Graph Neural Networks

Neural Networks

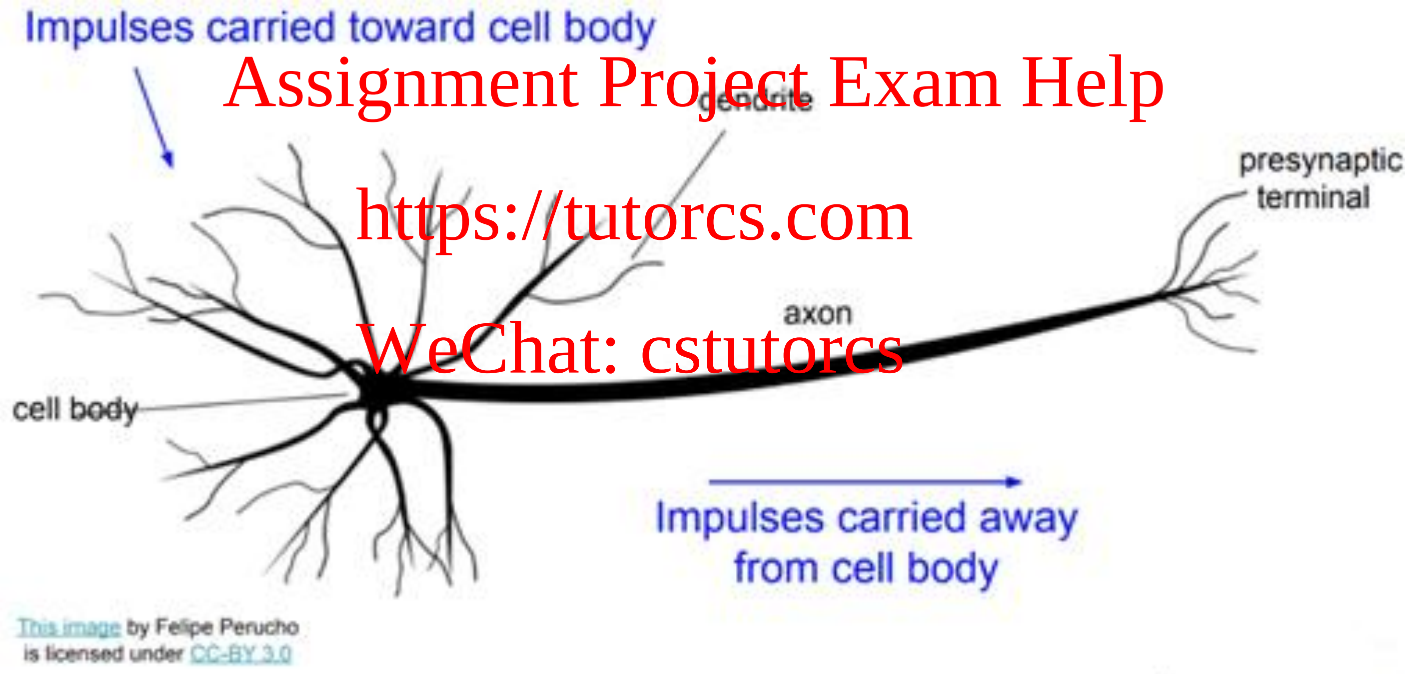
- Thrives in situations where it is challenging to defined rules by hand

- Algorithms that improve automatically through experience.

- The algorithm has a (large) number of parameters whose values need to be learned from the data.

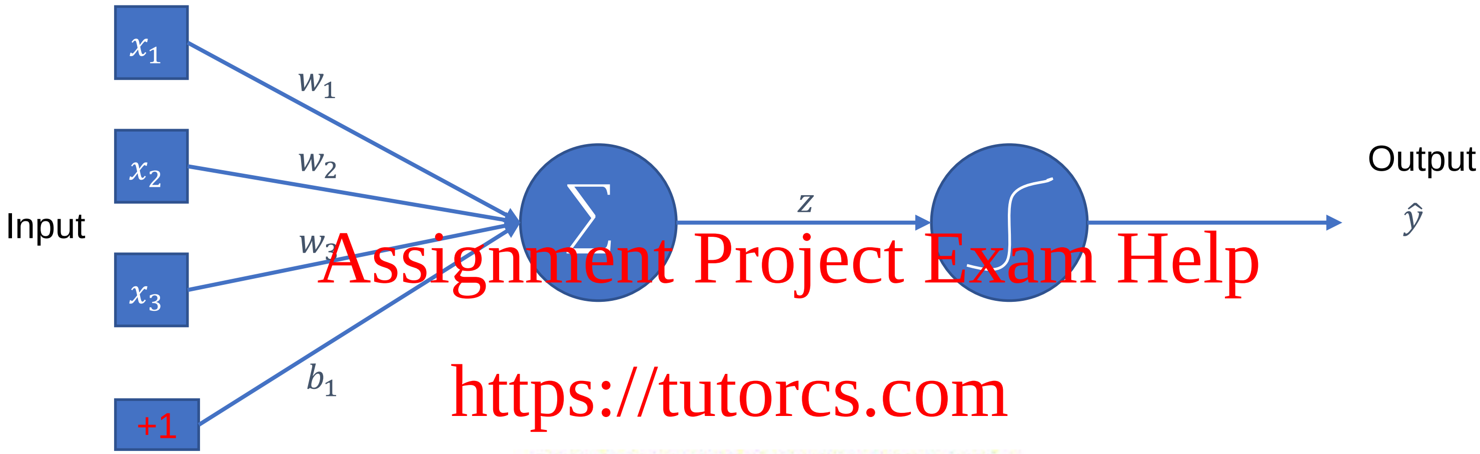
Nuerons

- Inspired by neurons in biology.

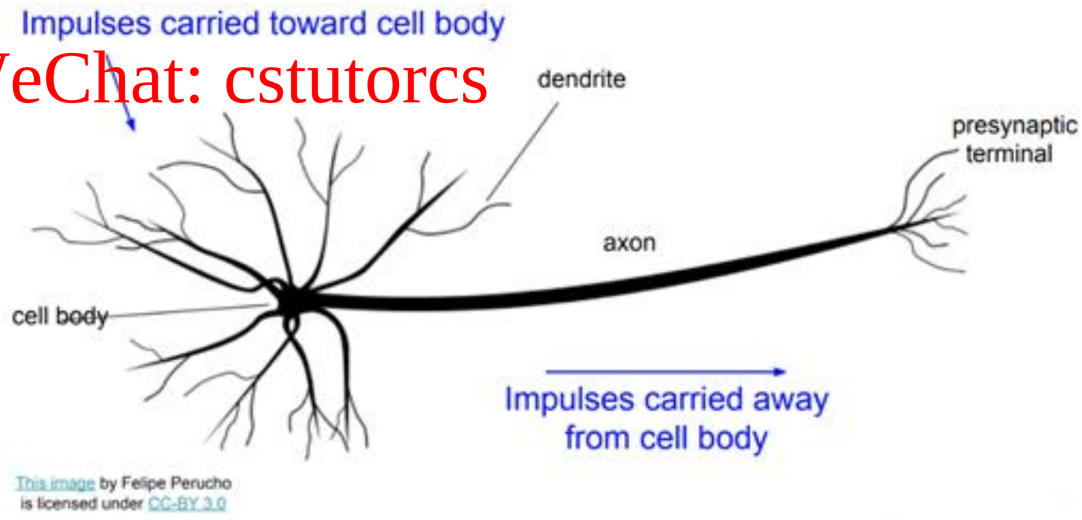


http://cs231n.stanford.edu/slides/winter1516_lecture5.pdf

Perceptron



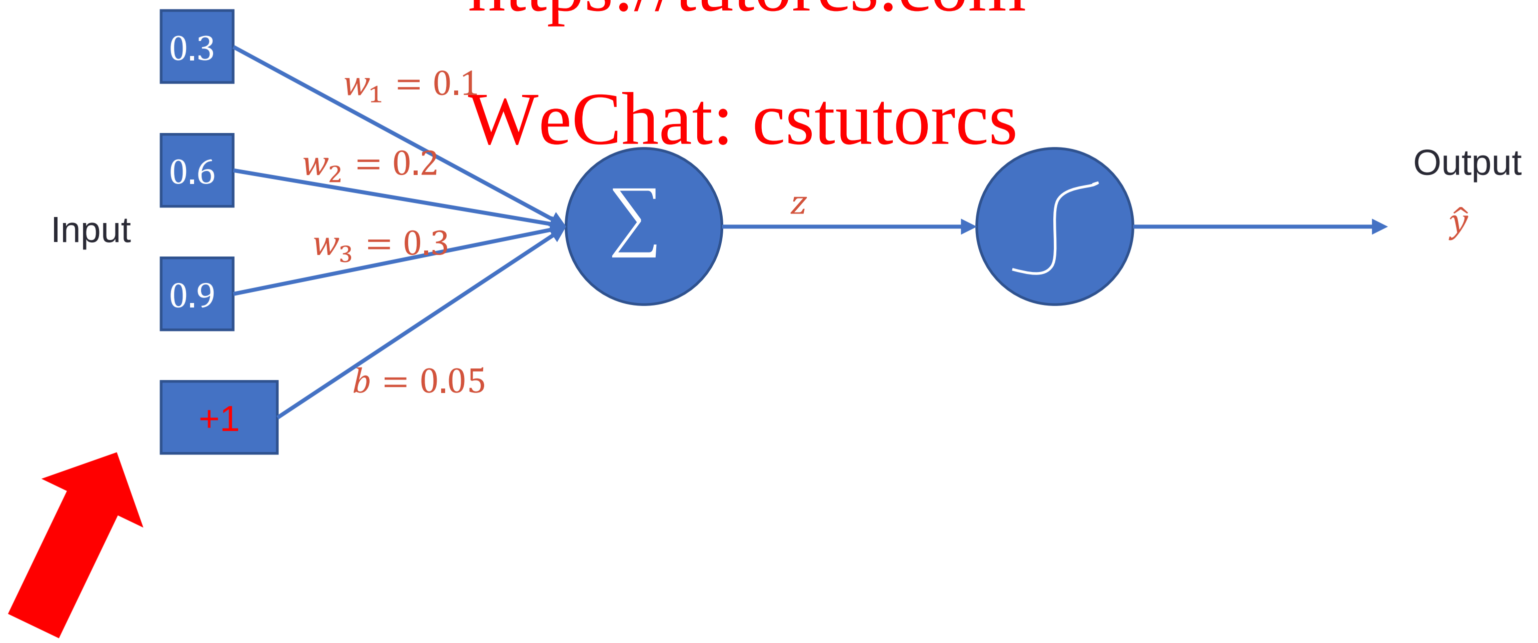
Input
x_1
x_2
x_3
+1
w_1
w_2
w_3
b_1
$\sum$
z
Output
$\hat{y}$



