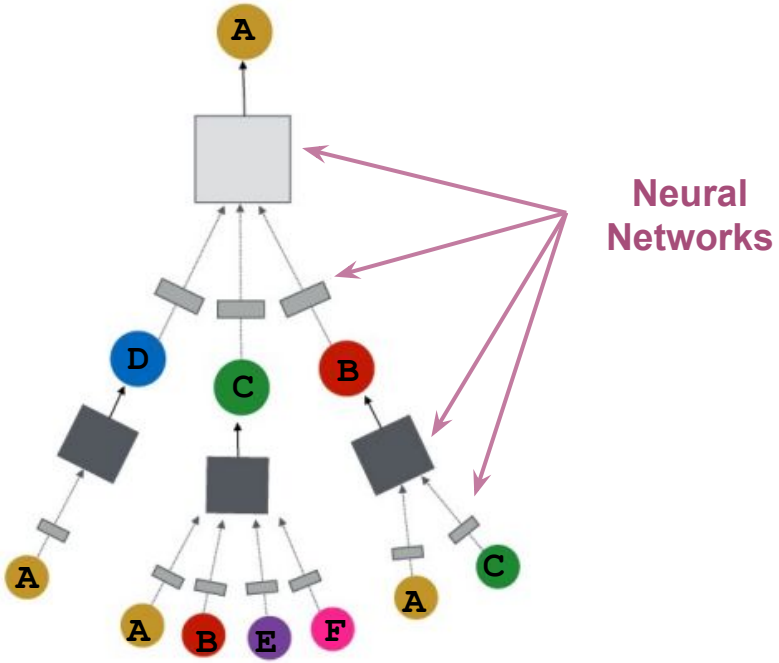


# Idea: aggregate information from neighbors

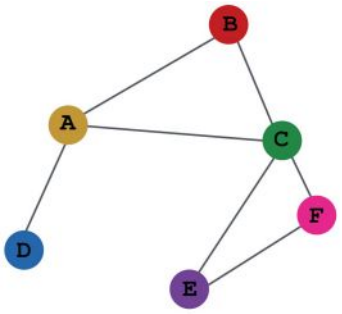
- **Intuition**: Nodes aggregate information from their neighbors using neural networks



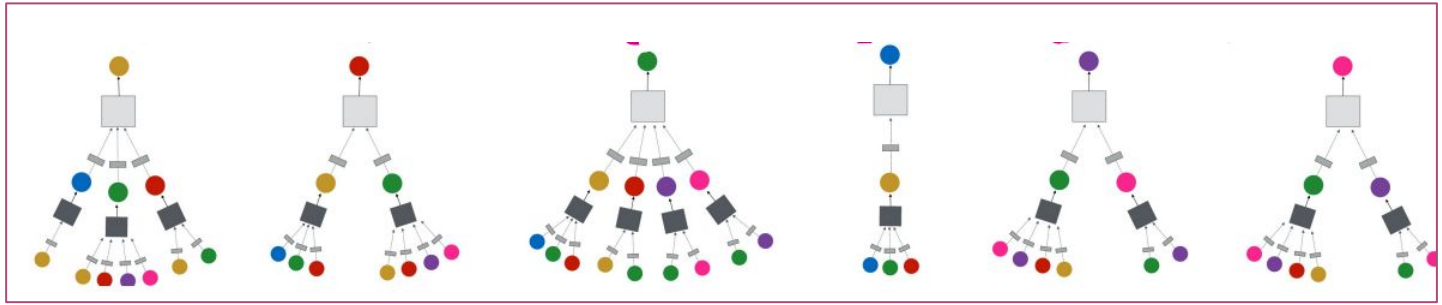
# Idea: aggregate information from neighbors

**Intuition**: Node's neighborhood defines a **computation graph**




INPUT GRAPH

Every node defines a computation graph based on its neighborhood!



# General GNN framework

**1. Message passing**: each node **transforms** its feature vector and **sends** it like a “message” to its neighbors

↓

**2. Aggregation**: each node **receives** feature vectors from its neighbors and **aggregates** them into a single vector

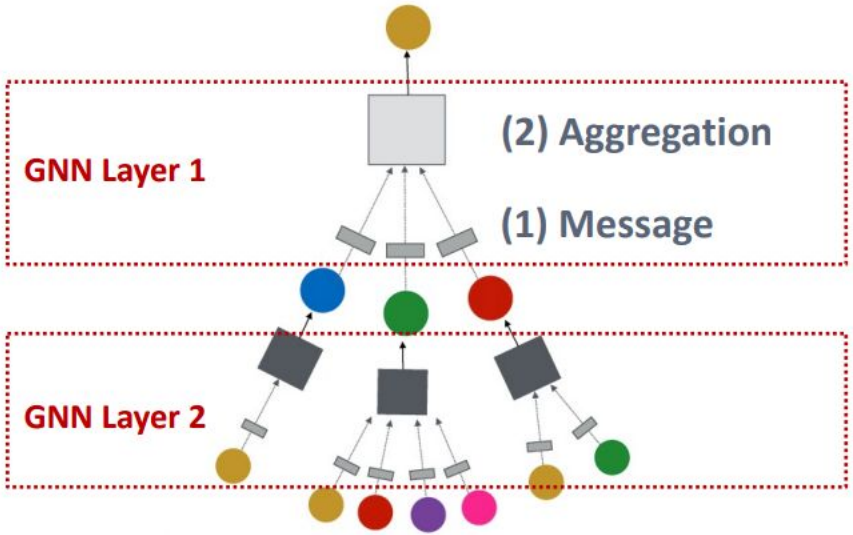
↓

**3. Readout**: **Summarize** node features into the final representation of nodes/edges/graph

Can repeat *K* times

# GNN can have many layers

- GNN can be of arbitrary depth
- Nodes have embeddings at each layer
- Layer-0 embedding of node v is its input feature, $\boldsymbol{x}_v$
- Layer-*k* embedding gets information from nodes that are *k* hops away



# General GNN framework

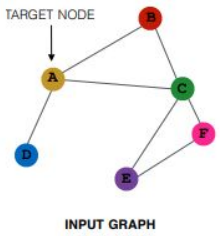
**1. Message passing**: each node **transforms** its feature vector and **sends** it like a “message” to its neighbors

