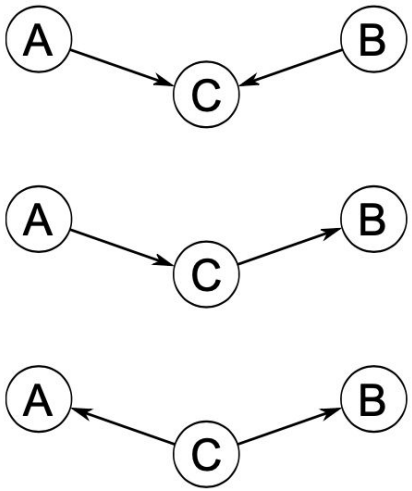
CSE8803/CX4803

Machine Learning in Computational Biology

Lecture 17:
Deep Learning for Networks
(Graph Neural Networks)

Yunan Luo

Conditional independence in BN



Case 1:

$$P(A,B,C) = P(C|A,B)P(A,B)$$
$$= P(C|A,B)P(A)P(B)$$

Case 2:

$$P(A,B,C) = P(B,C|A)P(A)$$
$$= P(B|A,C)P(C|A)P(A)$$
$$= P(B|C)P(C|A)P(A)$$

Case 3:

$$P(A,B,C) = P(A,B|C)P(C)$$
$$= P(A|C)P(B|C)P(C)$$

Example: proof of conditional independence of A, B in case 3

$$P(A,B|C) = \frac{P(A,B,C)}{P(C)} = \frac{P(C)P(A|C)P(B|C)}{P(C)} = P(A|C)P(B|C)$$

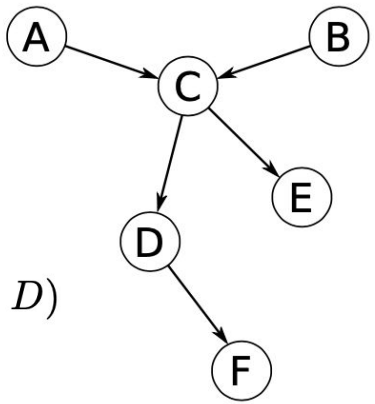
Conditional independence in BN

Each node is dependent only on its parents

Define:

$\mathrm{P}(\mathbf{X})$ - the global probability of all variables

$\mathrm{P}(\mathbf{X}) = \mathrm{P}(\mathrm{A}, \mathrm{B}, \mathrm{C}, \mathrm{D}, \mathrm{E}, \mathrm{F})$



$$\mathrm{P}(\mathbf{X}) = \mathrm{P}(A)\,\mathrm{P}(B)\,\mathrm{P}(C \mid A, B)\,\mathrm{P}(D \mid C)\,\mathrm{P}(E \mid C)\,\mathrm{P}(F \mid D)$$

In general:

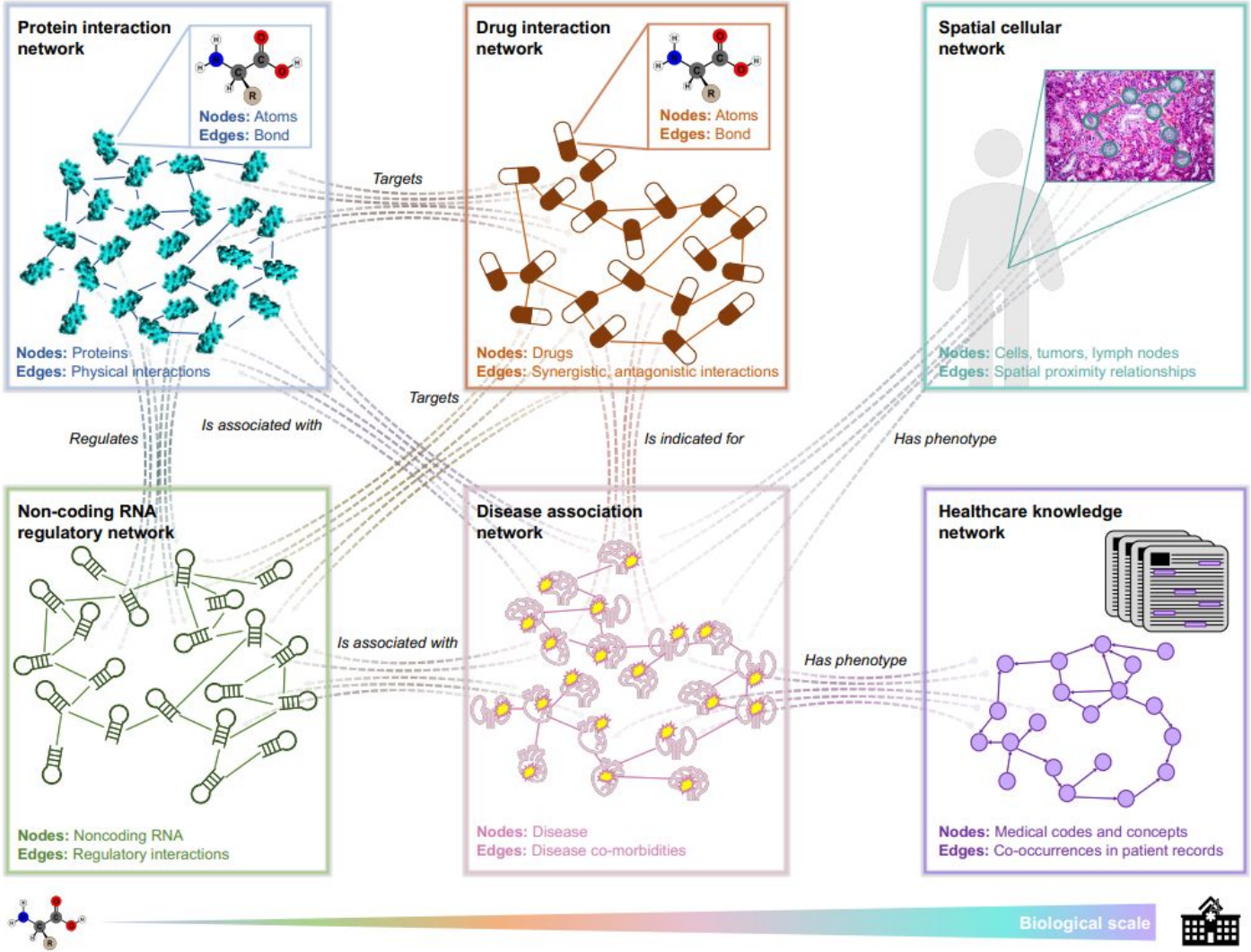
$$p(X_1, \ldots, X_n) = \prod_{i=1}^{n} p(X_i | X_{\mathsf{pa}(i)})$$

where pa(i) is the parents of node i

Each variable is conditionally independent of its non-descendants, given its parents.

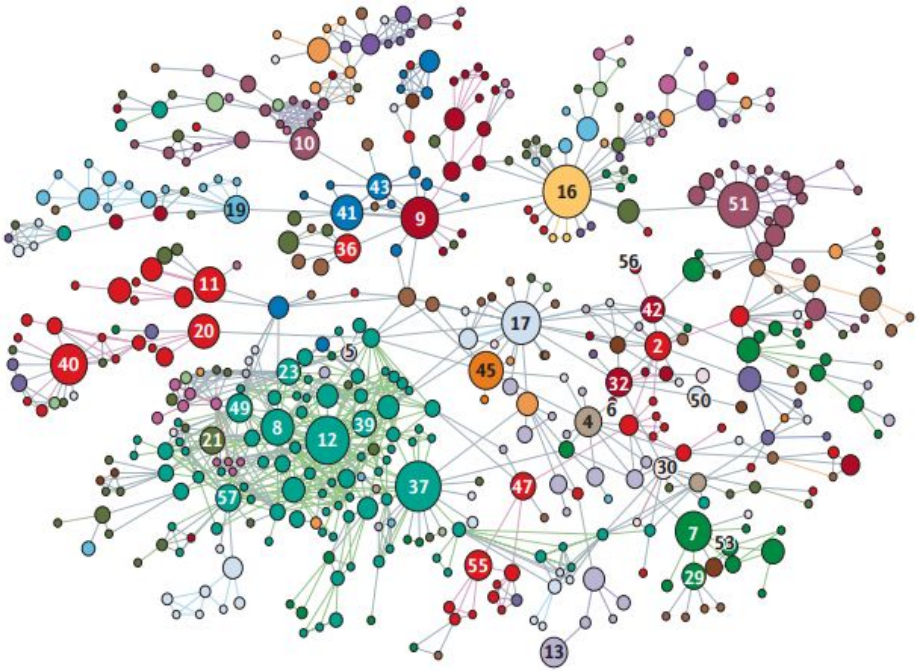
Slide credit: Marco Scutari

Why networks in biology?



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

Human disease network



Node: protein
Edge: protein-protein interaction

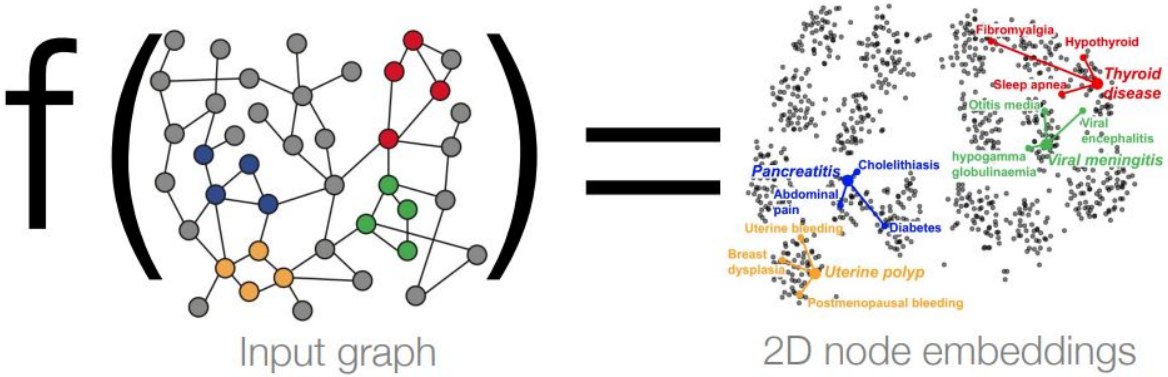
1. Aldosteronism
2. Alzheimer's disease
3. Anaemia, congenital deserythropoietic
4. Asthma
5. Ataxia-telangiectasia
6. Atherosclerosis
7. Blood group
8. Breast cancer
9. Cardiomyopathy
10. Cataract
11. Charcot–Marie–Tooth disease
12. Colon cancer
13. Complement component deficiency
14. Coronary artery 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?