Impulses carried toward cell body
dendrite
presynaptic terminal
axon
cell body
Impulses carried away from cell body
This image by Felipe Perucho is licensed under CC-BY 3.0

What does a Perceptron do? (1)

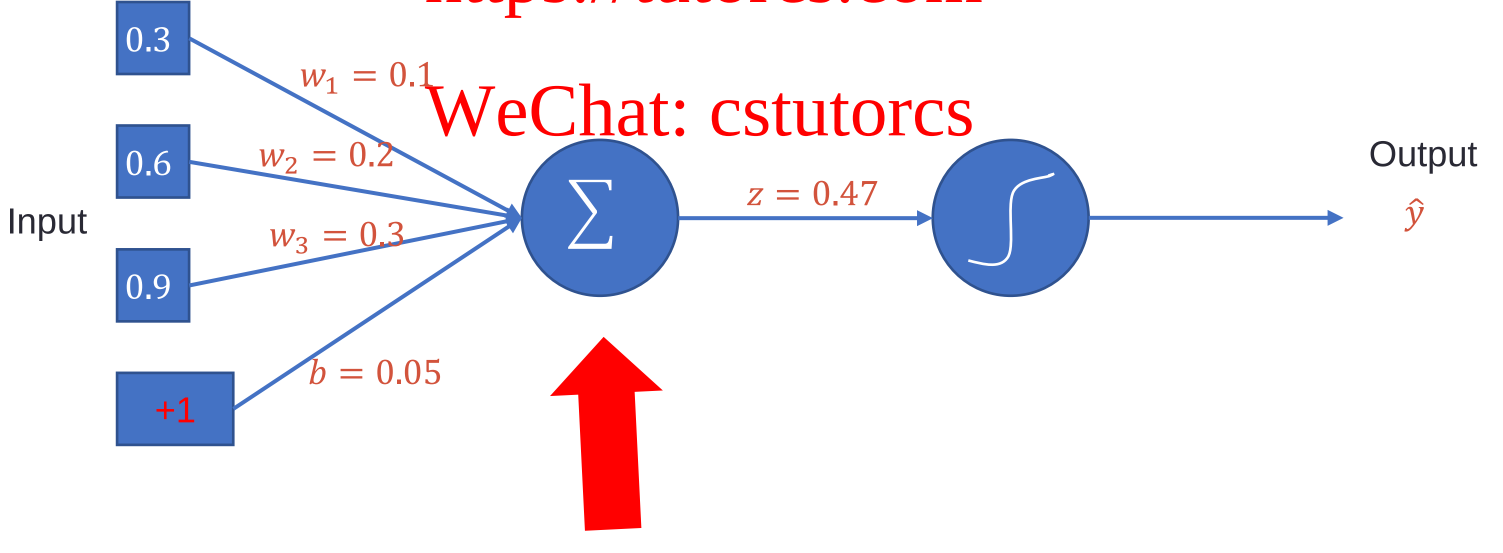
- Suppose a NN initialized to weight w be (0.1, 0.2, 0.3) & bias $b = 0.05$
- **Step0**: Take an input x (0.3, 0.6, 0.9)



What does a Perceptron do? (2)

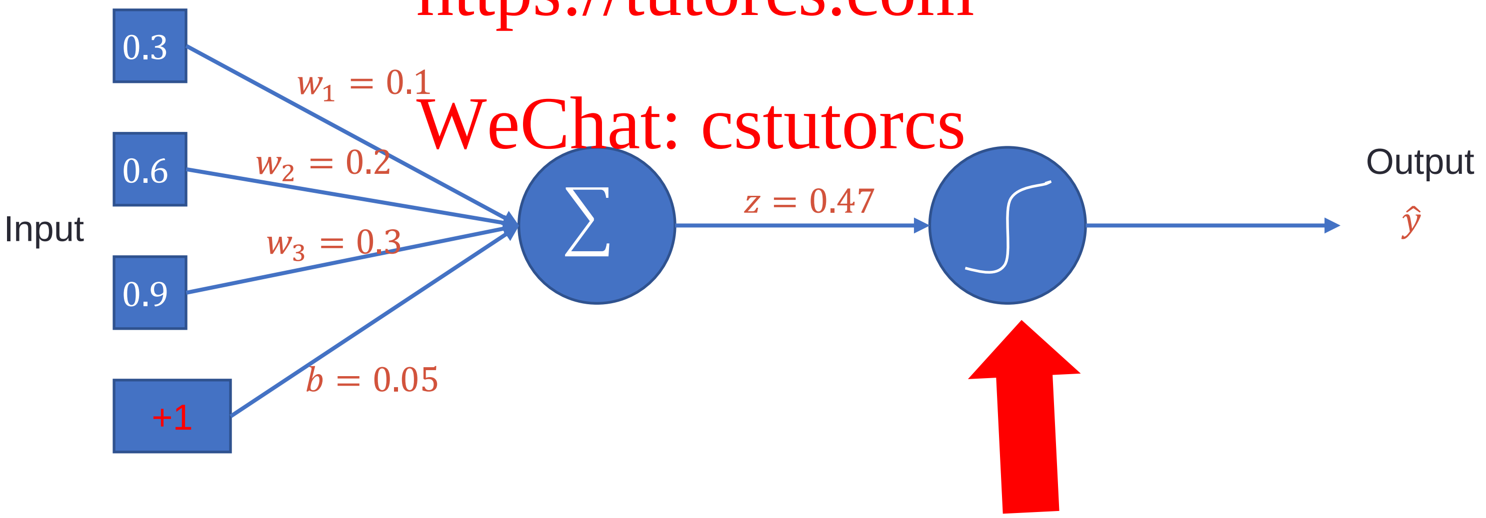
- **Step1**: Calculate a weighted sum

$$z = w^T x + b;\ z = 0.1 \times 0.3 + 0.2 \times 0.6 + 0.3 \times 0.9 + 0.05 = 0.47$$



What does a Perceptron do? (3)

- **Step2**: Apply an activation function



The Maths

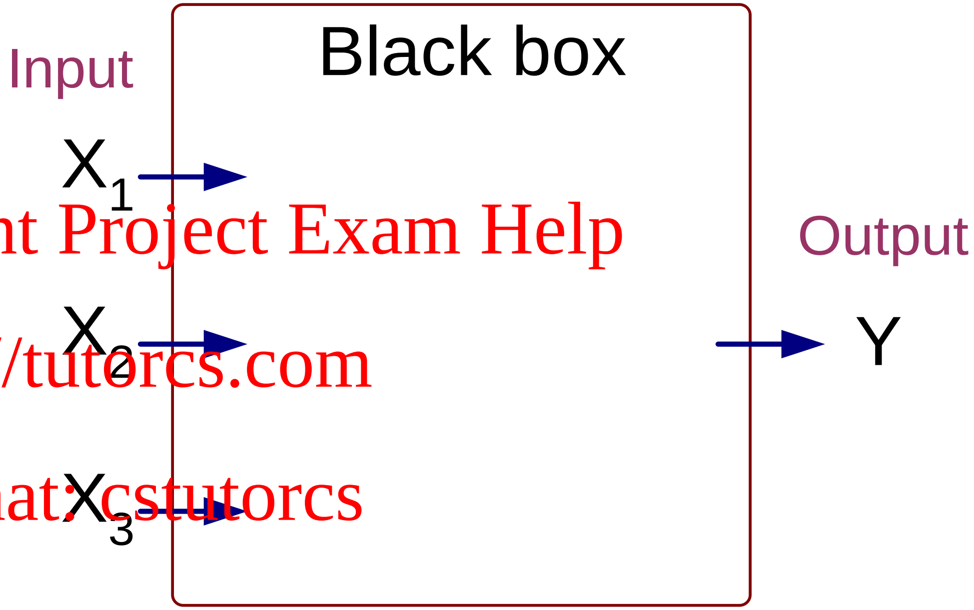
Mathematically: the computation of a neuron is shown below:

$$z = w^T x + b$$

We use a simple step rule as the activation function here.

Neural Networks

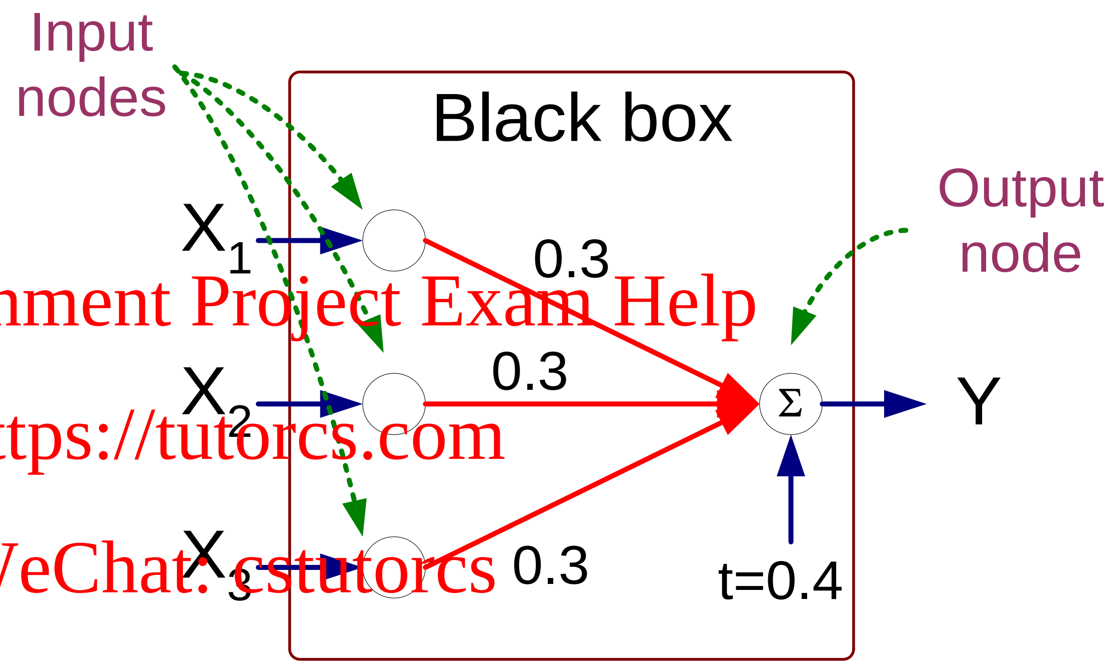
X_1	X_2	X_3	Y
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1
0	0	1	0
0	1	0	0
0	1	1	1
0	0	0	0



Output Y is 1 if at least two of the three inputs are equal to 1.