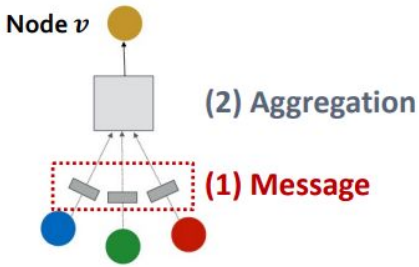
**2. Aggregation**: each node **receives** feature vectors from its neighbors and **aggregates** them into a single vector

**3. Readout**: **Summarize** node features into the final representation of nodes/edges/graph

Can repeat *K* times

# Message passing




The $l$-th layer of GNN

- **Message function:** $\mathbf{m}_u^{(l)} = \mathrm{MSG}^{(l)}\left(\mathbf{h}_u^{(l-1)}\right)$
  - **Intuition:** Each node will create a message, which will be sent to other nodes later
  - **Example:** A Linear layer $\mathbf{m}_u^{(l)} = \mathbf{W}^{(l)}\mathbf{h}_u^{(l-1)}$
    - Multiply node features with weight matrix $\mathbf{W}^{(l)}$

❖ At the 0-th step, $\boldsymbol{h}_u^{0}$ is simply the node feature $\boldsymbol{x}_u$

# General GNN framework

**1. Message passing**: each node **transforms** its feature vector and **sends** it like a “message” to its neighbors

↓

**2. Aggregation**: each node **receives** feature vectors from its neighbors and **aggregates** them into a single vector

↓

**3. Readout**: **Summarize** node features into the final representation of nodes/edges/graph

Can repeat *K* times

# Aggregation

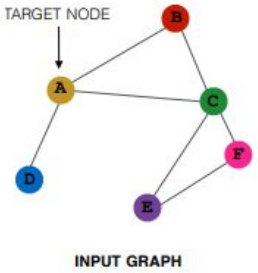
- Aggregation: each node receives feature vectors from its neighbors and aggregates them into a single vector

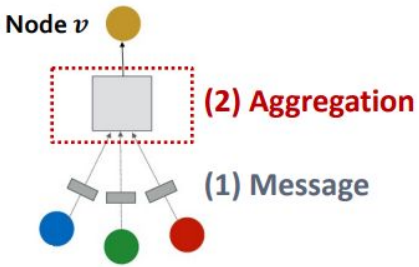
- **Intuition:** Each node will aggregate the messages from node $v$'s neighbors

$$\mathbf{h}_v^{(l)} = \mathrm{AGG}^{(l)}\left(\left\{\mathbf{m}_u^{(l)}, u \in N(v)\right\}\right)$$

aggregation function

  - **Example:** $\mathrm{Sum}(\cdot)$, $\mathrm{Mean}(\cdot)$ or $\mathrm{Max}(\cdot)$ aggregator
    - $\mathbf{h}_v^{(l)} = \mathrm{AGG}^{(l)}\left(\left\{\mathbf{m}_u^{(l)}, u \in N(v)\right\}, \mathbf{m}_v^{(l)}\right)$



# A single GNN layer

**Summary:**

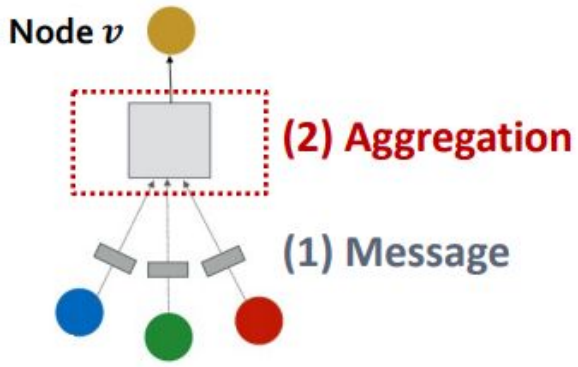
- **(1) Message**: each node computes its own message

$$\mathbf{m}_u^{(l)} = \mathrm{MSG}^{(l)}\left(\mathbf{h}_u^{(l-1)}\right), u \in \{N(v) \cup v\}$$

- **(2) Aggregation**: aggregate messages from neighbors

$$\mathbf{h}_v^{(l)} = \mathrm{AGG}^{(l)}\left(\left\{\mathbf{m}_u^{(l)}, u \in N(v)\right\}, \mathbf{m}_v^{(l)}\right)$$

- **Nonlinearity (activation)**
  - Often written as σ( · ): ReLU( · ), Sigmoid( · ) , …
  - Can be added to message or aggregation




GNN layer
= Message (transformation) + aggregation

# A single GNN layer

**Summary:**

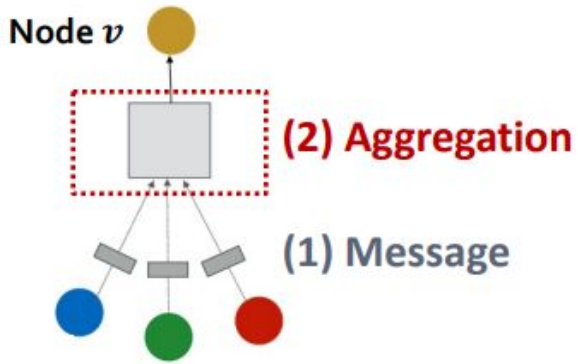
- **(1) Message**: each node computes its own message

$$\mathbf{m}_u^{(l)} = \boxed{\mathrm{MSG}^{(l)}}\left(\mathbf{h}_u^{(l-1)}\right), u \in \{N(v) \cup v\}$$

- **(2) Aggregation**: aggregate messages from neighbors

$$\mathbf{h}_v^{(l)} = \boxed{\mathrm{AGG}^{(l)}}\left(\left\{\mathbf{m}_u^{(l)}, u \in N(v)\right\}, \mathbf{m}_v^{(l)}\right)$$

- **Nonlinearity (activation)**
  - Often written as σ( · ): ReLU( · ), Sigmoid( · ) , …
  - Can be added to message or aggregation




GNN layers have different instantiations
- GCN, GraphSAGE, GAT, …
- Each has its own design of **MSG()** & **ADD()**

# Implemented GNN layers in PyG

PyG (PyTorch Geometric)