Adapted from CS224W, Jure Leskovec

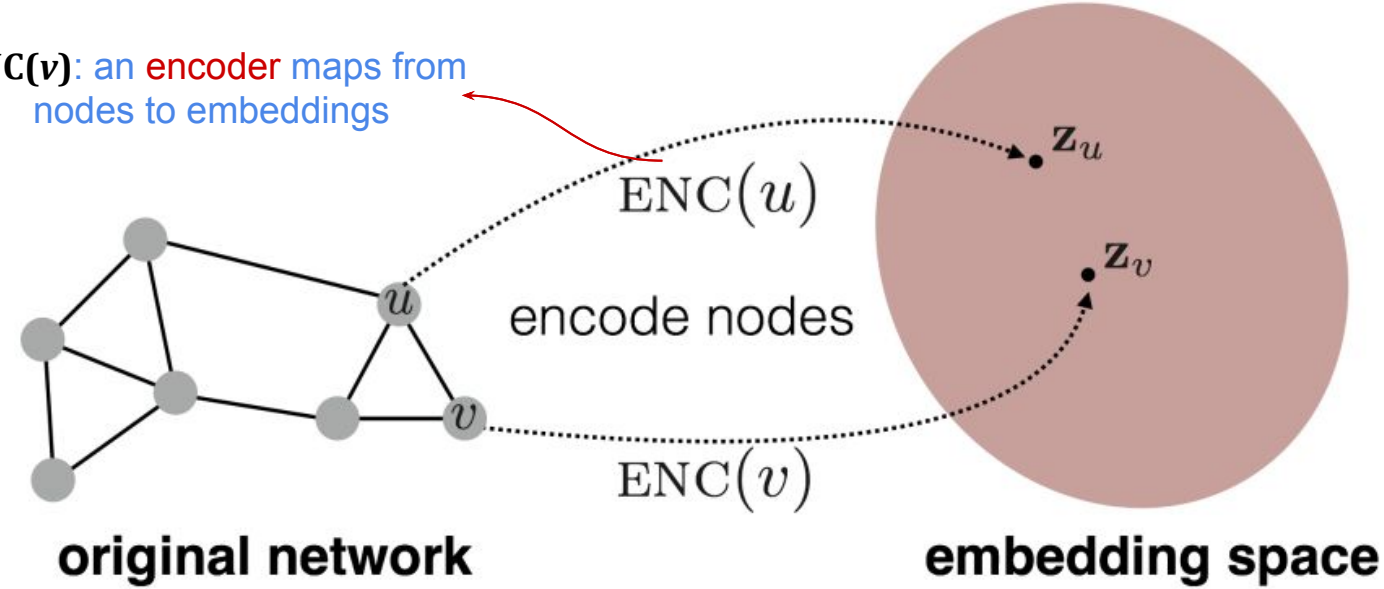
Recap: Embedding nodes

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

In the original network

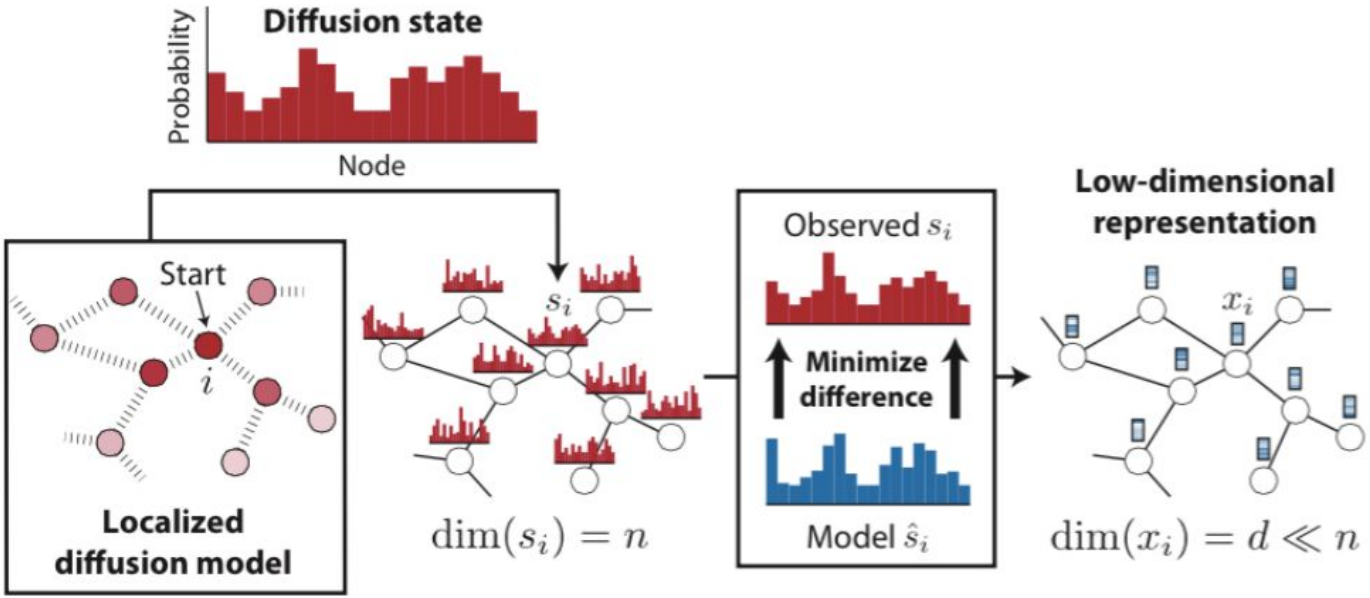
Similarity of the embeddings

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



Adapted from CS224W, Jure Leskovec

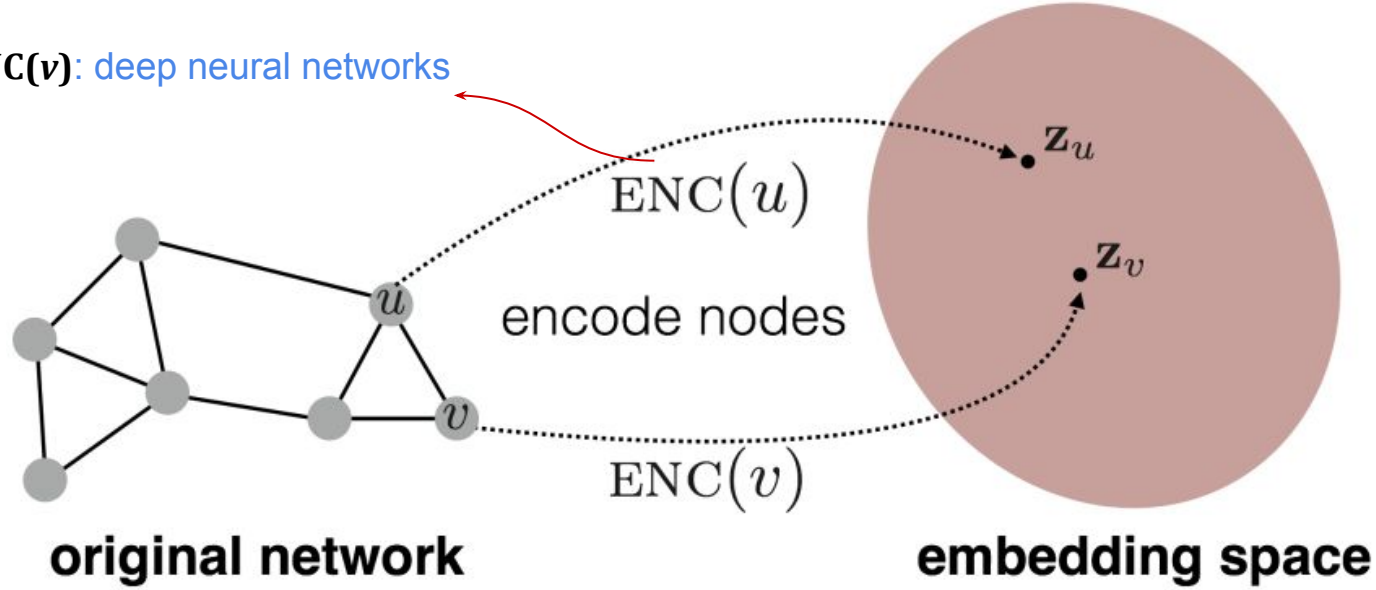
Recap: Diffusion-based approaches



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.