Neural Networks

X_1	X_2	X_3	Y
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1
0	0	1	0
0	1	0	0
0	1	1	1
0	0	0	0

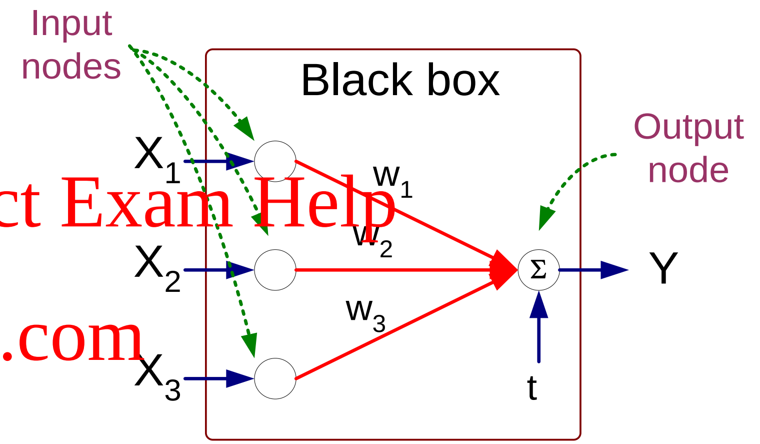


$$Y = I(0.3X_1 + 0.3X_2 + 0.3X_3 - 0.4 > 0)$$

$$\text{where } I(z) = \begin{cases} 1 & \text{if } z \text{ is true} \\ 0 & \text{otherwise} \end{cases}$$

Neural Networks

- Model is an assembly of inter-connected nodes and weighted links
- Output node sums up each of its input value according to the weights of its links
- Compare output node against some threshold t
- The sign function (activation function) outputs a value +1 if its argument is positive and -1 otherwise.



Perceptron Model

$$Y = I\left(\sum_i w_i X_i - t\right) \quad \text{or}$$

$$Y = sign\left(\sum_i w_i X_i - t\right)$$

Perceptron Learning

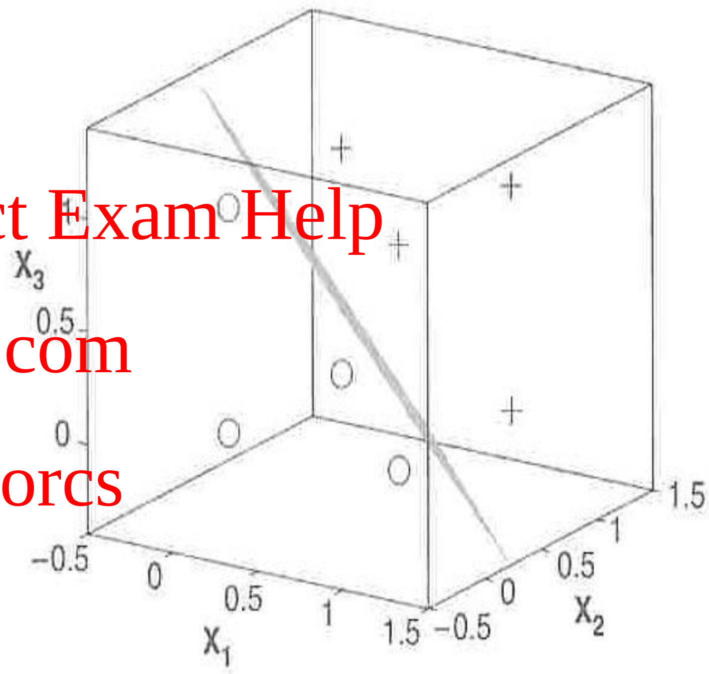
- $\hat{\boldsymbol{y}} = sign[w_d x_d + w_{d-1} x_{d-1} + \cdots + w_1 x_1 + w_0 x_0]$ $= sign(\boldsymbol{w} \cdot \boldsymbol{d})$ where $w_0 = -t, x_0 = 1.$
- $\boldsymbol{\lambda}$ is a parameter known as the learning rate and is between 0 and 1.

Algorithm 5.4 Perceptron learning algorithm.


```
Let D = {(x_i, y_i) | i = 1, 2, ..., N} be the set of training examples.
Initialize the weight vector with random values, w^(0)
repeat
   for each training example (x_i, y_i) ∈ D do
      Compute the predicted output ŷ_i^(k)
      for each weight w_j do
         Update the weight, w_j^(k+1) = w_j^(k) + λ(y_i − ŷ_i^(k)) x_ij.
      end for
   end for
until stopping condition is met
```

Perceptron Decision Boundary

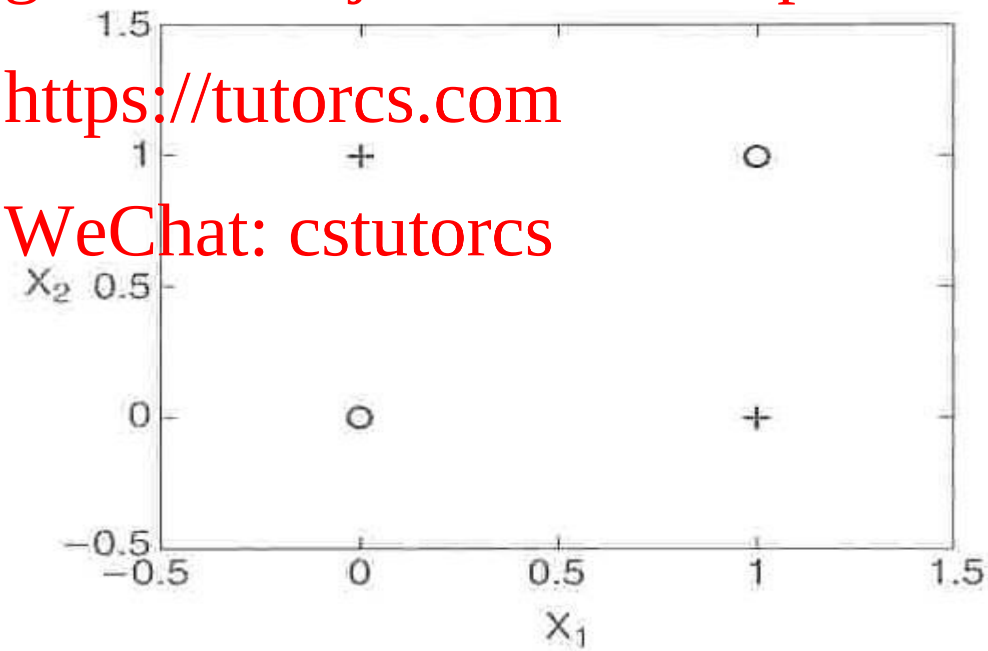
- The previous slide shows a perception model which is linear.
- The figure on the right shows the decision boundary by $\hat{y} = 0$.
- It is a linear hyperplane that separates the data into two classes, -1 and +1.



Nonlinear Hyperplane (XOR Problem)

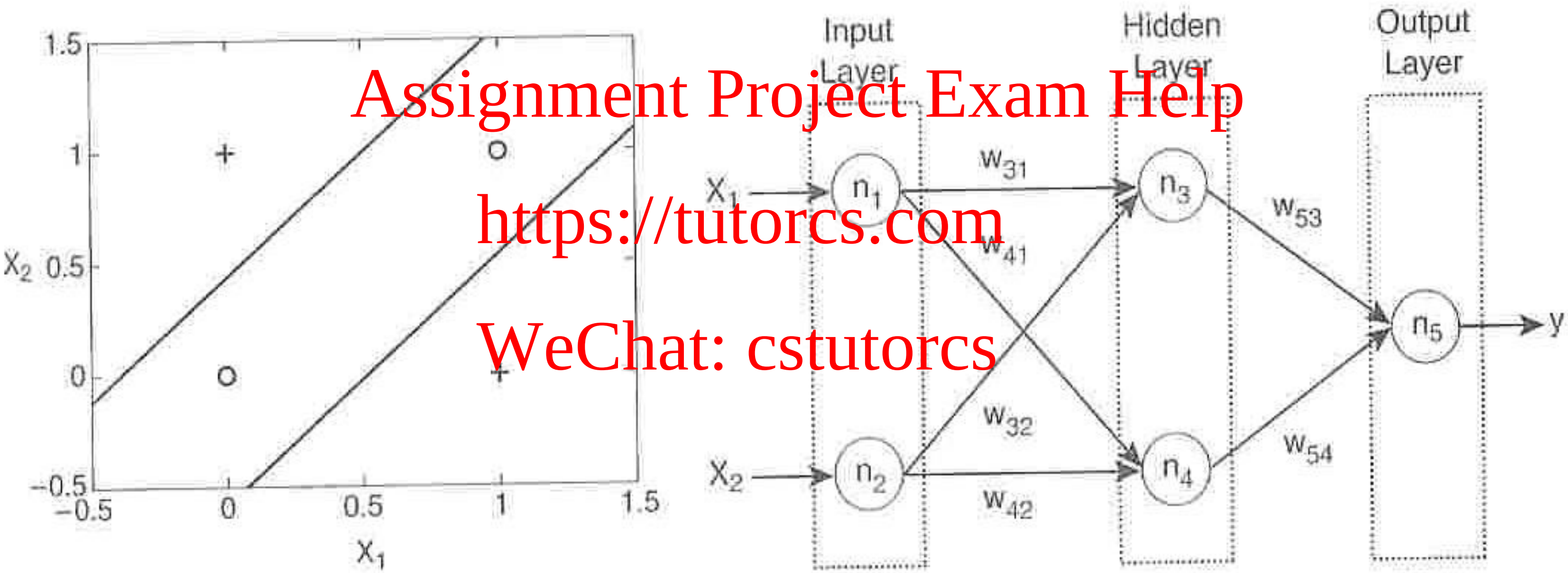
- Consider an example of nonlinearly separable data by the XOR function. The linear perception model cannot fid the solution for it.