**https://www.pyg.org/**

| | |
|---|---|
| `MessagePassing` | Base class for creating message passing layers of the form |
| `GCNConv` | The graph convolutional operator from the "Semi-supervised Classification with Graph Convolutional Networks" paper |
| `ChebConv` | The chebyshev spectral graph convolutional operator from the "Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering" paper |
| `SAGEConv` | The GraphSAGE operator from the "Inductive Representation Learning on Large Graphs" paper |
| `GraphConv` | The graph neural network operator from the "Weisfeiler and Leman Go Neural: Higher-order Graph Neural Networks" paper |
| `GravNetConv` | The GravNet operator from the "Learning Representations of Irregular Particle-detector Geometry with Distance-weighted Graph Networks" paper, where the graph is dynamically constructed using nearest neighbors. |
| `GatedGraphConv` | The gated graph convolution operator from the "Gated Graph Sequence Neural Networks" paper |
| `ResGatedGraphConv` | The residual gated graph convolutional operator from the "Residual Gated Graph ConvNets" paper |
| `GATConv` | The graph attentional operator from the "Graph Attention Networks" paper |
| `GATv2Conv` | The GATv2 operator from the "How Attentive are Graph Attention Networks?" paper, which fixes the static attention problem of the standard `GATConv` layer: since the linear layers in the standard GAT are applied right after each other, the ranking of attended nodes is unconditioned on the query node. |
| `TransformerConv` | The graph transformer operator from the "Masked Label Prediction: Unified Message Passing Model for Semi-Supervised Classification" paper |
| `AGNNConv` | The graph attentional propagation layer from the "Attention-based Graph Neural Network for Semi-Supervised Learning" paper |
| `TAGConv` | The topology adaptive graph convolutional networks operator from the "Topology Adaptive Graph Convolutional Networks" paper |
| `GINConv` | The graph isomorphism operator from the "How Powerful are Graph Neural Networks?" paper |
| `GINEConv` | The modified `GINConv` operator from the "Strategies for Pre-training Graph Neural Networks" paper |

# How to represent a graph in PyG

The graph is represented by a Data object (documentation) in PyG, which we can access as a standard Python namespace.

```
from torch_geometric.data import Data

data = Data(x=x, edge_index=edge_index, ...)
```

- **x** (*torch.Tensor*, *optional*) – Node feature matrix with shape `[num_nodes, num_node_features]`. (default: `None`)
- **edge_index** (*LongTensor*, *optional*) – Graph connectivity in COO format with shape `[2, num_edges]`. (default: `None`)
- **edge_attr** (*torch.Tensor*, *optional*) – Edge feature matrix with shape `[num_edges, num_edge_features]`. (default: `None`)
- **y** (*torch.Tensor*, *optional*) – Graph-level or node-level ground-truth labels with arbitrary shape. (default: `None`)

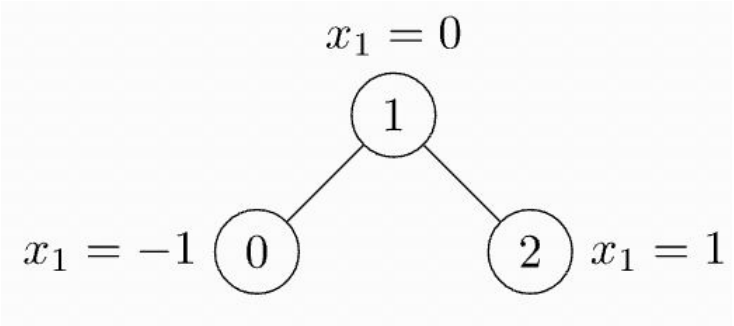
# How to represent a graph in PyG

We show a simple example of an unweighted and undirected graph with three nodes and four edges. Each node contains exactly one feature:

```
import torch
from torch_geometric.data import Data

edge_index = torch.tensor([[0, 1, 1, 2],
                           [1, 0, 2, 1]], dtype=torch.long)
x = torch.tensor([[-1], [0], [1]], dtype=torch.float)

data = Data(x=x, edge_index=edge_index)
>>> Data(edge_index=[2, 4], x=[3, 1])
```



# TODO: Learning PyG basics

https://pytorch-geometric.readthedocs.io/en/latest/get_started/introduction.html

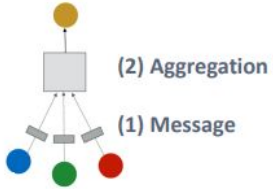
# Classical GNN layers: GCN

## Graph convolutional networks (GCN)

Message

$$\mathbf{h}_v^{(l)} = \sigma\left(\sum_{u \in N(v)} \mathbf{W}^{(l)} \frac{\mathbf{h}_u^{(l-1)}}{|N(v)|}\right)$$

Aggregation




- **Message:**
  - Each Neighbor: $\mathbf{m}_u^{(l)} = \frac{1}{|N(v)|} \mathbf{W}^{(l)} \mathbf{h}_u^{(l-1)}$

    Normalized by node degree
    (In the GCN paper they use a slightly different normalization)
- **Aggregation:**
  - **Sum** over messages from neighbors, then apply activation
  - $\mathbf{h}_v^{(l)} = \sigma\left(\text{Sum}\left(\left\{\mathbf{m}_u^{(l)}, u \in N(v)\right\}\right)\right)$

In GCN graph is assumed to have self-edges that are included in the summation.

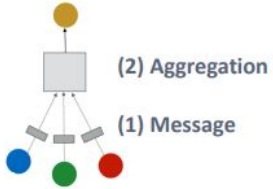
# Classical GNN layers: GCN

## Graph convolutional networks (GCN)

Message

$$\mathbf{h}_v^{(l)} = \sigma\left(\sum_{u \in N(v)} \mathbf{W}^{(l)} \frac{\mathbf{h}_u^{(l-1)}}{|N(v)|}\right)$$

Aggregation




All neighbors have the same weight in the average.
What if not all nodes are equally important?

- **Message:**
  - Each Neighbor: $\mathbf{m}_u^{(l)} = \boxed{\frac{1}{|N(v)|}} \mathbf{W}^{(l)} \mathbf{h}_u^{(l-1)}$

    Normalized by node degree
    (In the GCN paper they use a slightly different normalization)
- **Aggregation:**
  - **Sum** over messages from neighbors, then apply activation
  - $\mathbf{h}_v^{(l)} = \sigma\left(\text{Sum}\left(\left\{\mathbf{m}_u^{(l)}, u \in N(v)\right\}\right)\right)$

In GCN graph is assumed to have self-edges that are included in the summation.