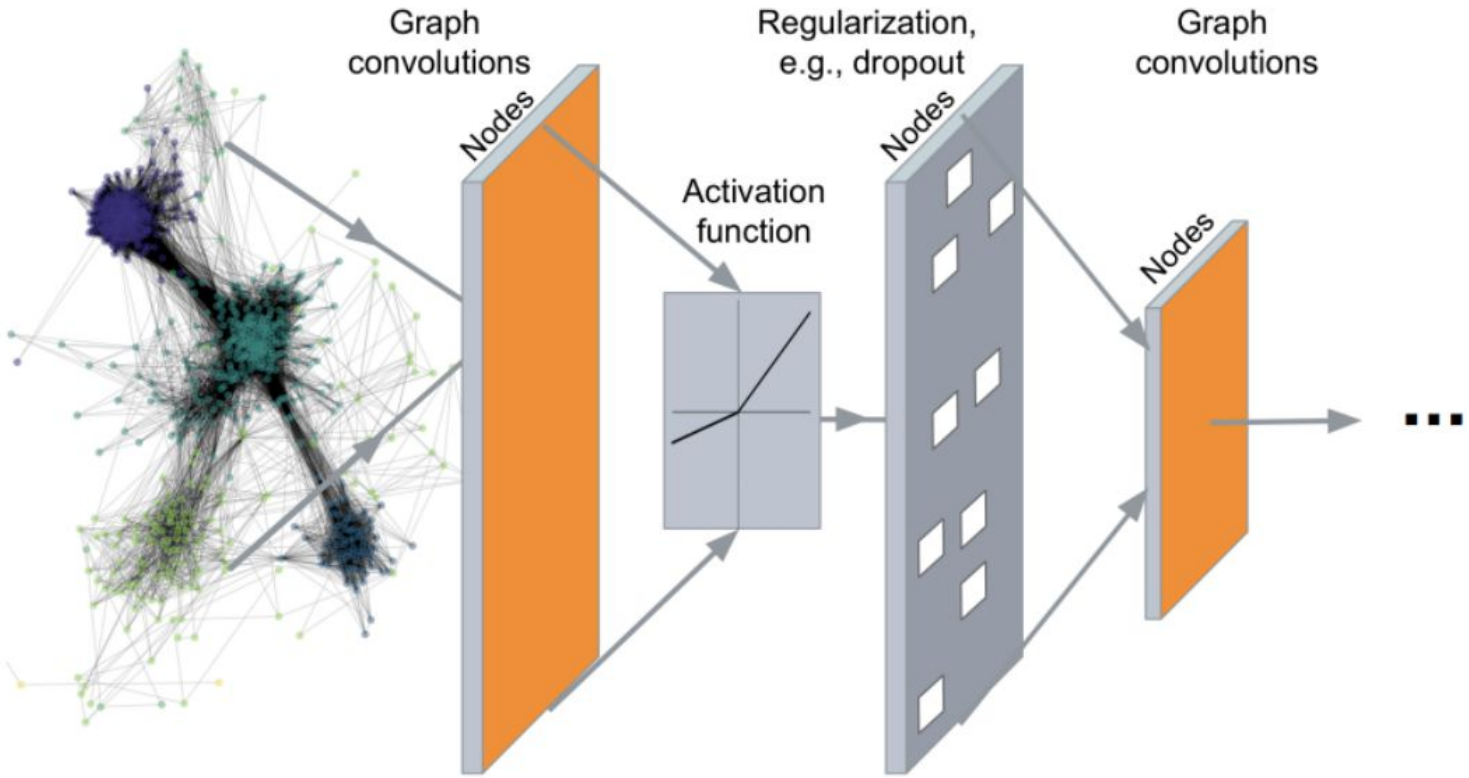
Today: Deep learning for graphs

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



Adapted from CS224W, Jure Leskovec

Graph Neural Network (GNN)



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

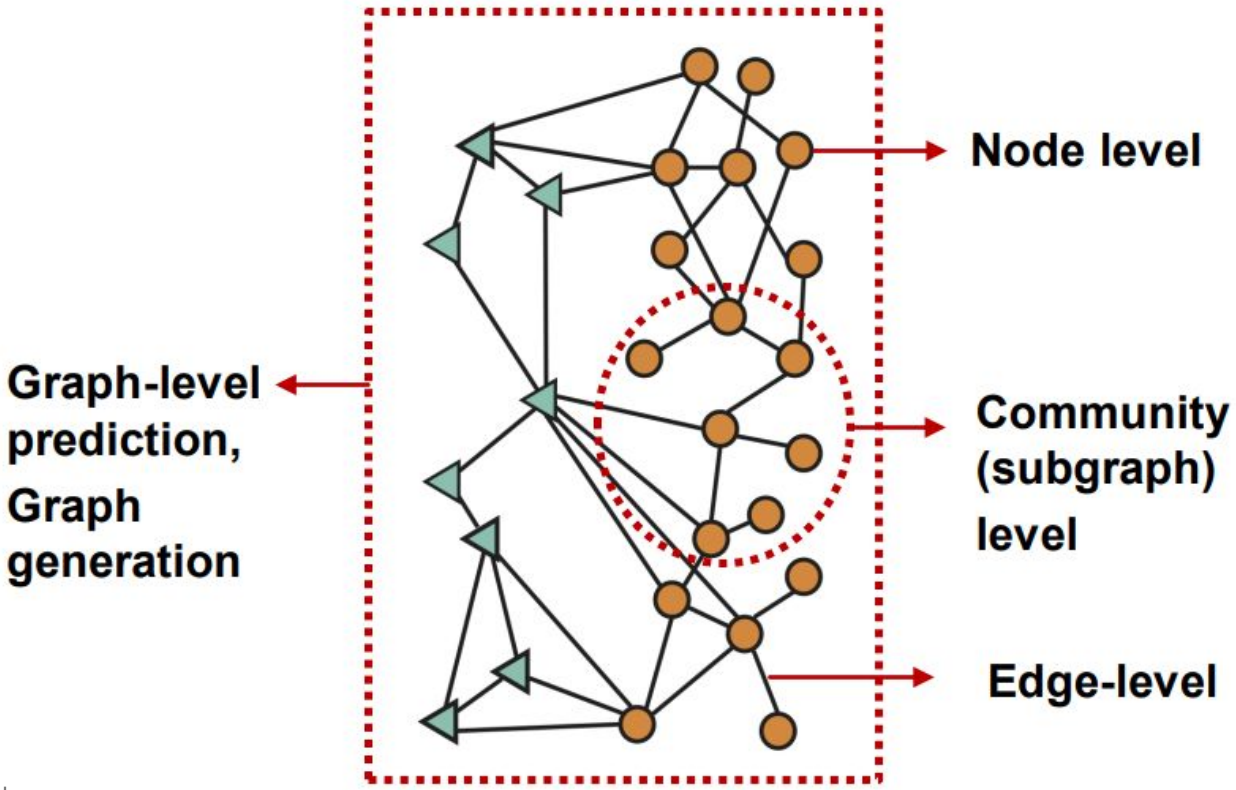
Adapted from CS224W, Jure Leskovec

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]

Adapted from CS224W, Jure Leskovec

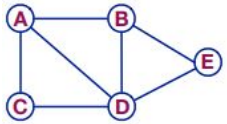
Recap: Different types of ML tasks on graphs



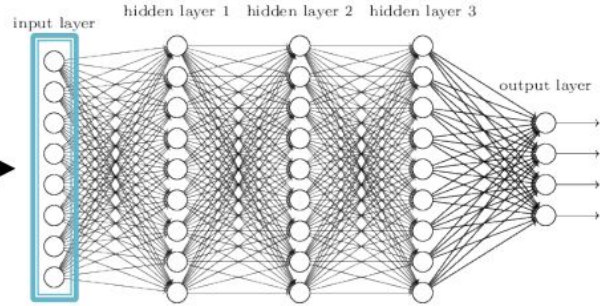
Adapted from CS224W, Jure Leskovec

Idea 1: Fully-connected neural network

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



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



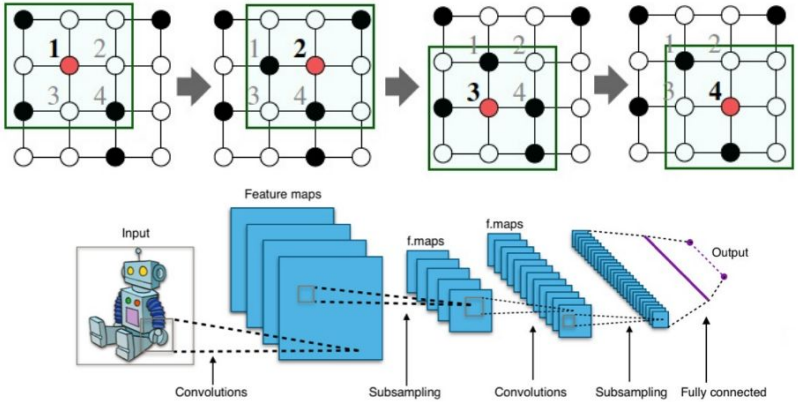
?