X_1	X_2	y
0	0	−1
1	0	1
0	1	1
1	1	−1



Two-Layers for XOR Problem

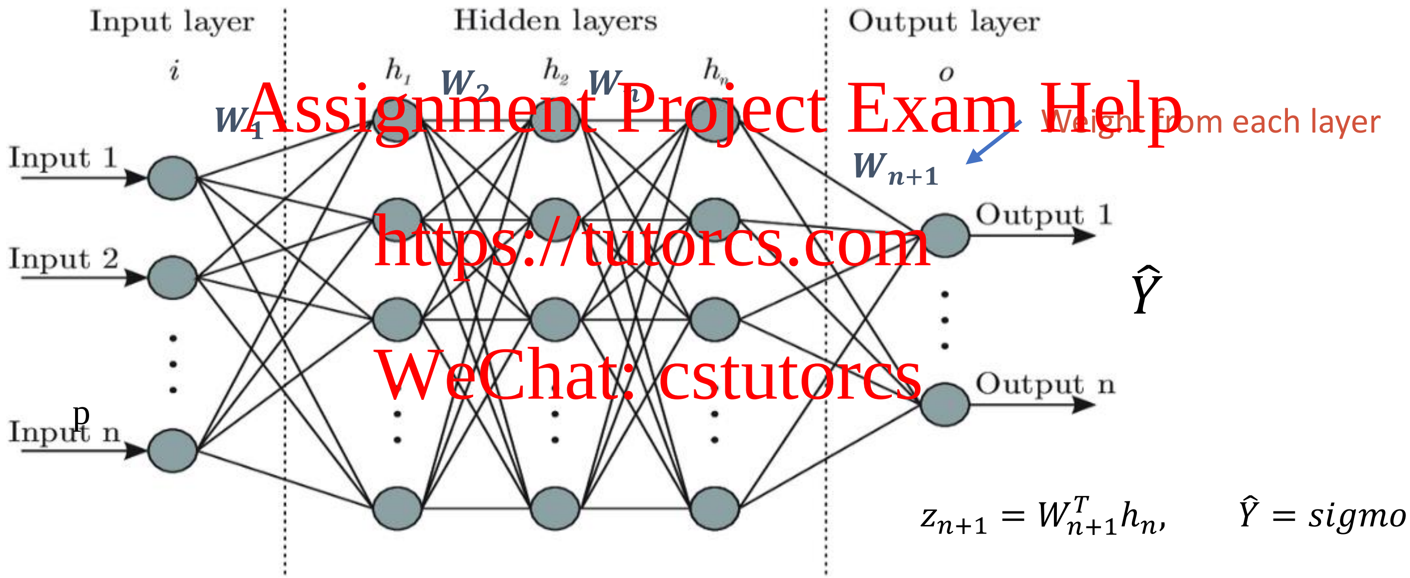
- It uses a two-layer, feed-forward ANN.



(a) Decision boundary. (b) Neural network topology.

Classic Architecture

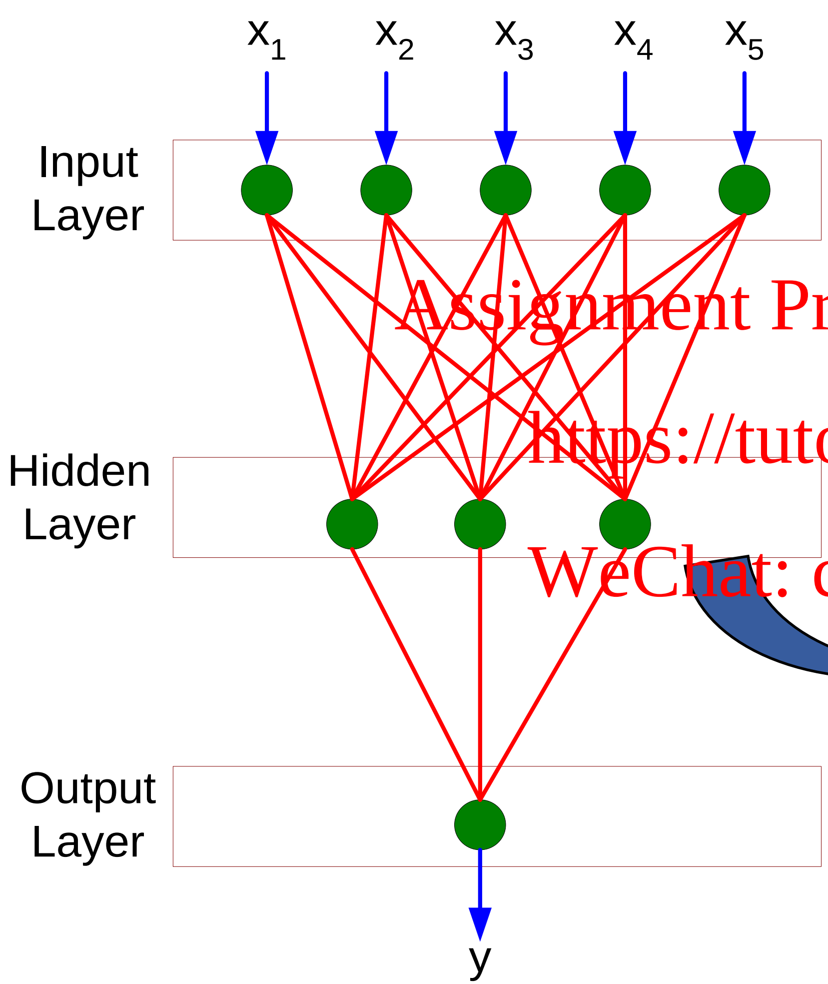
- Multilayer Perceptrons (MLP)

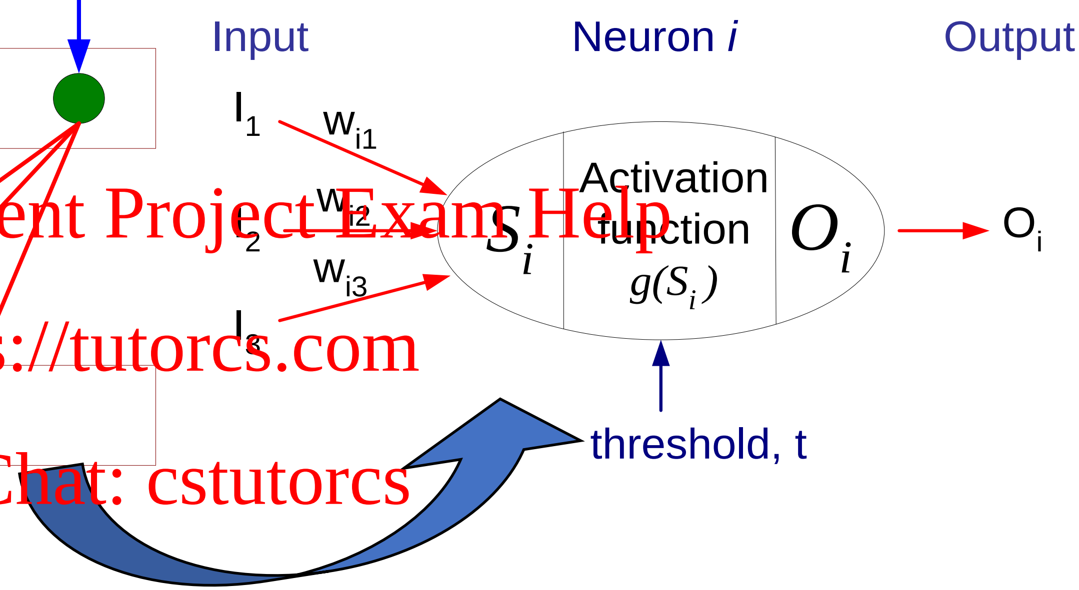


$z_{n+1} = W_{n+1}^T h_n, \qquad \hat{Y} = sigmoid(z_{n+1})$

$z_1 = W_1^T X, \;\; h_1 = sigmoid(z_1) \qquad z_2 = W_2^T h_1, \;\; h_2 = sigmoid(z_2)$