# Classical GNN layers: GAT

## Graph attention networks (GAT)

$$\mathbf{h}_v^{(l)} = \sigma\left(\sum_{u \in N(v)} \boxed{\alpha_{vu}} \mathbf{W}^{(l)} \mathbf{h}_u^{(l-1)}\right)$$

Attention weights

- **Solution:** **Attention mechanism**
  - **Goal**: Learn weights $\boldsymbol{\alpha}_{vu}$ from data, instead of specifying manually (e.g., 1/N(v))
  - Used as a drop-in layer to aggregate embeddings in a neural network (not only in GNN)

# Attention mechanism

**(1)** Apply small neural network (e.g., a single layer) ***a*** to compute the **attention coefficients** $\mathbf{e}_{vu}$ across pairs of nodes u, v based on their messages:

$$e_{vu} = a(\mathbf{W}^{(l)}\mathbf{h}_u^{(l-1)}, \mathbf{W}^{(l)}\boldsymbol{h}_v^{(l-1)})$$

- $\mathbf{e}_{vu}$ indicates the importance of ***u***'s message to node ***v***

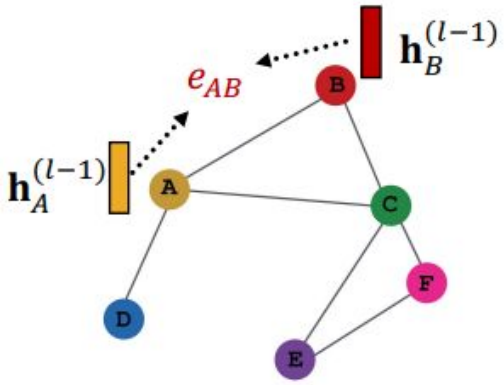
**(2) Normalize** $\mathbf{e}_{vu}$ into the final attention weight $\alpha_{vu}$

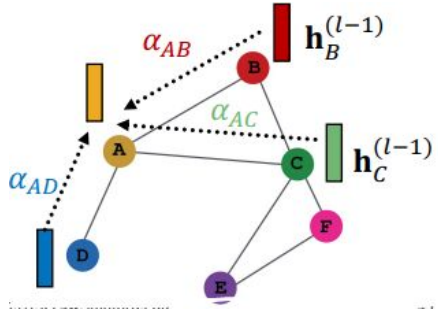
- Use the **softmax** function, so that $\sum_{u \in N(v)} \alpha_{vu} = 1$:

$$\alpha_{vu} = \frac{\exp(e_{vu})}{\sum_{k \in N(v)} \exp(e_{vk})}$$

**(3) Weighted sum** based on the final attention weight $\alpha_{vu}$

$$\mathbf{h}_v^{(l)} = \sigma(\textstyle\sum_{u \in N(v)} \alpha_{vu} \mathbf{W}^{(l)} \mathbf{h}_u^{(l-1)})$$




**Weighted sum using** $\alpha_{AB}$, $\alpha_{AC}$, $\alpha_{AD}$**:**

$$\mathbf{h}_A^{(l)} = \sigma(\alpha_{AB}\mathbf{W}^{(l)}\mathbf{h}_B^{(l-1)} + \alpha_{AC}\mathbf{W}^{(l)}\mathbf{h}_C^{(l-1)} + \alpha_{AD}\mathbf{W}^{(l)}\mathbf{h}_D^{(l-1)})$$

Adapted from CS224W, Jure Leskovec

# General GNN framework

**1. Message passing**: each node **transforms** its feature vector and **sends** it like a “message” to its neighbors

**2. Aggregation**: each node **receives** feature vectors from its neighbors and **aggregates** them into a single vector

**3. Readout**: **Summarize** node features into the final representation of nodes/edges/graph

Can repeat *K* times (e.g., *K*=1,2,3,4)

# General GNN framework

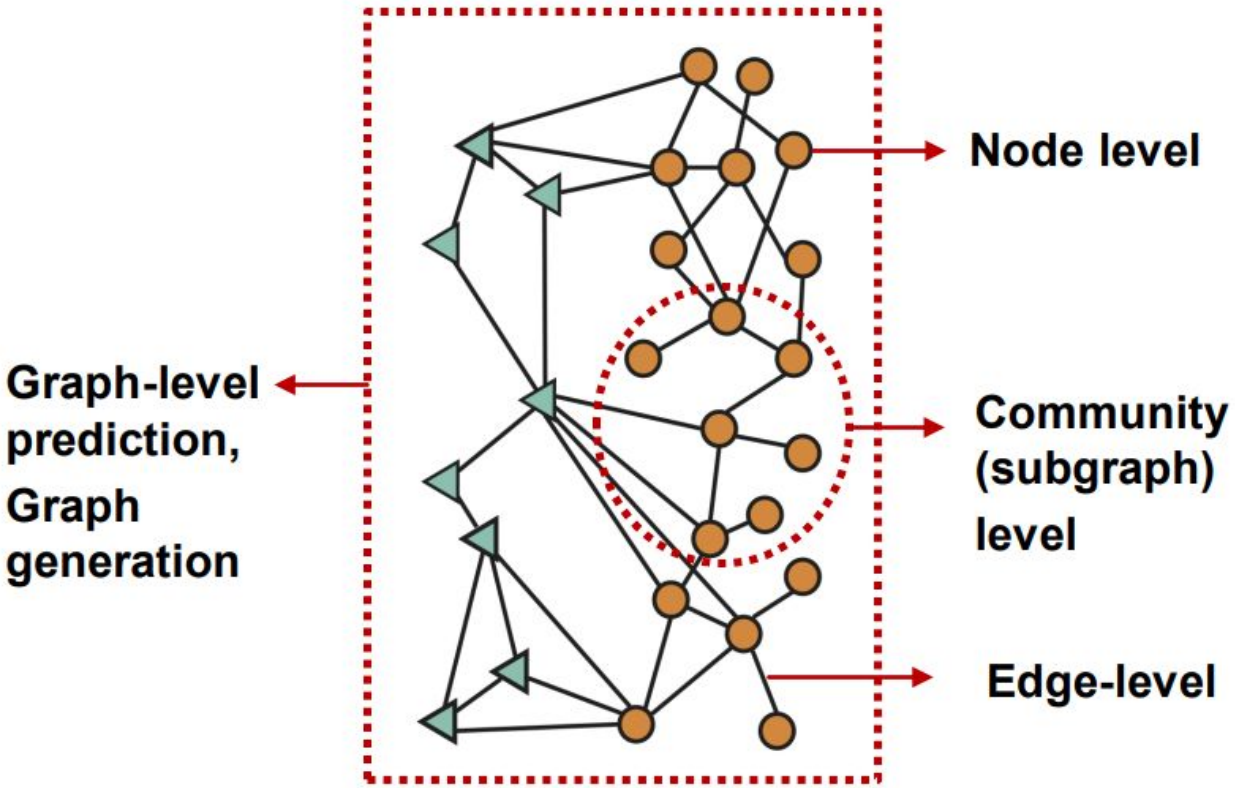
**1. Message passing**: each node **transforms** its feature vector and **sends** it like a “message” to its neighbors