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

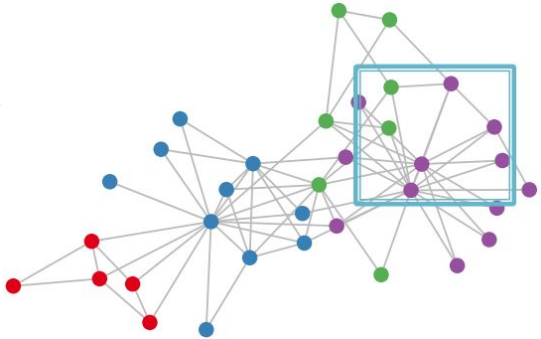
Adapted from CS224W, Jure Leskovec

Idea 2: Convolutional neural network

- CNN for image data



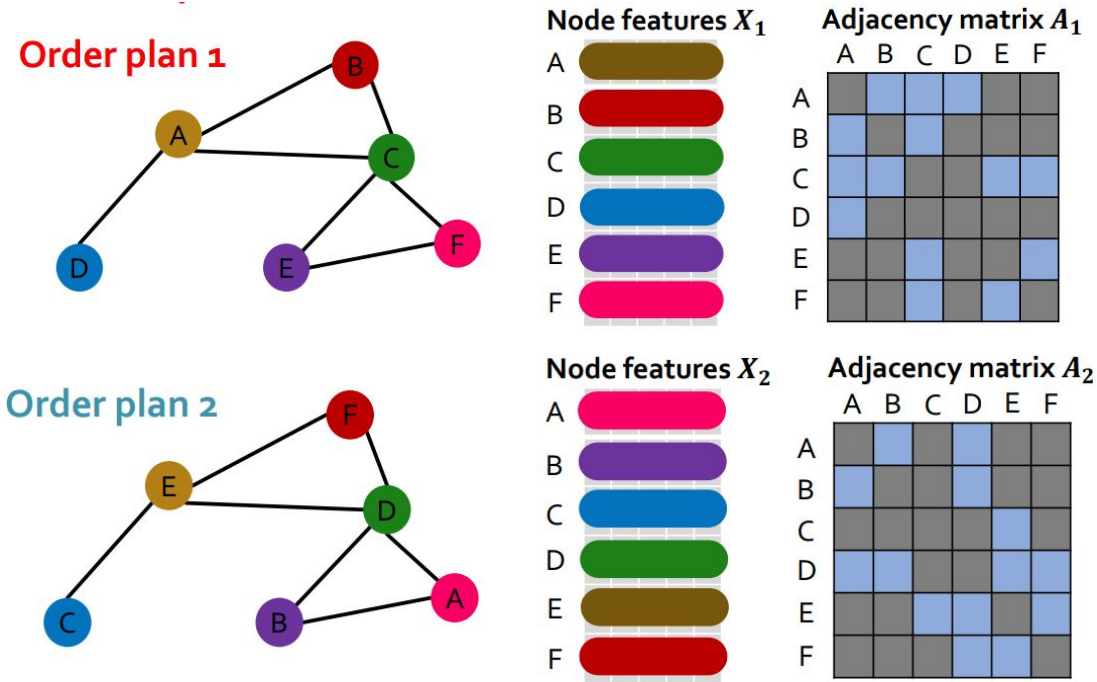
- CNN for Graph data?
  - No fixed notion of locality or sliding window on the graph
  - **Graph is permutation invariant**



Adapted from CS224W, Jure Leskovec

Permutation invariance

- Graph does not have a canonical order of the nodes

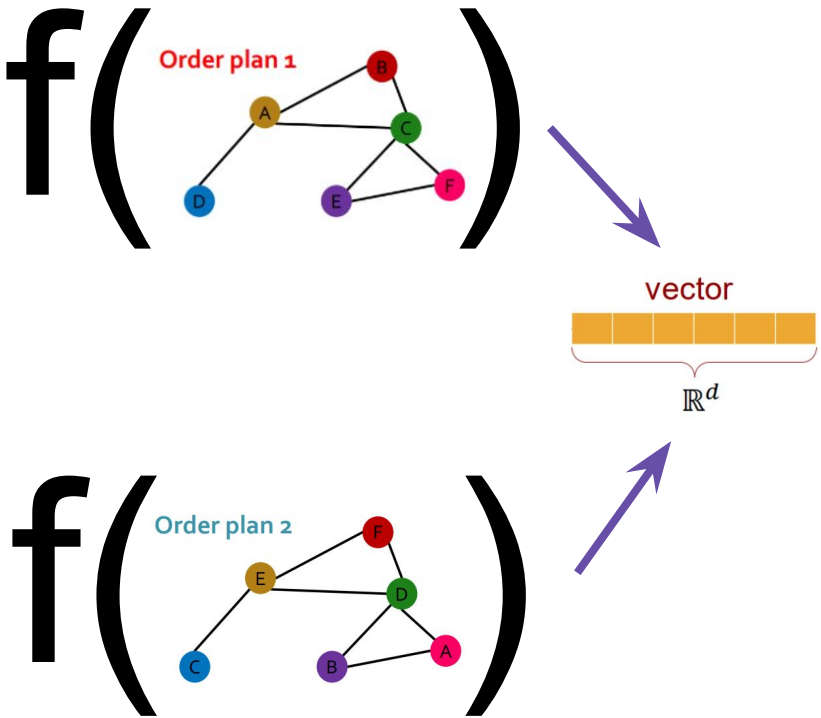


Graph/node representations should be the same for **Order plan 1** and **Order plan 2**

Adapted from CS224W, Jure Leskovec

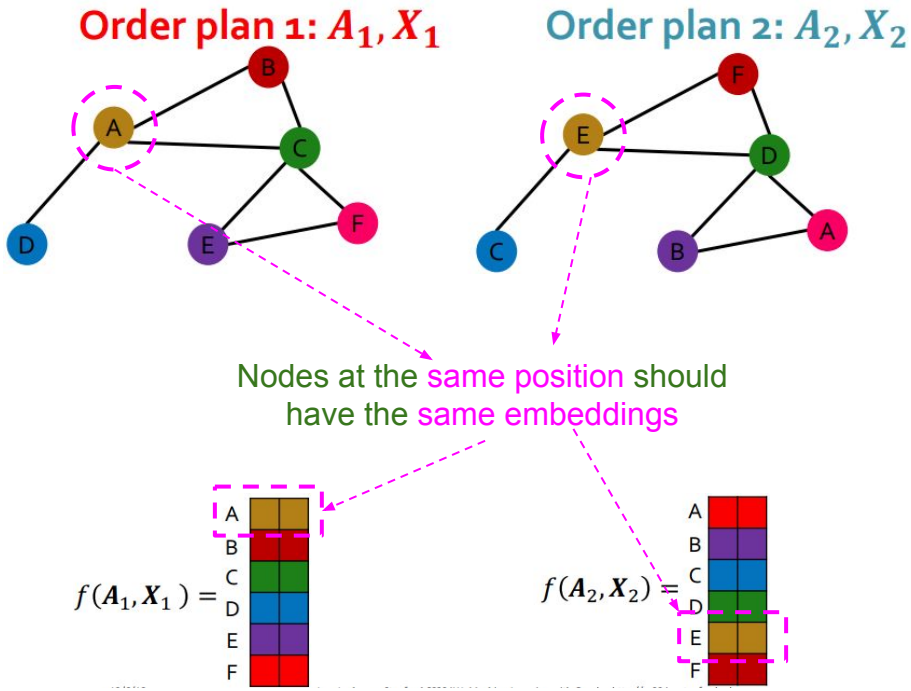
Permutation invariance

- 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 equivariance

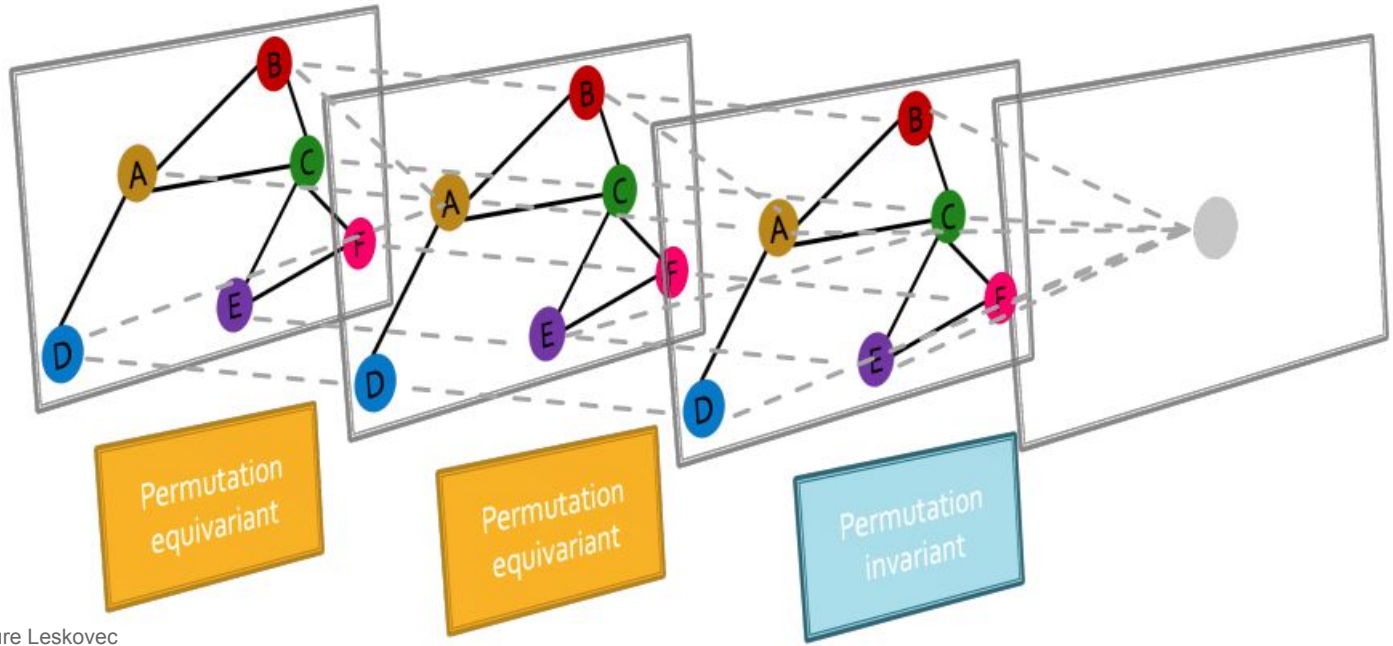
- Consider **node** representation
- Goal learn a function ***f*** that maps a graph ***G*** = (***A***, ***X***) to a vector R^{m*d}
  - m: #nodes, each row is the embedding of a node
- If every pair of nodes at the same position have the same embedding, we say f is a **permutation equivariant function**