Classic Architecture

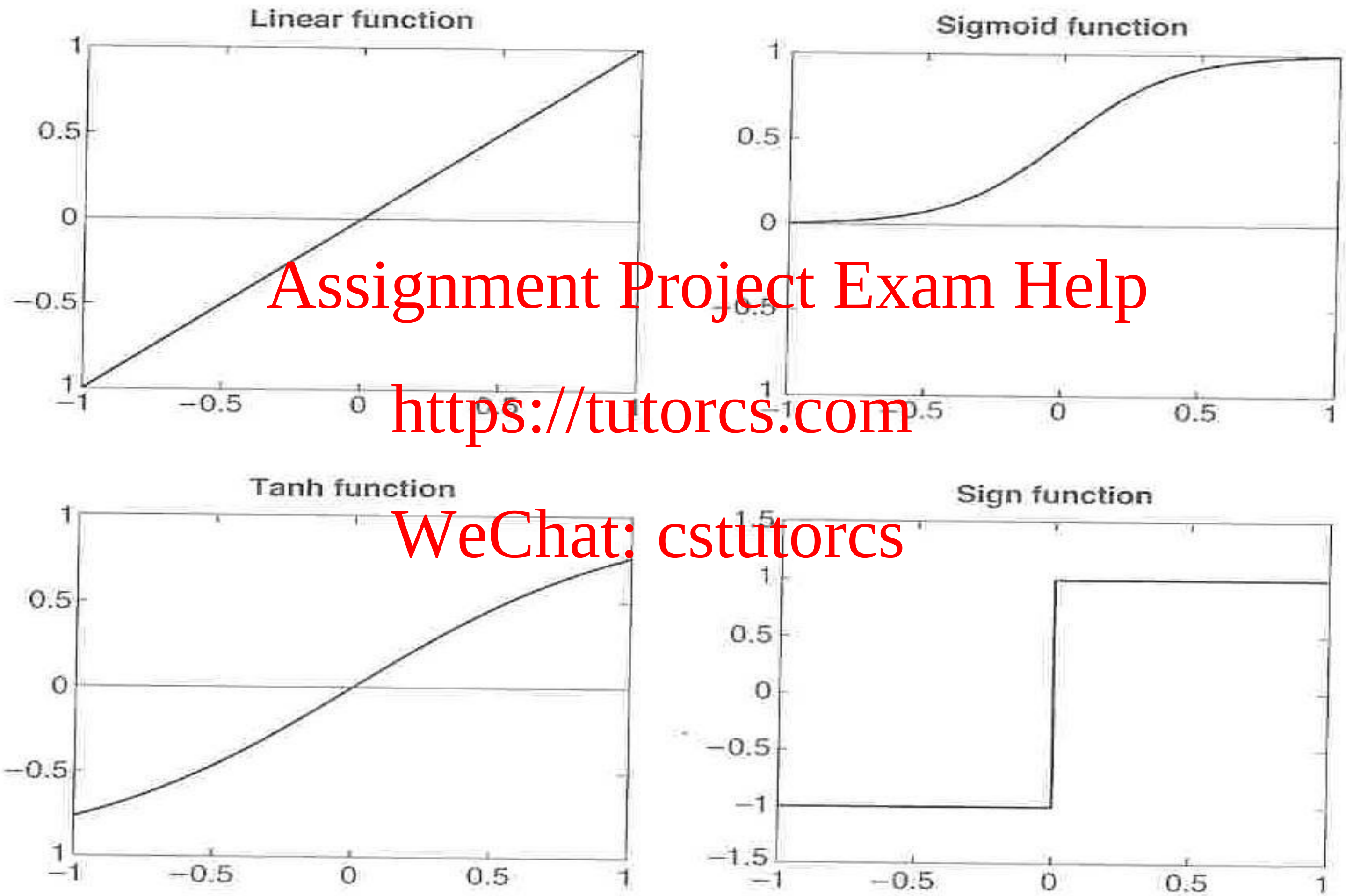




Training ANN means learning the weights of the neurons

This is a feed-forward ANN.

Activation Functions



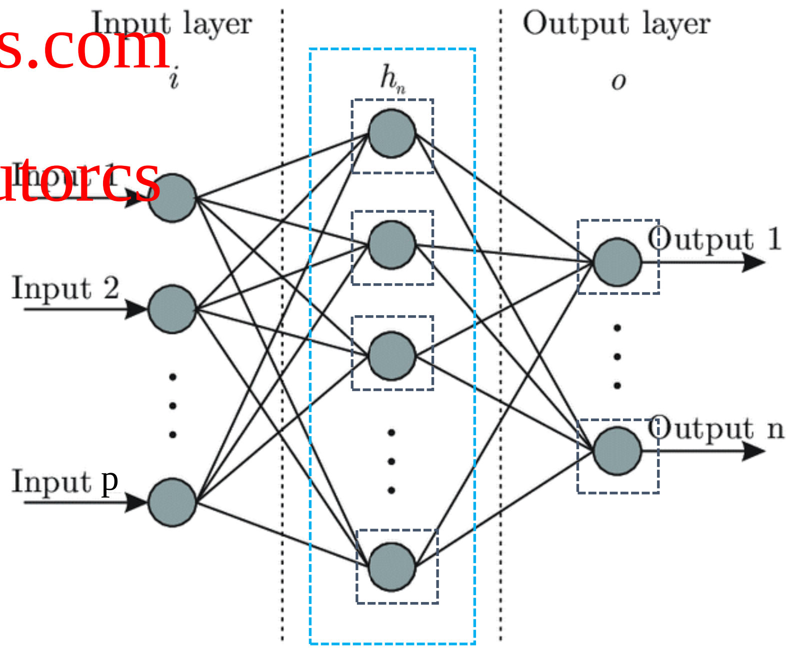
Increased Expressive Power

From Perceptrons to NN

- Perceptrons are a basic unit of a neural network.
- 1-layered neural network on the right

Structure:

- Input layer, output layer,
- Middle are hidden layers.

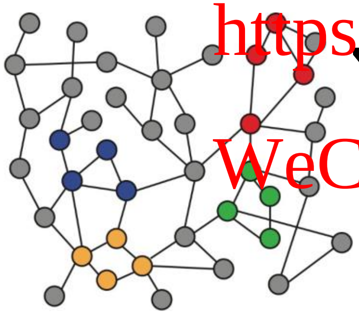
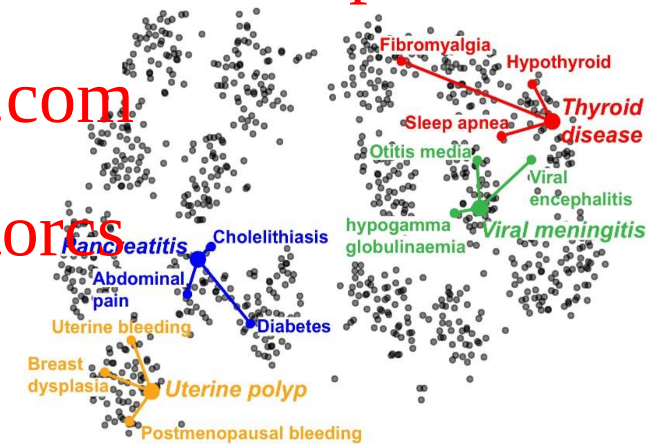


Design Issues in NN Learning

- The number of nodes in the input layer should be determined.
- Generally, the number of nodes in the output layer can be 2 for binary classification, and k for k-class classification.
- The network topology
  - The number of hidden layers and hidden nodes
  - Feed-forward or recurrent network
  - Activation function
- Weights and biases need to be initialized

Recap: Node Embeddings

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

f() = 

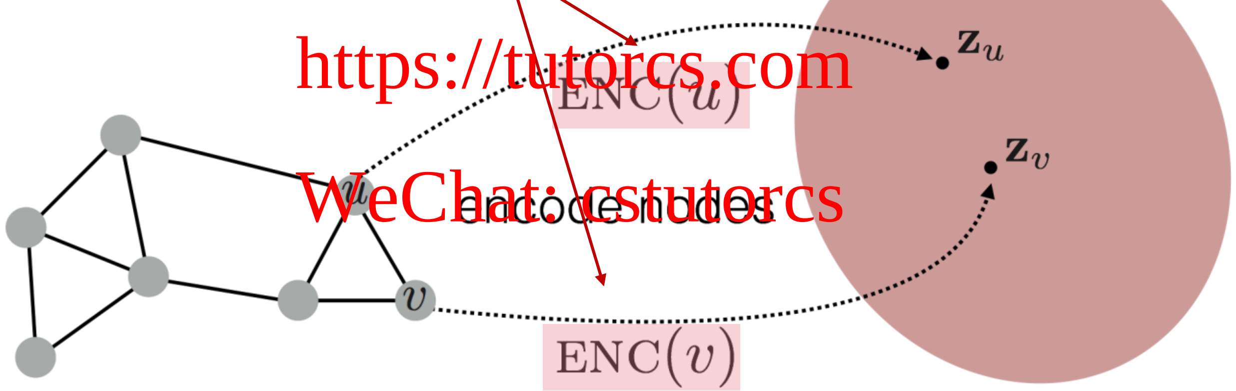
Input graph

2D node embeddings

Recap: Node Embeddings

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

Need to define!



Input network

d-dimensional embedding space

Recap: Two Key Components

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

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

d-dimensional embedding

node in the input graph

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

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

Decoder

Similarity of u and v in the original network

dot product between node embeddings

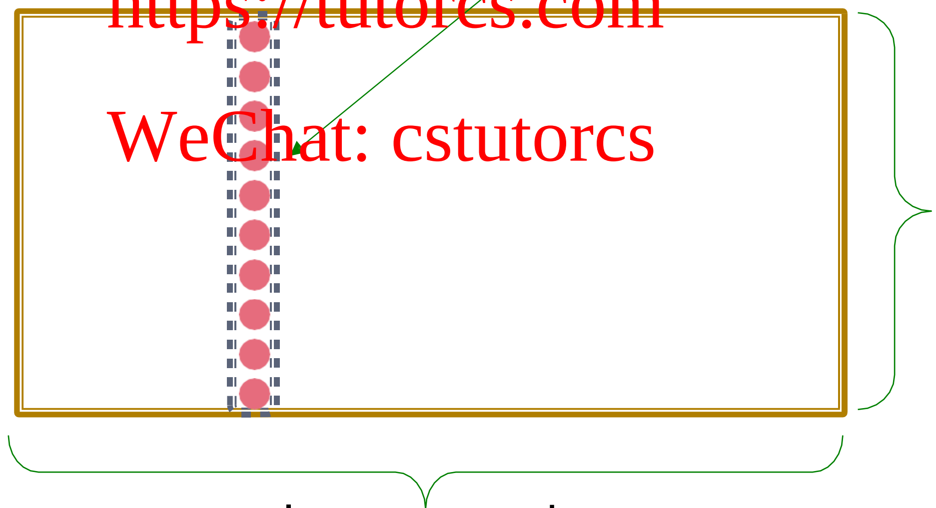
Recap: “Shallow” Encoding

Simplest encoding approach: encoder is just an embedding-lookup

embedding vector for a specific node

embedding matrix

$\mathbf{Z} =$



Dimension/size of embeddings

one column per node

Recap: "Shallow" Encoding

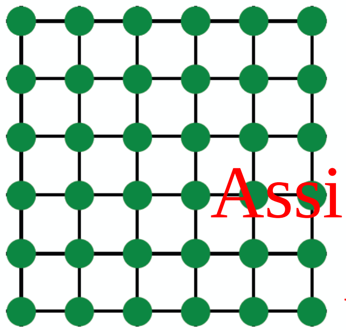
- **Limitations** of shallow embedding methods:
  - **$O(|V|)$ parameters are needed**:
    - No sharing of parameters between nodes
    - Every node has its own unique embedding
  - **Inherently "transductive"**:
    - Cannot generate embeddings for nodes that are not seen during training
  - **Do not incorporate node features**:
    - Many graphs have features that we can and should leverage