**2. Aggregation**: each node **receives** feature vectors from its neighbors and **aggregates** them into a single vector

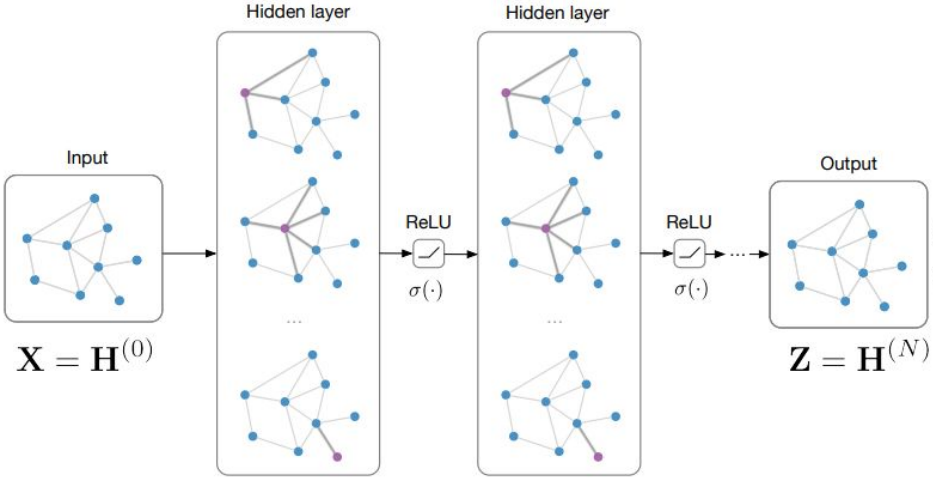
Can repeat *K* times

**3. Readout**: **Summarize** node features into the final representation of nodes/edges/graph

# Recap: Different types of ML tasks on graphs



# Readout operation depends on the task




$$\mathbf{H}^{(l+1)} = \sigma\left(\hat{\mathbf{A}}\mathbf{H}^{(l)}\mathbf{W}^{(l)}\right)$$

**Node classification:**

$$\mathrm{softmax}(\mathbf{z_n})$$

e.g. Kipf & Welling (ICLR 2017)

**Graph classification:**

$$\mathrm{softmax}\left(\sum_n \mathbf{z_n}\right)$$

e.g. Duvenaud et al. (NIPS 2015)

Similar to AGG(): Mean(), Add(), Sum()

**Link prediction:**

$$p(A_{ij}) = \sigma(\mathbf{z_i^T z_j})$$

Kipf & Welling (NIPS BDL 2016)
**"Graph Auto-Encoders"**

Dot product or concatenation

Image credit: Thomas Kipf

# Demo: GNN

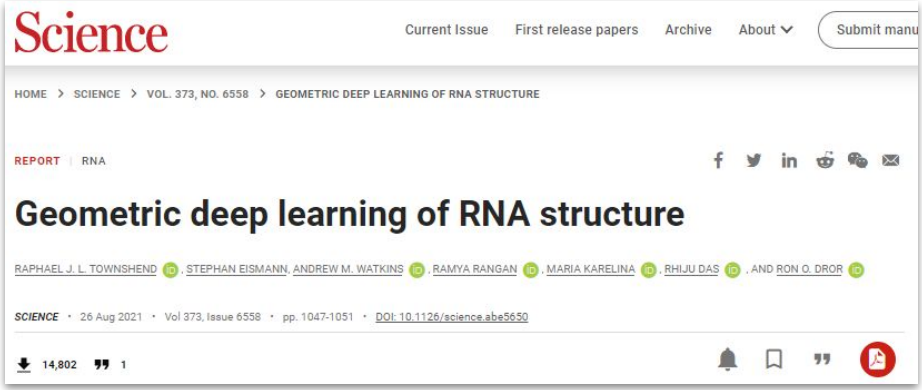
Demo 1: GNN basics
Demo 2: Node-level prediction
Demo 3: Graph-level prediction

Code demos from PyG tutorials

# Bonus slides: GNN for 3D structure

# Learning from structure

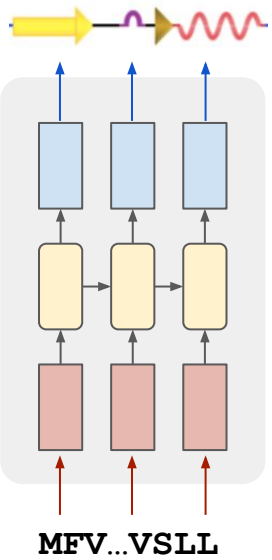
- RNA
  - Secondary structure
    - dynamic programming
  - Tertiary structure
    - Still very challenging
    - Deep learning for 3D RNA structure assessment




Townshend et al., *Science*, 2021

# Learning from structure

- RNA
- Protein
  - Secondary structure
    - “Many-to-many” sequence learning
  - Tertiary structure
    - AlphaFold



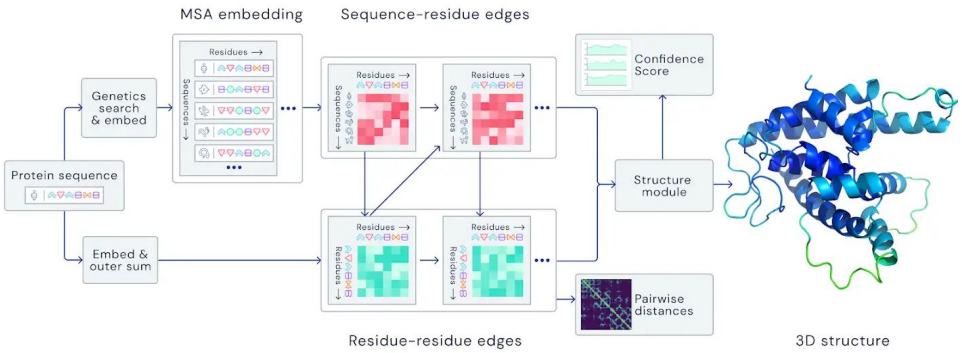
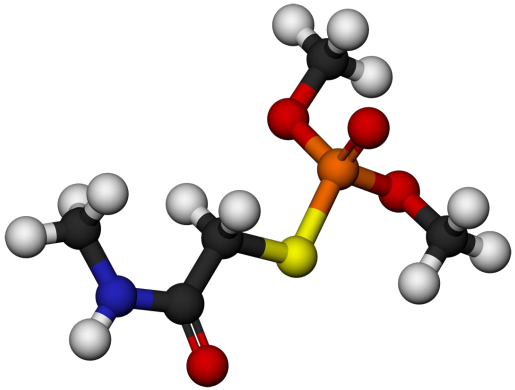