Adapted from CS224W, Jure Leskovec

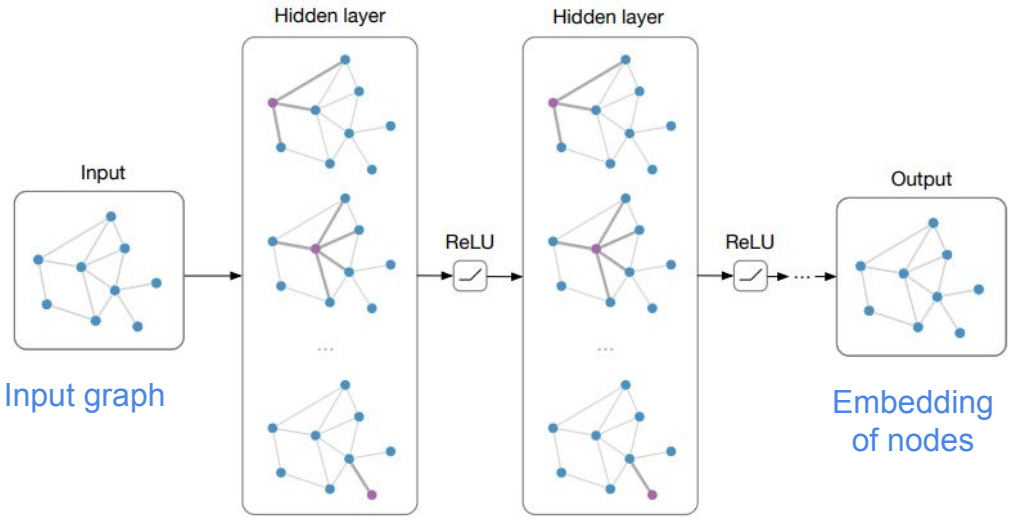
Graph neural networks overview

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



Adapted from CS224W, Jure Leskovec

Graph neural network

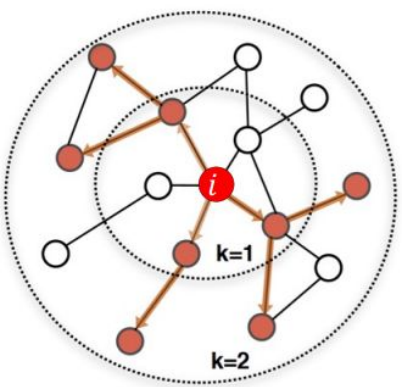


- **Main Idea**: Pass messages between pairs of nodes and agglomerate
- **Alternative Interpretation**: Pass messages between nodes to refine node (and possibly edge) representations

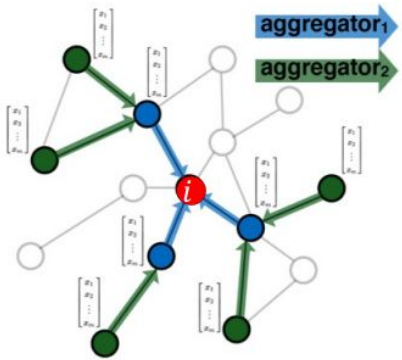
Image credit: Thomas Kipf

Graph neural network

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



Determine node computation graph



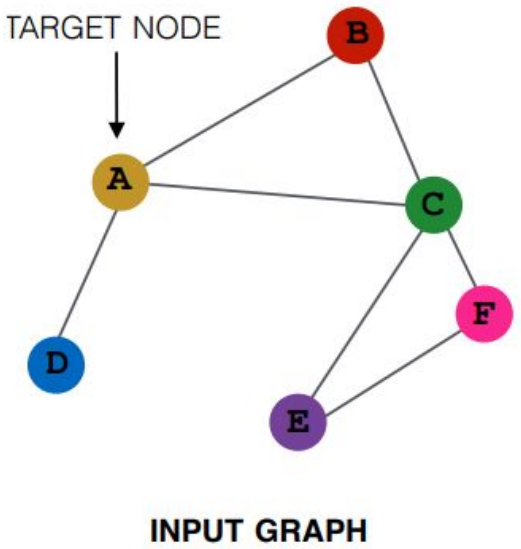
Propagate and transform information

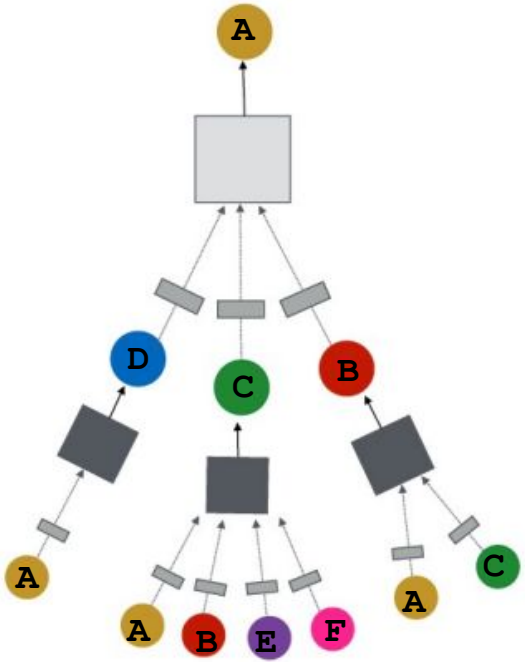
Goal: Learn how to propagate information across the graph to compute node features

Adapted from CS224W, Jure Leskovec

Idea: aggregate information from neighbors

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

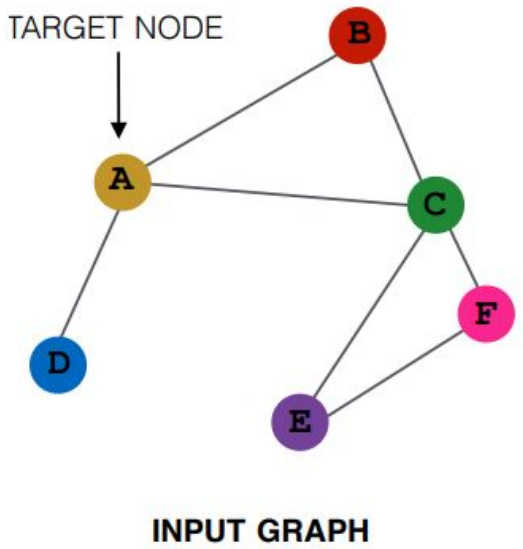


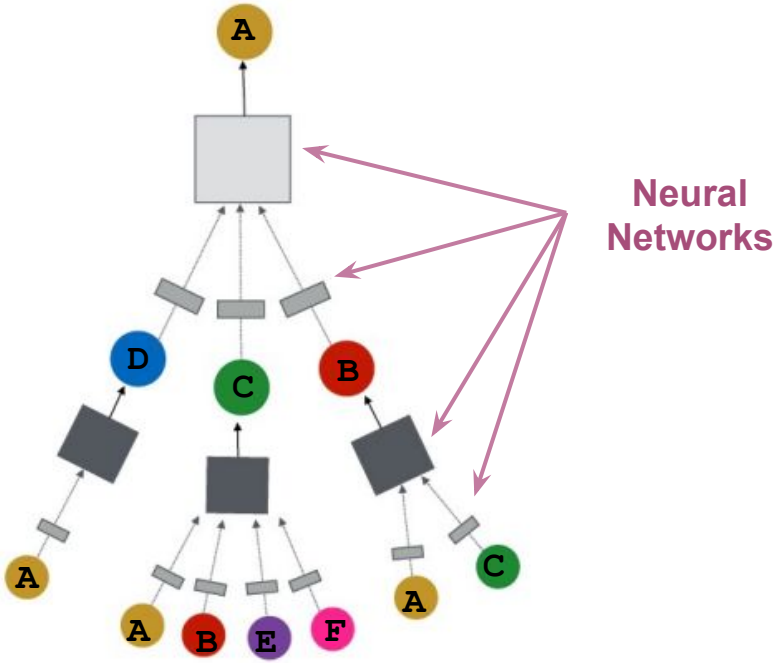


Adapted from CS224W, Jure Leskovec

Idea: aggregate information from neighbors

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

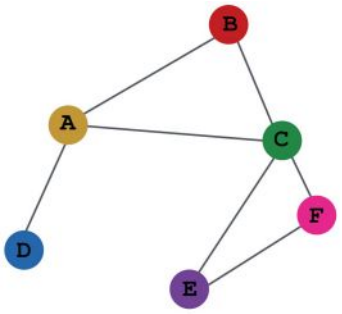




Adapted from CS224W, Jure Leskovec

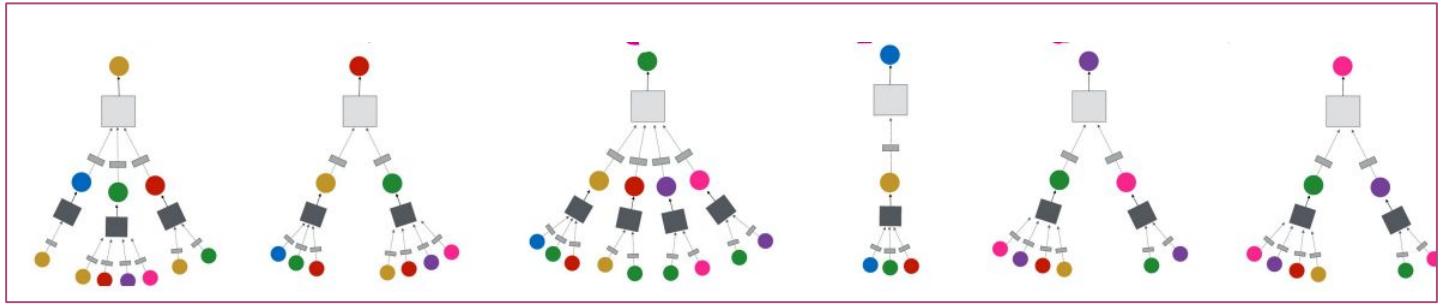
Idea: aggregate information from neighbors

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



INPUT GRAPH

Every node defines a computation graph based on its neighborhood!