Today: Deep Graph Encoders

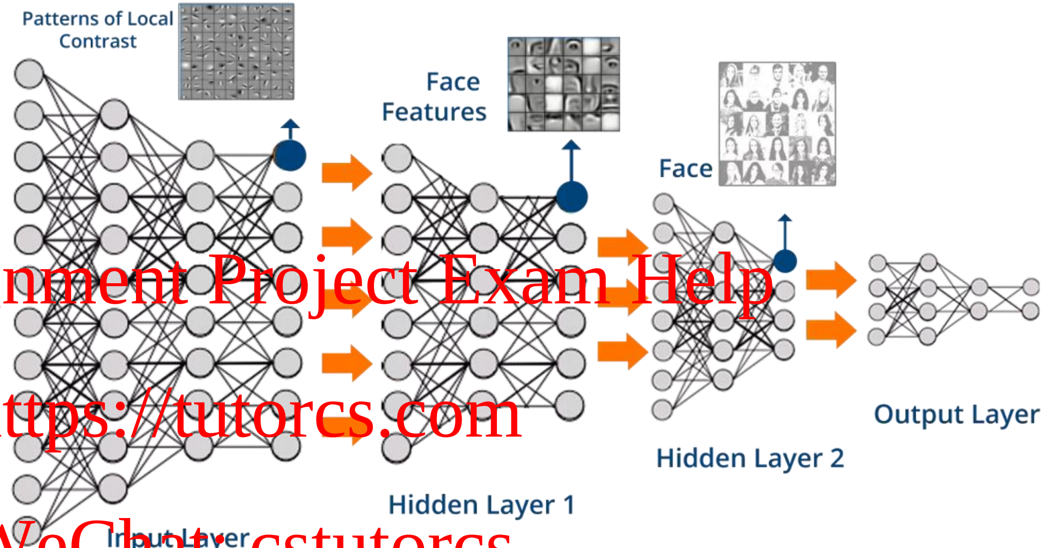
- **Today:** We will now discuss deep methods based on **graph neural networks (GNNs):**

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

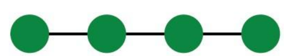
Modern ML Toolbox

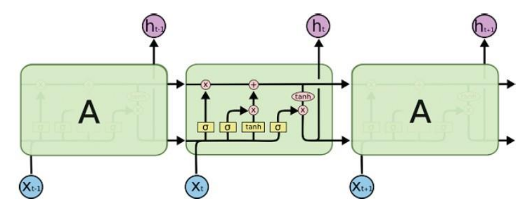


Images



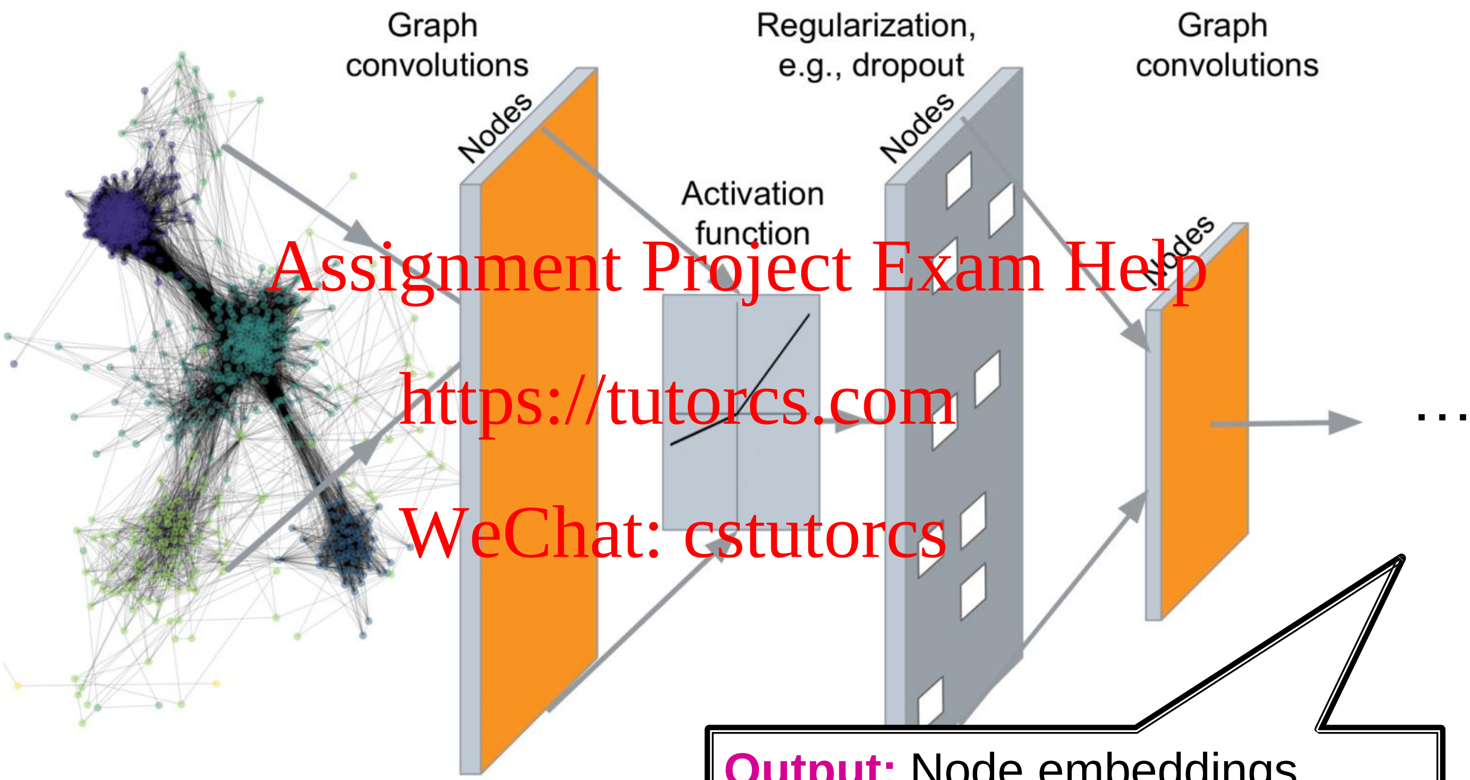
Text/Speech





Modern deep learning toolbox is designed for simple sequences & grids

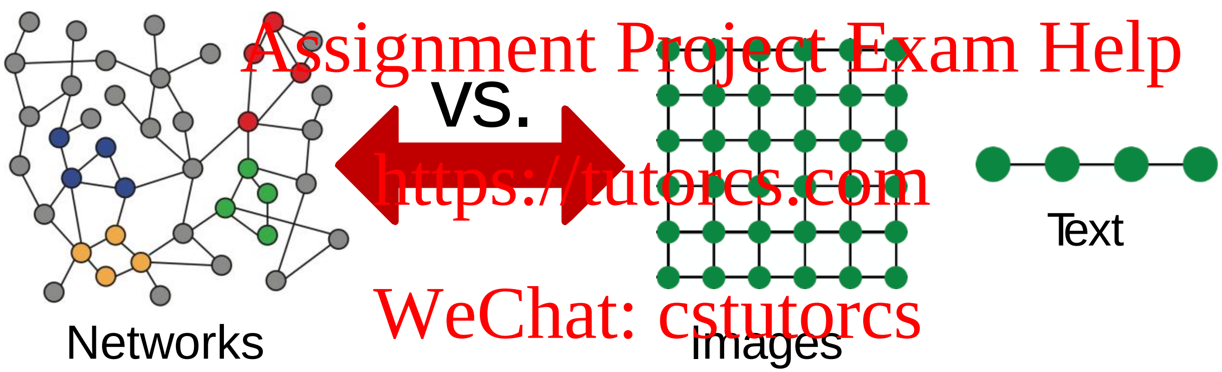
Deep Graph Encoders



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

But networks are far more complex!

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



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

Tasks on Networks

Tasks we will be able to solve:

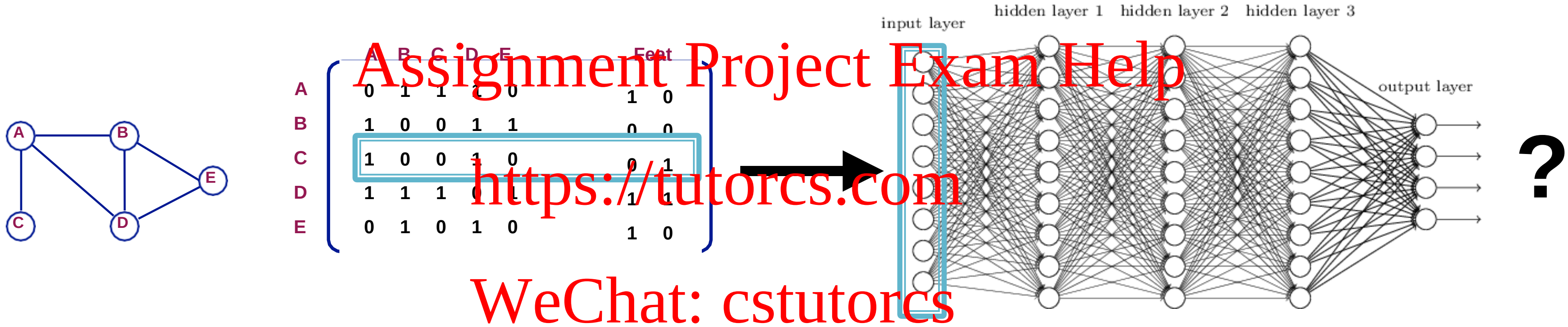
- Node classification
  - Predict a type of a given node
- Link prediction
  - Predict whether two nodes are linked
- Community detection
  - Identify densely linked clusters of nodes
- Network similarity
  - How similar are two (sub)networks