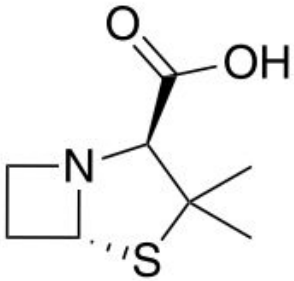

Image credit: DeepMind

# Learning from structure

- RNA
- Protein
- Chemical molecule structure (e.g., small-molecule drugs)



# Molecule structure

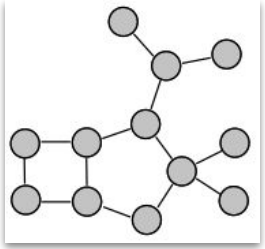



Molecule

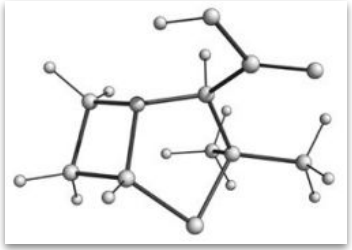
- 1D representation (SMILES string)

CC1(C)[C@H](C(O)=O)N2[C@@H](CC2)S1

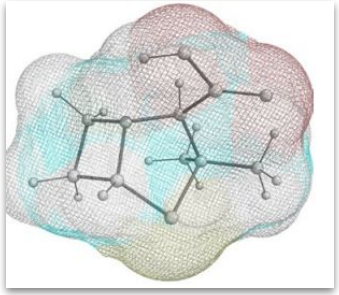
- 2D representation (graph)



- 3D representation (coordinates)



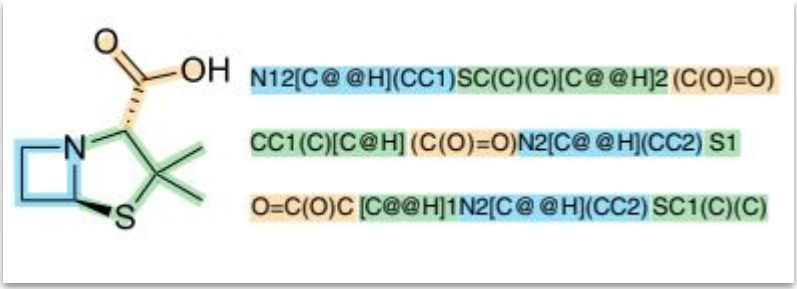
- Surface representation (mesh)



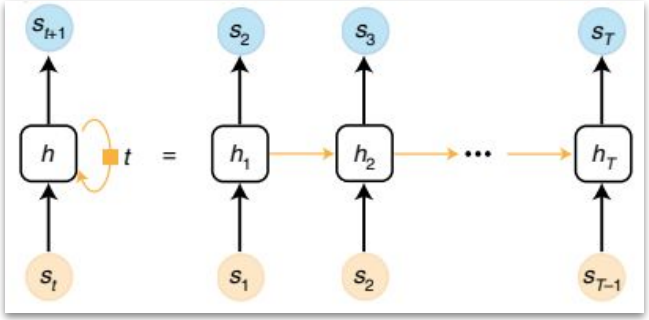
Atz et al., *Nature Machine Intelligence*, 2021