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

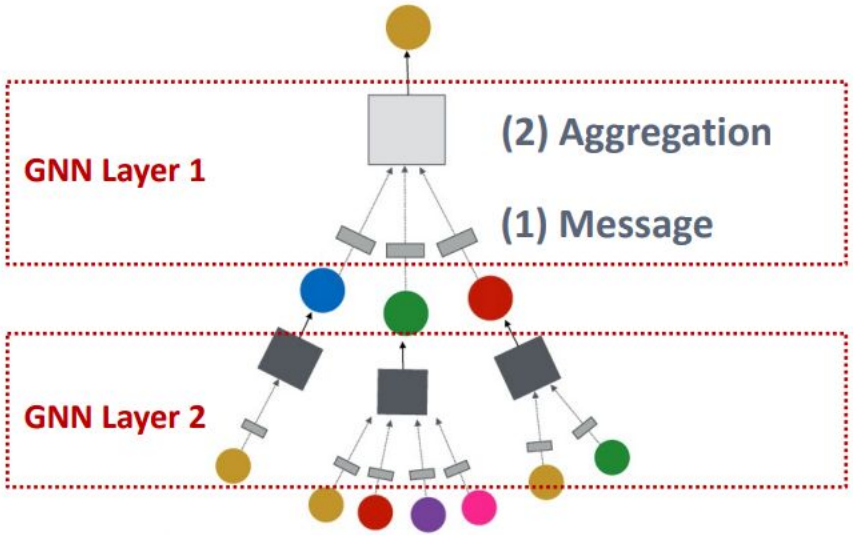
↓

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

Can repeat *K* times

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



Adapted from CS224W, Jure Leskovec

General GNN framework

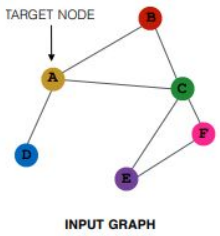
1. Message passing: each node **transforms** its feature vector and **sends** it like a “message” to its neighbors

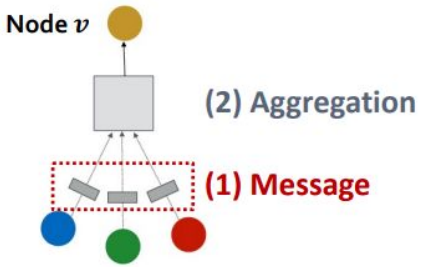
2. Aggregation: each node **receives** feature vectors from its neighbors and **aggregates** them into a single vector

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

Can repeat *K* times

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 $\boldsymbol{x}_u$

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

Aggregation

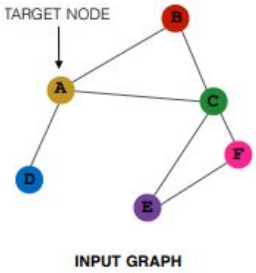
- Aggregation: each node receives feature vectors from its neighbors and aggregates them into a single vector

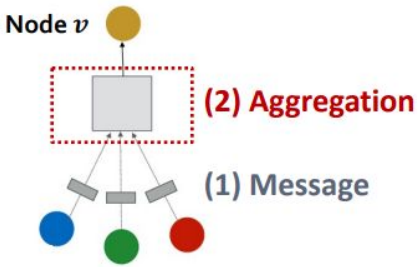
- **Intuition:** Each node will aggregate the messages from node v's neighbors

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

aggregation function

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





Adapted from CS224W, Jure Leskovec

A single GNN layer

Summary:

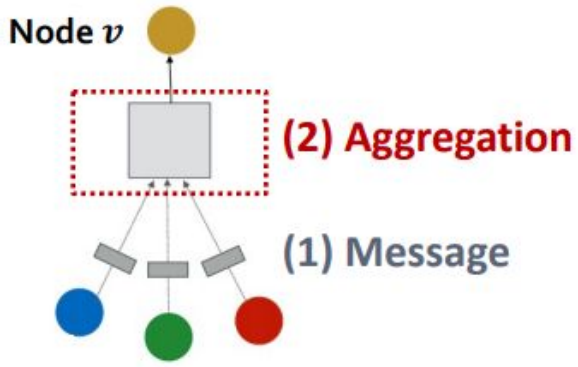
- **(1) Message**: each node computes its own message

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

- **(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

Adapted from CS224W, Jure Leskovec

A single GNN layer

Summary:

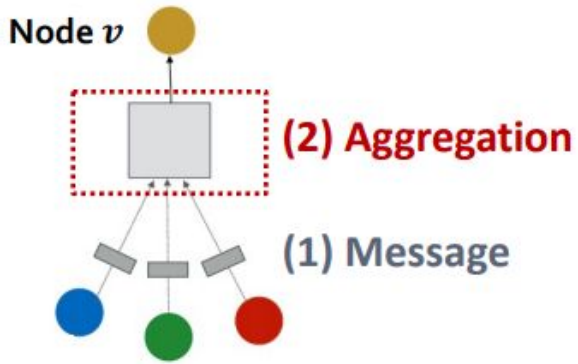
- **(1) Message**: each node computes its own message

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

- **(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()**

Adapted from CS224W, Jure Leskovec

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

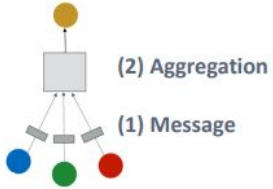
Classical GNN layers: GCN

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.

Adapted from CS224W, Jure Leskovec

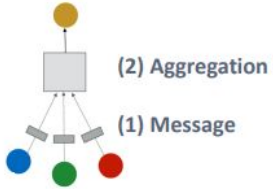
Classical GNN layers: GCN

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.

Adapted from CS224W, Jure Leskovec

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)

Adapted from CS224W, Jure Leskovec

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***

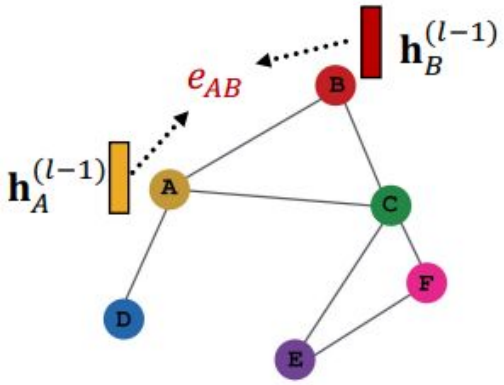
(2) Normalize $\mathbf{e}_{vu}$ into the final attention weight α_{vu}

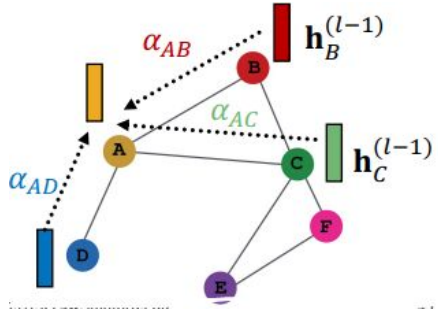
- Use the **softmax** function, so that $\sum_{u \in N(v)} \alpha_{vu} = 1$:

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

(3) Weighted sum based on the final attention weight α_{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 α_{AB}, α_{AC}, α_{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

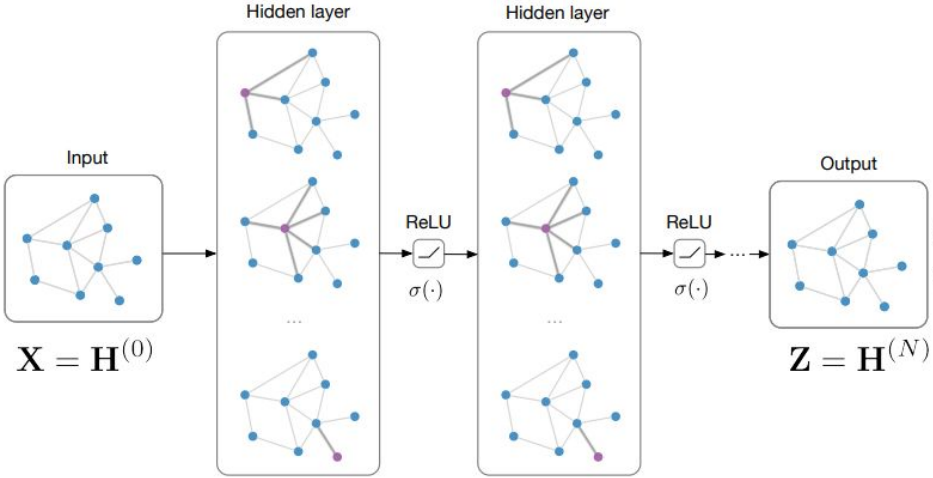
1. Message passing: each node **transforms** its feature vector and **sends** it like a “message” to its neighbors

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

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:

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

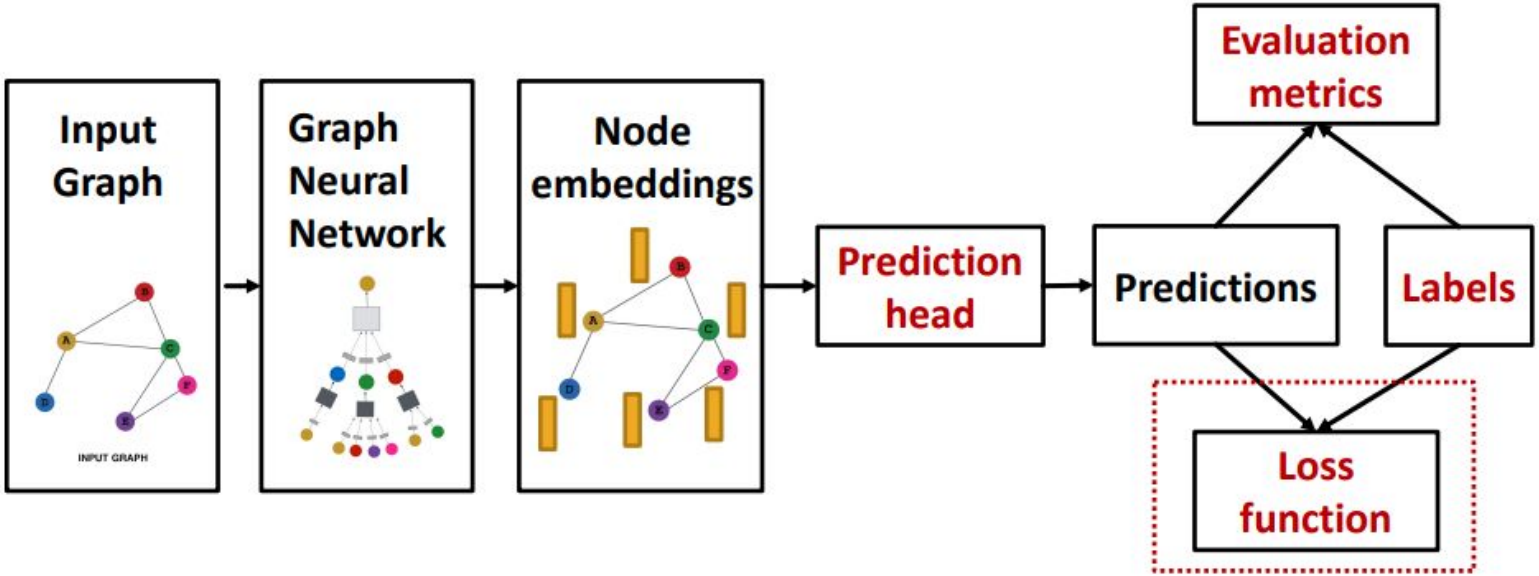
Kipf & Welling (NIPS BDL 2016)

"Graph Auto-Encoders"

Dot product or concatenation

Image credit: Thomas Kipf

Training GNN models



Adapted from CS224W, Jure Leskovec

Classification & Regression

- **Classification**: labels $\boldsymbol{y}^{(i)}$ with **discrete** value
  - E.g., Node classification: which category does a node belong to

- **Regression**: labels $\boldsymbol{y}^{(i)}$ with **continuous** value
  - E.g., predict the drug likeness of a molecular graph

- GNNs can be applied to both settings
- **Differences**: loss function & evaluation metrics

Classification loss

- **Cross entropy (CE)** is a common loss function in classification
  - K-way prediction for i-th data point:

$$\mathrm{CE}\left(\boldsymbol{y}^{(i)}, \hat{\boldsymbol{y}}^{(i)}\right) = -\sum_{j=1}^{K} \boldsymbol{y}_j^{(i)} \log(\hat{\boldsymbol{y}}_j^{(i)})$$

Label Prediction

i-th data point

j-th class

where:

E.g.

0	0	1	0	0

$\boldsymbol{y}^{(i)} \in \mathbb{R}^K$ = one-hot label encoding

$\hat{\boldsymbol{y}}^{(i)} \in \mathbb{R}^K$ = prediction after $\mathrm{Softmax}(\cdot)$

E.g.

0.1	0.3	0.4	0.1	0.1

  - Total loss over all N training examples

$$\mathrm{Loss} = \sum_{i=1}^{N} \mathrm{CE}\left(\boldsymbol{y}^{(i)}, \hat{\boldsymbol{y}}^{(i)}\right)$$

Adapted from CS224W, Jure Leskovec

Regression loss

- **Mean squared loss (MSE)** is a common loss function in regression

K=1 in most of the applications

- K-way regression for data point (i):

$$\mathrm{MSE}\left(\boldsymbol{y}^{(i)}, \hat{\boldsymbol{y}}^{(i)}\right) = \sum_{j=1}^{K} \left(y_j^{(i)} - \hat{y}_j^{(i)}\right)^2$$

i-th data point

j-th target

where:

E.g.

1.4	2.3	1.0	0.5	0.6

$\boldsymbol{y}^{(i)} \in \mathbb{R}^k$ = Real valued vector of targets

$\hat{\boldsymbol{y}}^{(i)} \in \mathbb{R}^k$ = Real valued vector of predictions

E.g.

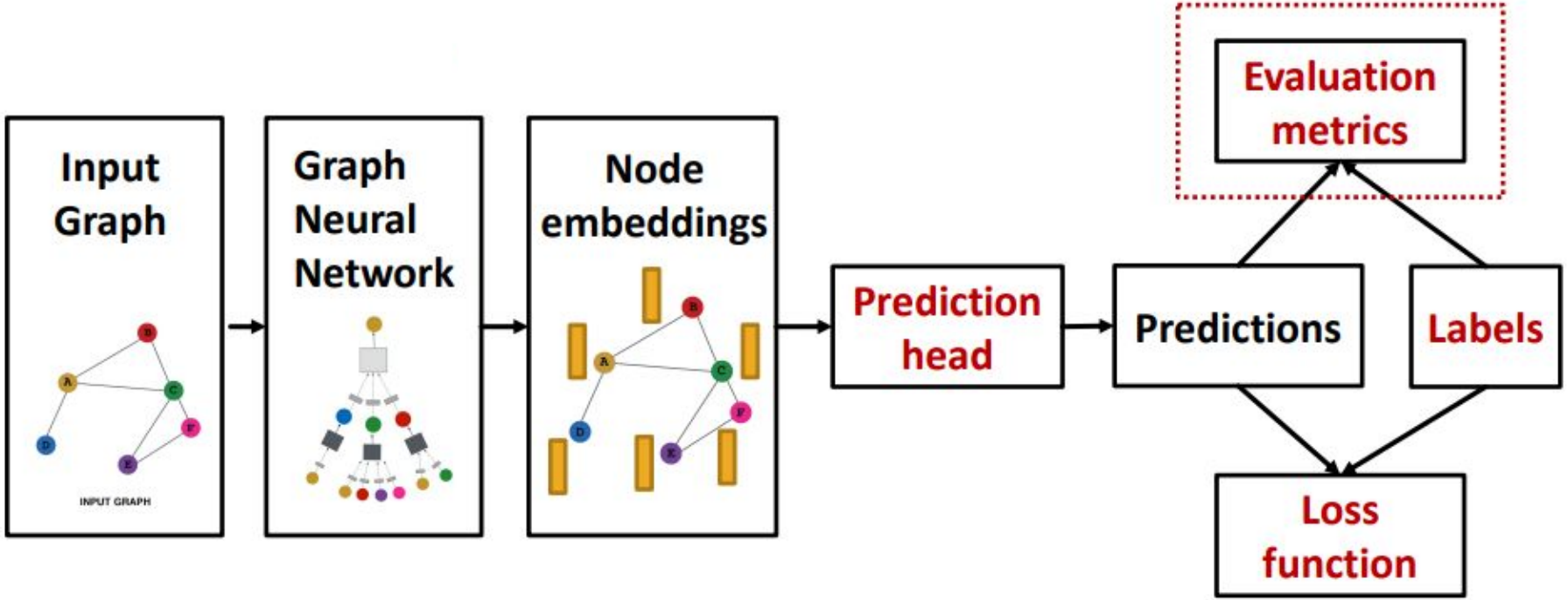
0.9	2.8	2.0	0.3	0.8

- Total loss over all N training examples

$$\text{Loss} = \sum_{i=1}^{N} \mathrm{MSE}\left(\boldsymbol{y}^{(i)}, \hat{\boldsymbol{y}}^{(i)}\right)$$

Adapted from CS224W, Jure Leskovec

GNN evaluation



Adapted from CS224W, Jure Leskovec

Regression metrics

- **Root mean square error (RMSE)**

$$\sqrt{\sum_{i=1}^{N} \frac{(\boldsymbol{y}^{(i)} - \hat{\boldsymbol{y}}^{(i)})^2}{N}}$$

- **Mean absolute error (MAE)**

$$\frac{\sum_{i=1}^{N} \left|\boldsymbol{y}^{(i)} - \hat{\boldsymbol{y}}^{(i)}\right|}{N}$$

$\hat{\boldsymbol{y}}^{(i)}$: model prediction

$\boldsymbol{y}^{(i)}$: true label

Adapted from CS224W, Jure Leskovec

Classification metrics

- **Binary classification**:
  - **Accuracy**:

$$\frac{\mathrm{TP}+\mathrm{TN}}{\mathrm{TP}+\mathrm{TN}+\mathrm{FP}+\mathrm{FN}}=\frac{\mathrm{TP}+\mathrm{TN}}{|\mathrm{Dataset}|}$$

  - **Precision (P)**:

$$\frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FP}}$$

  - **Recall (R)**:

$$\frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}}$$

  - **F1-Score**:

$$\frac{2\mathrm{P}*\mathrm{R}}{\mathrm{P}+\mathrm{R}}$$

Confusion matrix

	Actually Positive (1)	Actually Negative (0)
Predicted Positive (1)	True Positives (TPs)	False Positives (FPs)
Predicted Negative (0)	False Negatives (FNs)	True Negatives (TNs)

- **Multi-class classification**:

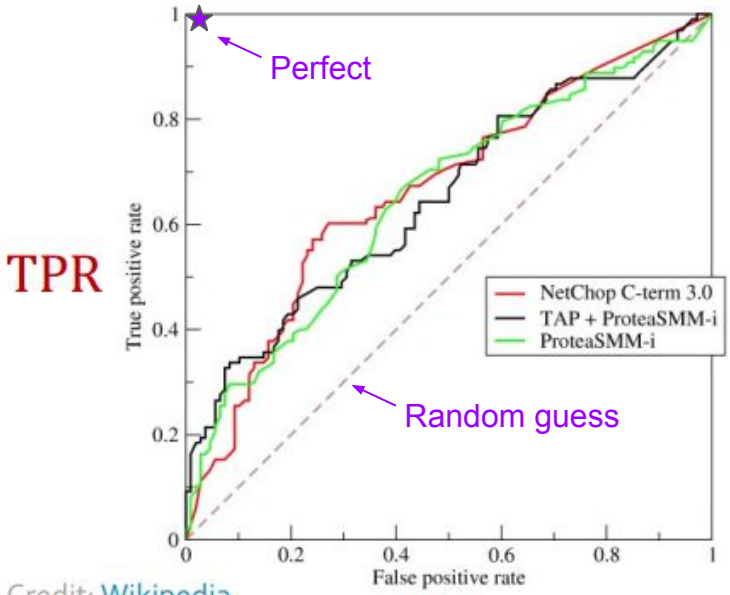
Accuracy

$$\frac{1\left[\operatorname{argmax}\left(\hat{\boldsymbol{y}}^{(i)}\right)=\boldsymbol{y}^{(i)}\right]}{N}$$

Adapted from CS224W, Jure Leskovec

Classification metric: ROC

- **ROC Curve**: Captures the tradeoff in TPR and FPR as the classification threshold is varied for a **binary** classifier.