Setup

- **Assume we have a graph G:**
  - V is the **vertex set**
  - $\boldsymbol{A}$ is the **adjacency matrix** (assume binary)
  - $\boldsymbol{X} \in \mathbb{R}^{m\times|V|}$ 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
    - When there is no node feature in the graph dataset:
      - Indicator vectors (one-hot encoding of a node)
      - Vector of constant 1: [1, 1, …, 1]

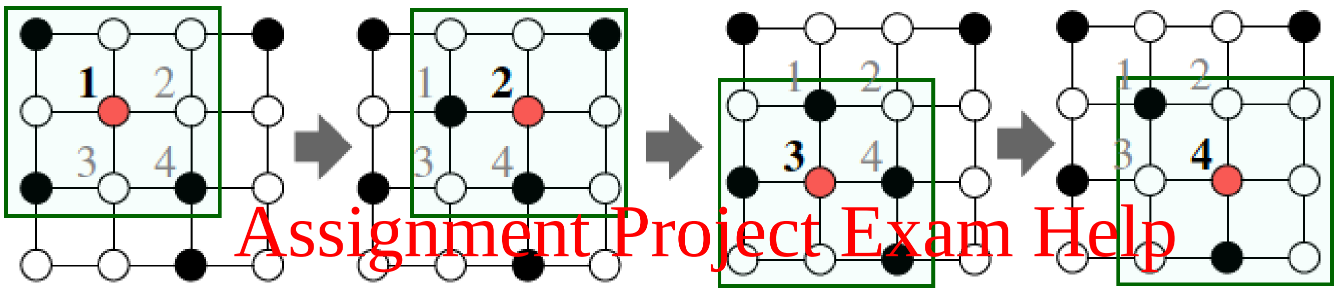
A Naïve Approach

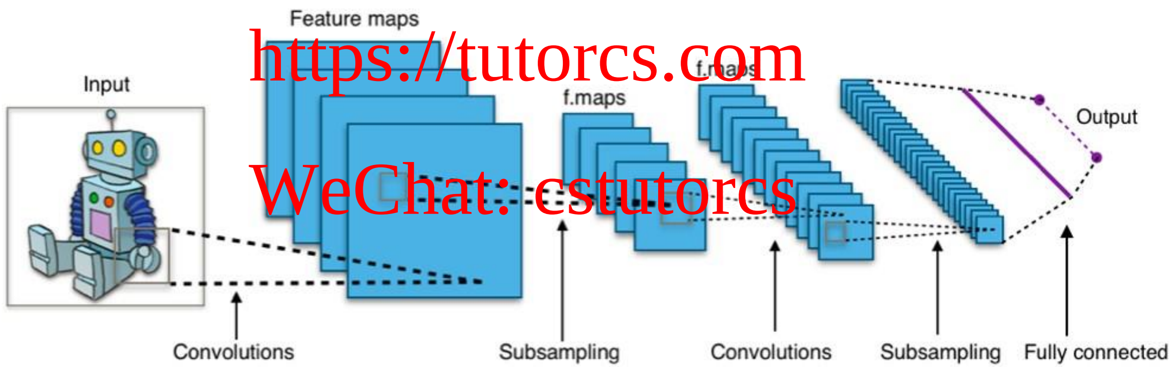
- Join adjacency matrix and features
- Feed them into a deep neural net:



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

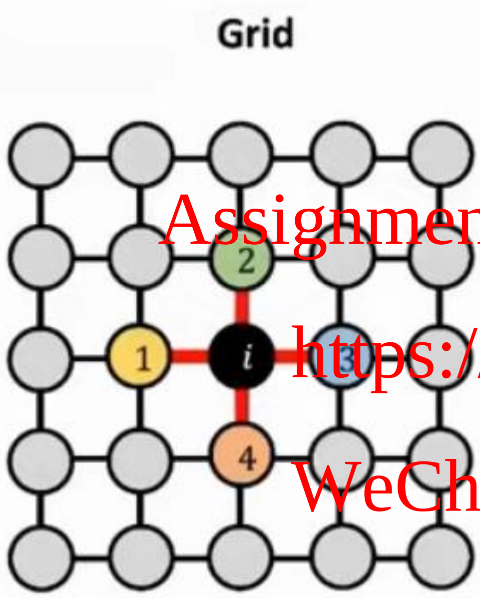
CNN on an image:





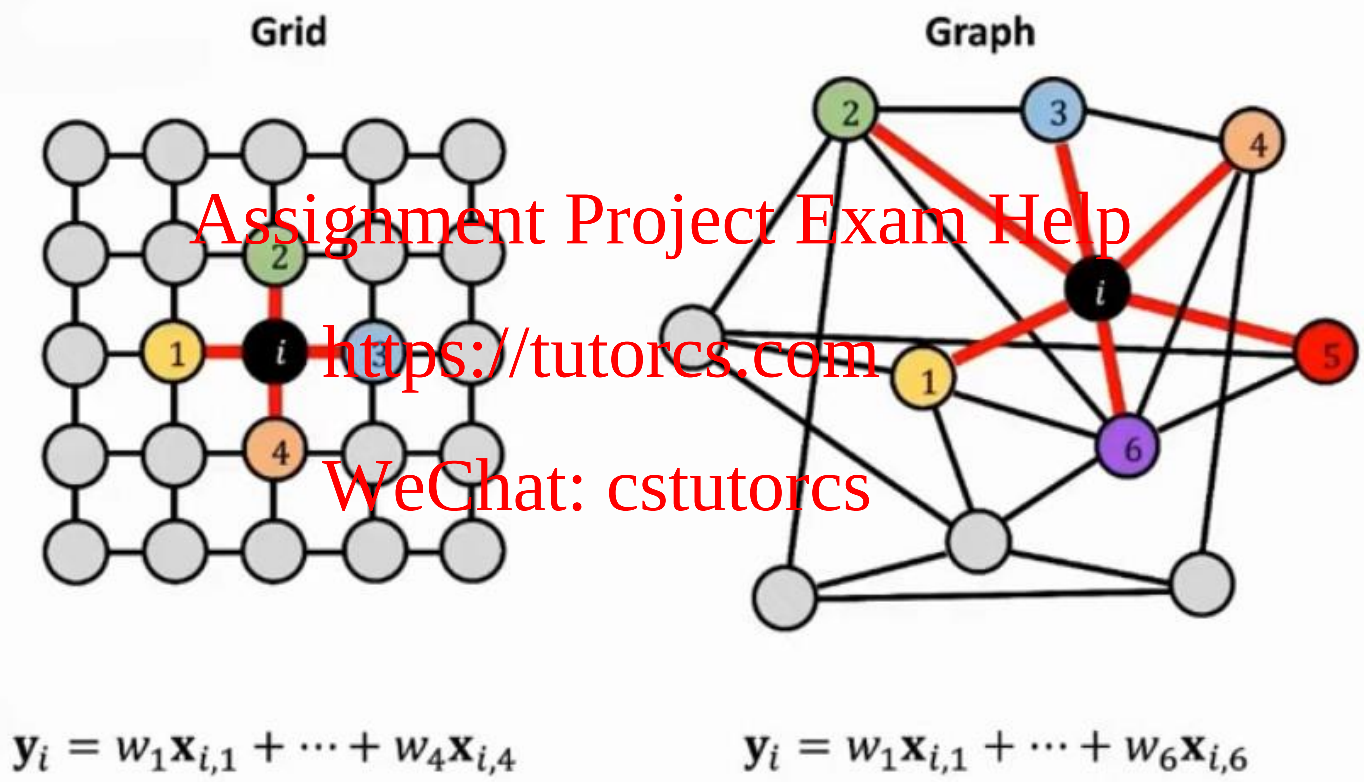
Goal is to generalize convolutions beyond simple lattices
Leverage node features/attributes (e.g., text, images)

What about Graphs?



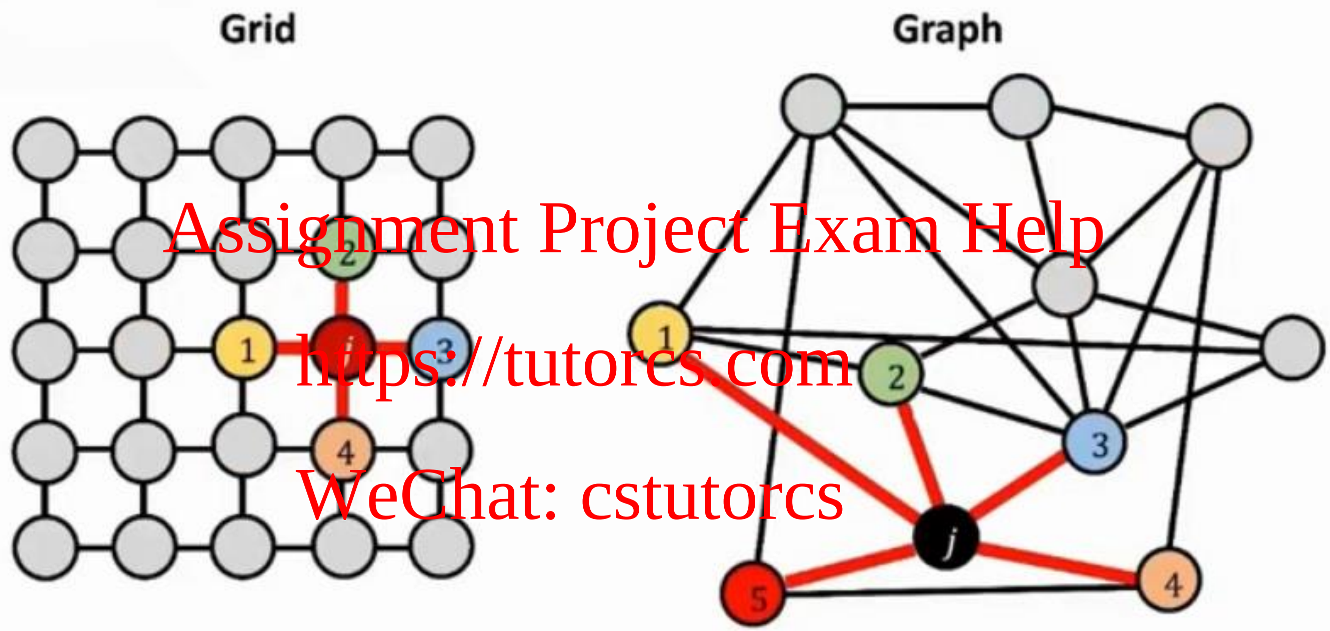
$$\mathbf{y}_i = w_1 \mathbf{x}_{i,1} + \cdots + w_4 \mathbf{x}_{i,4}$$

What about Graphs?