# Deep learning for molecule structure (1D)

**SMILES** string: linear representation of a molecule



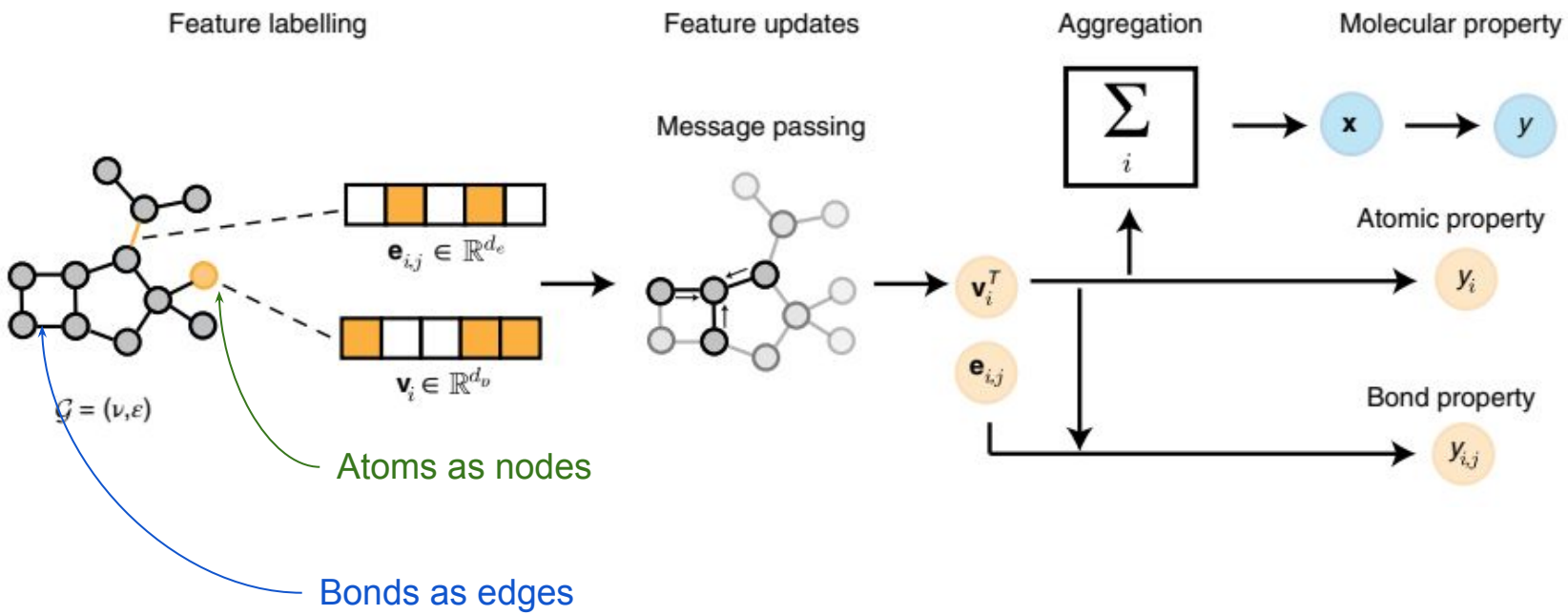

**Example:** RNN-based models




Atz et al., *Nature Machine Intelligence*, 2021

# Deep learning for molecule structure (2D)

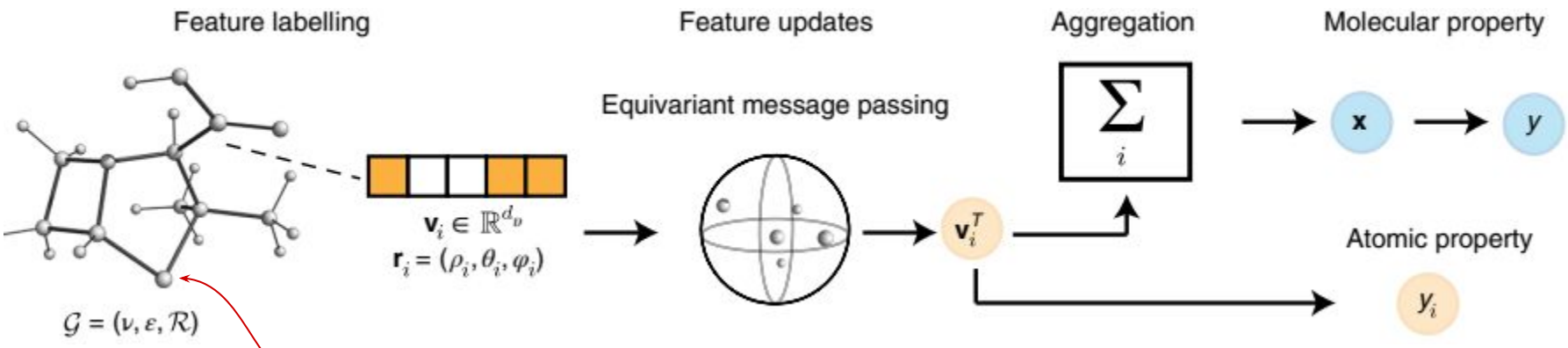
**Example:** Graph neural network (GNN)




Atz et al., *Nature Machine Intelligence*, 2021

# Deep learning for molecule structure (3D)

**Example:** *Equivariant* Graph neural network




Atoms represented by 3D coordinate (x, y, z) and potentially other features

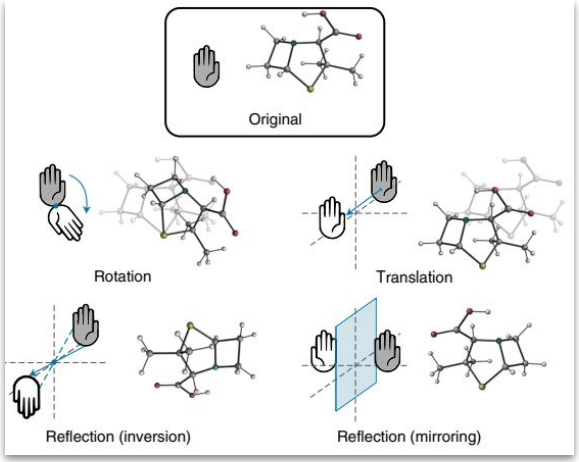
Atz et al., *Nature Machine Intelligence*, 2021

# Why invariance and equivariance?

- ***X***: input molecule
- ***F***: neural network
- ***T***: transformation (e.g., rotation, translation, reflection)

- Invariance
  - $F(T(X)) = F(X)$
    - Output remains the same no matter how the input is rotated, shifted, etc
  - Motivation: many molecular descriptors are invariant to the rotation and translation of the molecular representation

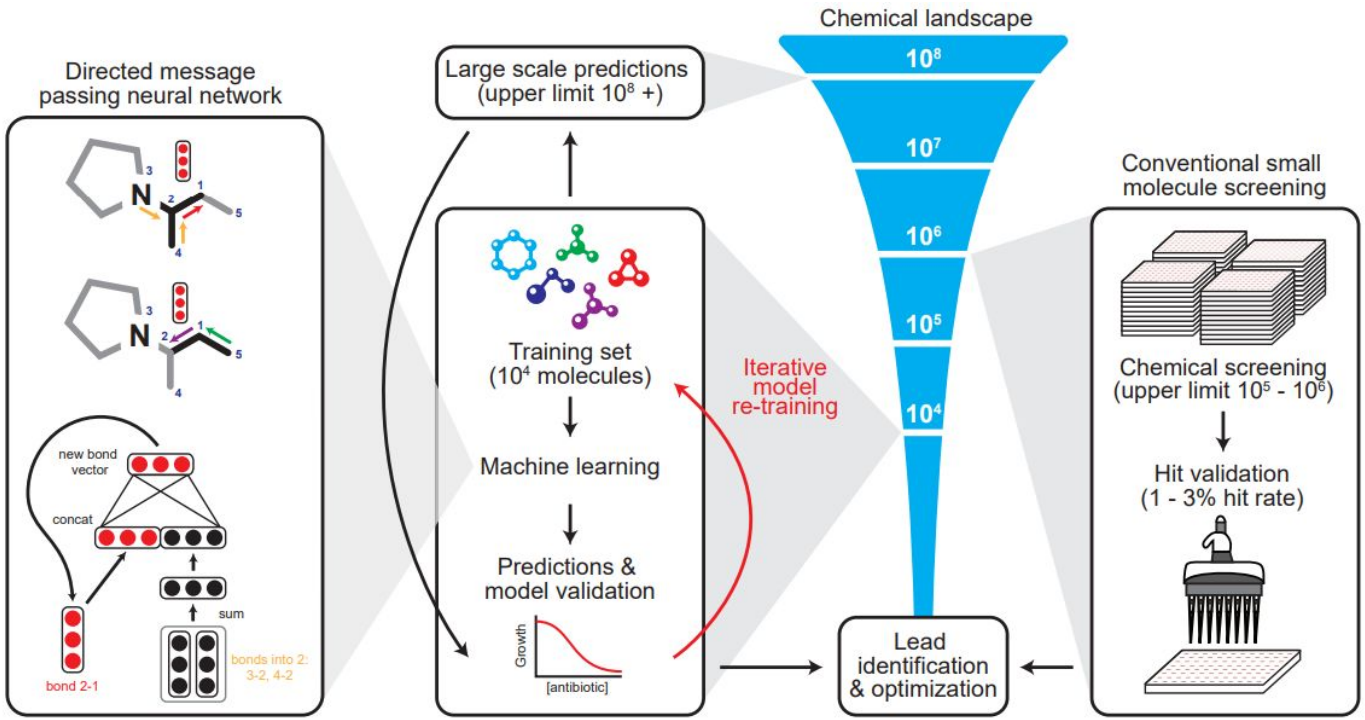
- Equivariance
  - $F(T(X)) = TF(X)$
    - Output changes in the same way as the input
  - Motivation: some property changes following a symmetry transformation (e.g., chiral properties that change under reflection of the molecule)