TPR

FPR

Image Credit: Wikipedia

$$\text{TPR} = \text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

$$\text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}}$$

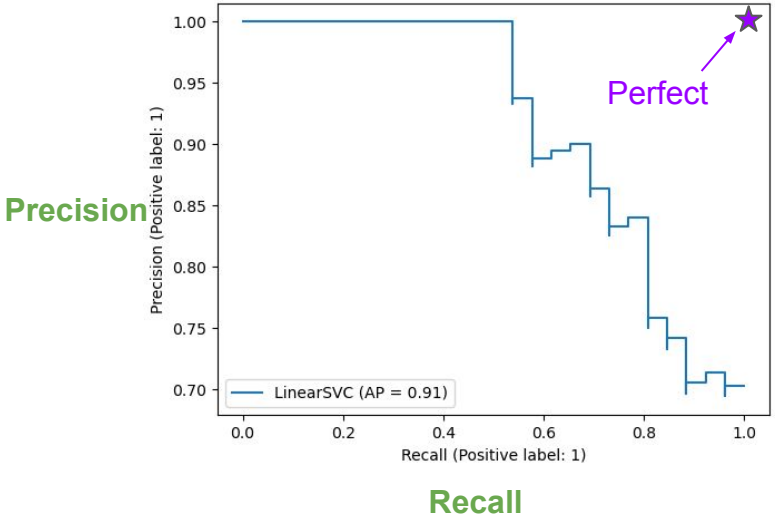
- **Metric**: **AUROC** (Area under the ROC Curve)
  - 1.0: perfect prediction
  - 0.0: worst (random guess)

What if you get a classifier with AUROC = 0.1?

Adapted from CS224W, Jure Leskovec

Classification metric: Precision-Recall curve

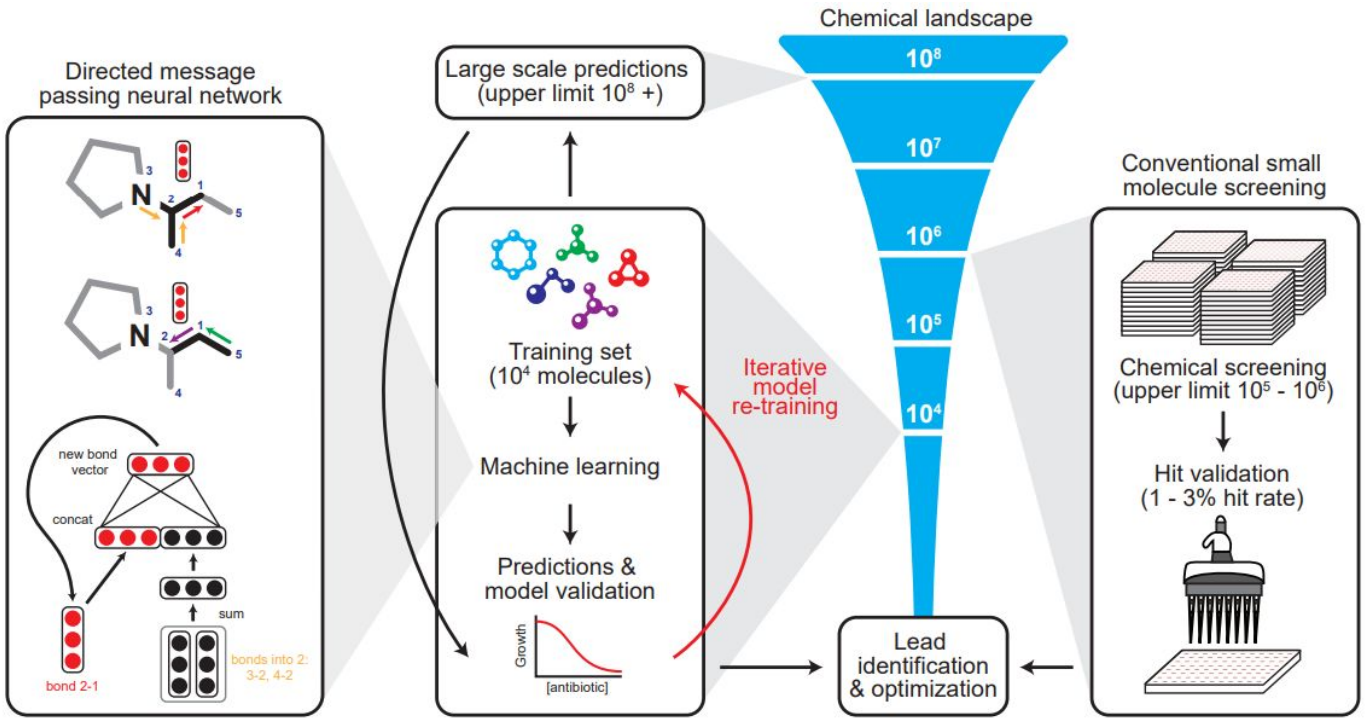
- **PR Curve**: Captures the tradeoff in **Precision** and **Recall** as the classification threshold is varied for a **binary** classifier.



- **Metric**: **AUPR** (Area under the PR curve)
  - 1.0: perfect prediction
  - 0.0: worst

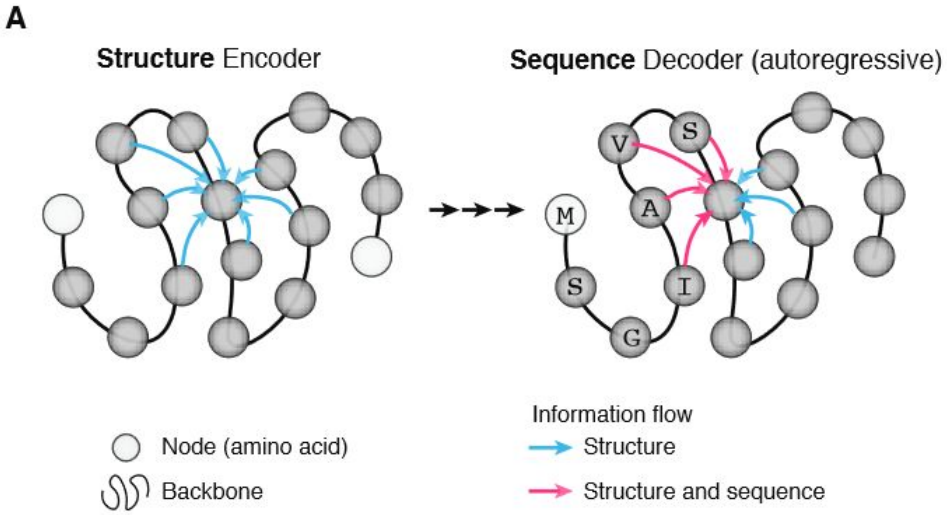
Image credit: sklearn

Applications: antibiotic discovery



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

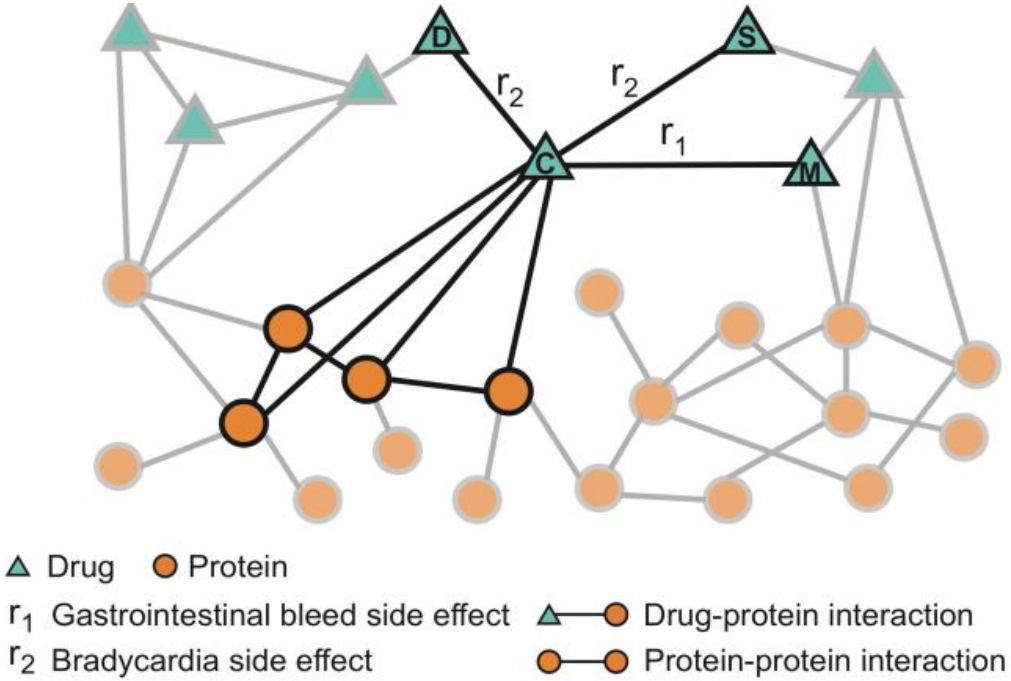
Applications: protein sequence design



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

Summary of today

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