Atz et al., *Nature Machine Intelligence*, 2021

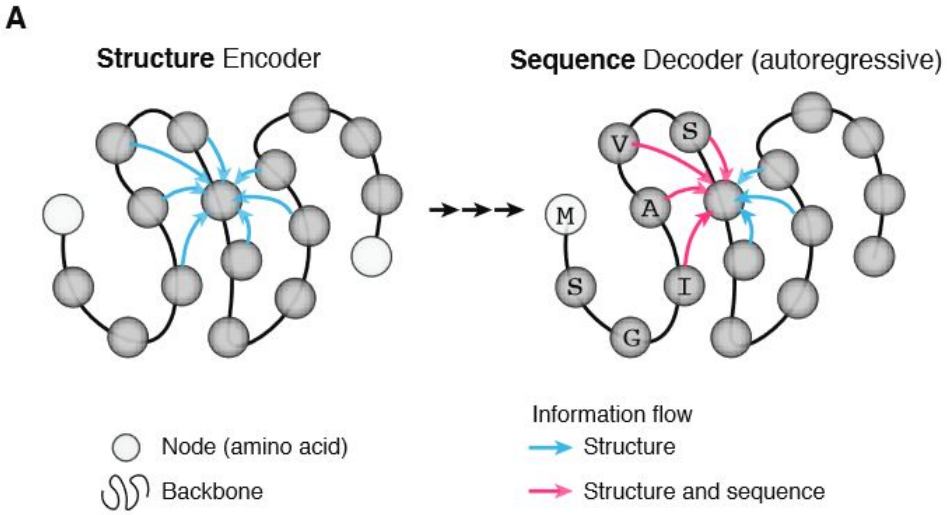
# GNN for Computational Biology

# Applications: antibiotic discovery




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

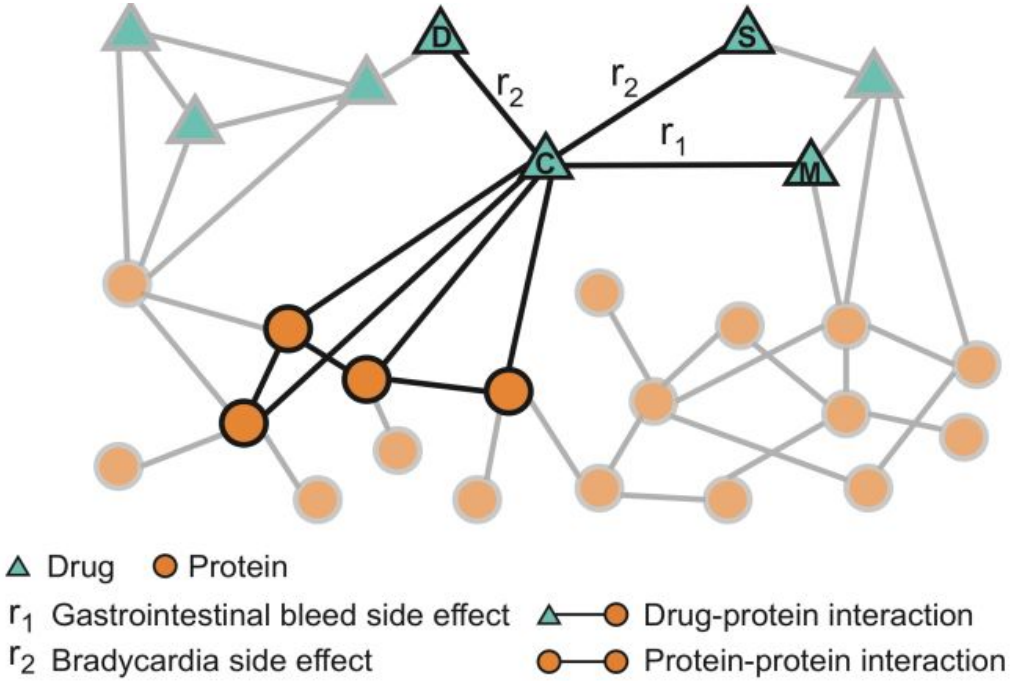
# Applications: protein sequence design




Ingraham et al. “Generative models for graph-based protein design”, NeurIPS, 2019

Jin et al. “Iterative Refinement Graph Neural Network for Antibody Sequence-Structure Co-Design”, ICLR 2022

# Applications: Polypharmacy effect prediction




Zitnik et al., “Modeling Polypharmacy Side Effects with Graph Convolutional Networks”, *Bioinformatics*, 2018

# Project ideas

GNNs for computational biology problems

- Drug-target interaction prediction
- Protein-protein interaction prediction
- Molecular property prediction
- …

Datasets & Problems resources

- Therapeutics Data Commons
- Open Graph Benchmark

# Summary of today

- Graph neural network (GNN)
  - Generalize convolution to graphs
  - Invariance and equivariance
- GNN framework
  - Message
  - Aggregation
  - Readout

# A CSE Faculty Candidate Seminar Tomorrow (02/29)

Join this seminar if you're interested in genomics and deep learning!

**Title**: Dissecting the Cell Type-Specific Regulatory Role of Each Nucleotide in the Human Genome

**Speaker**: Jacob Schreiber (postdoctoral scholar at Stanford University)

**Location**: **CODA** Building, Second Floor, Room 230

**Date**: Thursday, **February 29, 2024 at 11:00 am**

**Abstract**:
https://www.cse.gatech.edu/events/2024/02/29/cse-faculty-candidate-seminar-jacob-schreiber