Talk on Deep learning on graphs: successes, challenges by Michael Bronstein

What about Graphs?



$\mathbf{y}_j = w_1\mathbf{x}_{j,1} + \cdots + w_4\mathbf{x}_{j,4}$

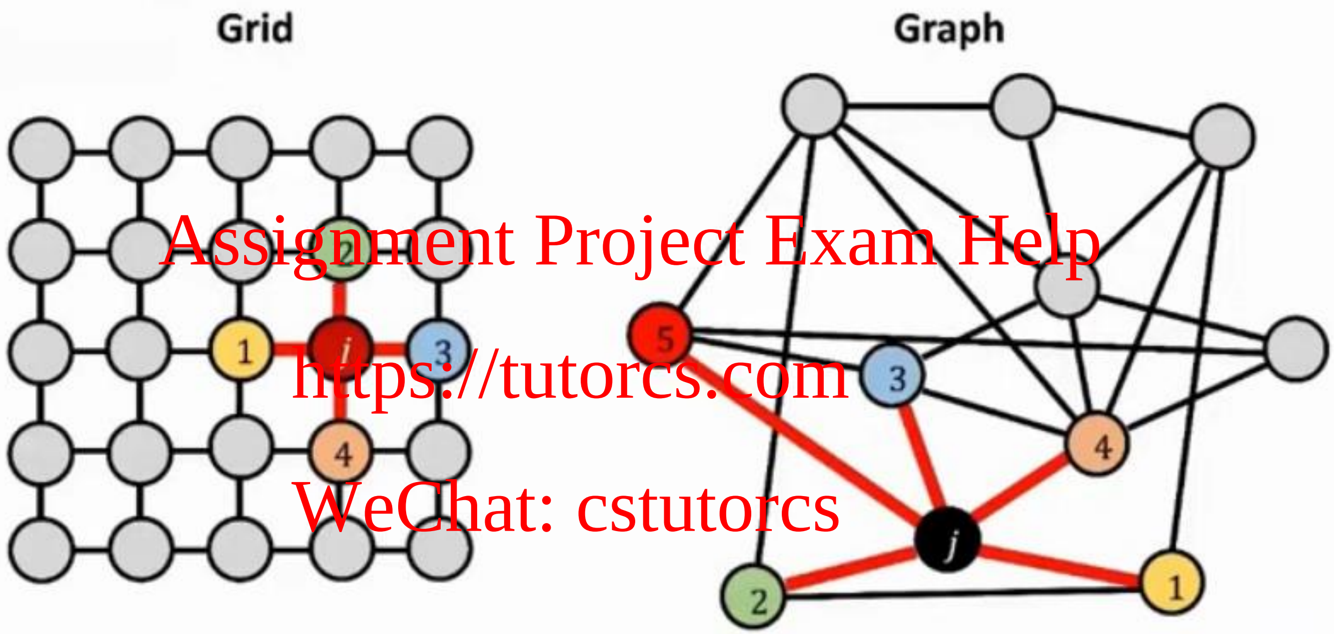
- Constant number of neighbors

$\mathbf{y}_j = w_1\mathbf{x}_{j,1} + \cdots + w_5\mathbf{x}_{j,5}$

- Different number of neighbors

Talk on Deep learning on graphs: successes, challenges by Michael Bronstein

What about Graphs?



$$\mathbf{y}_j = w_1 \mathbf{x}_{j,1} + \cdots + w_4 \mathbf{x}_{j,4}$$

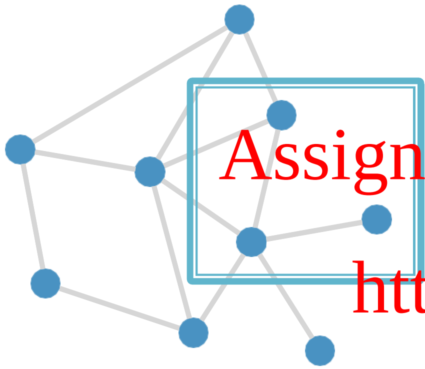
- Constant number of neighbors
- Fixed ordering of neighbors

$$\mathbf{y}_j = w_1 \mathbf{x}_{j,5} + \cdots + w_5 \mathbf{x}_{j,2}$$

- Different number of neighbors
- No ordering of neighbors

Talk on Deep learning on graphs: successes, challenges by Michael Bronstein

Graphs look like this

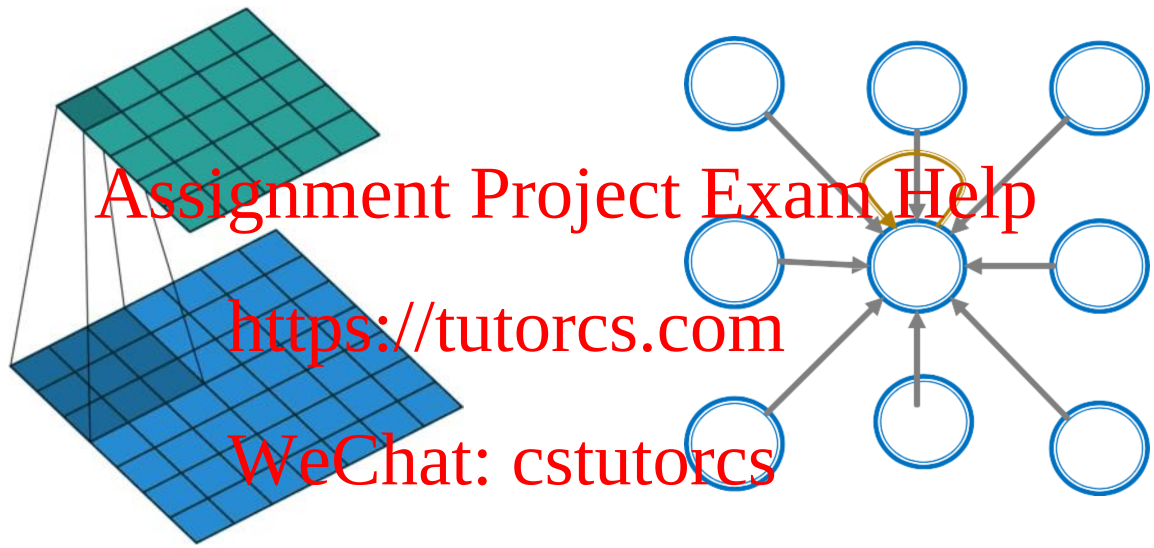


or this

1. No fixed notion of locality or sliding window on the graph
2. Graph is permutation invariant

Convolutional layer with 3x3 filter

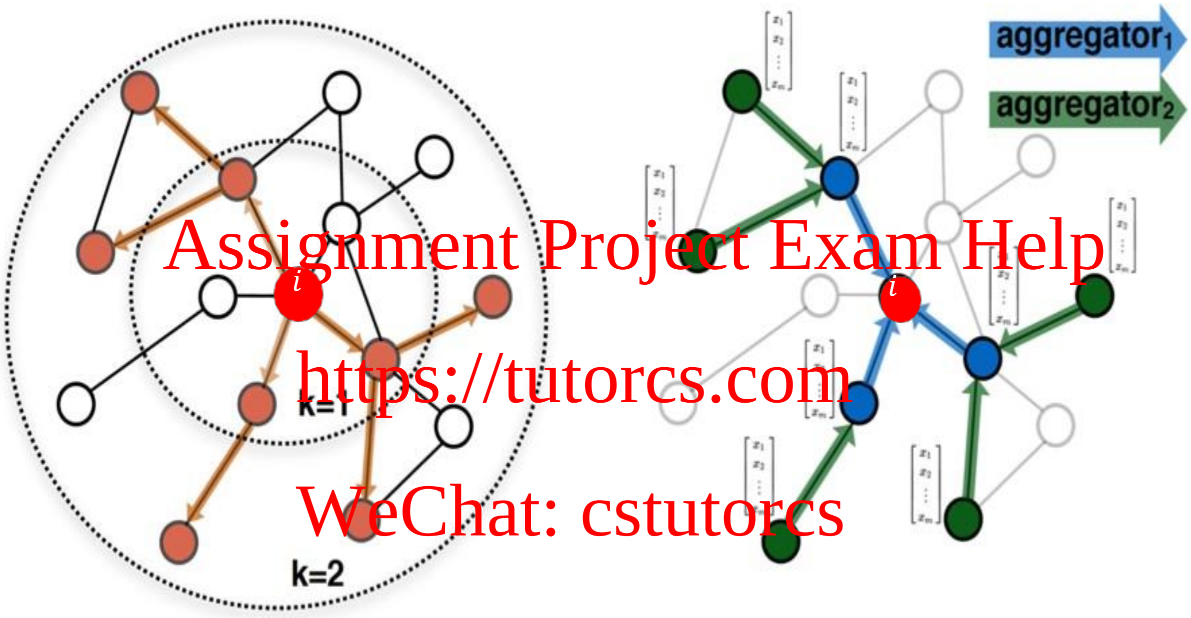
Image Graph



Idea: transform information at the neighbors and combine it:

- Transform "messages" h_i from neighbors: $W_i h_i$
- Add them up: $\sum_i W_i \, h_i$

A Computation Graph



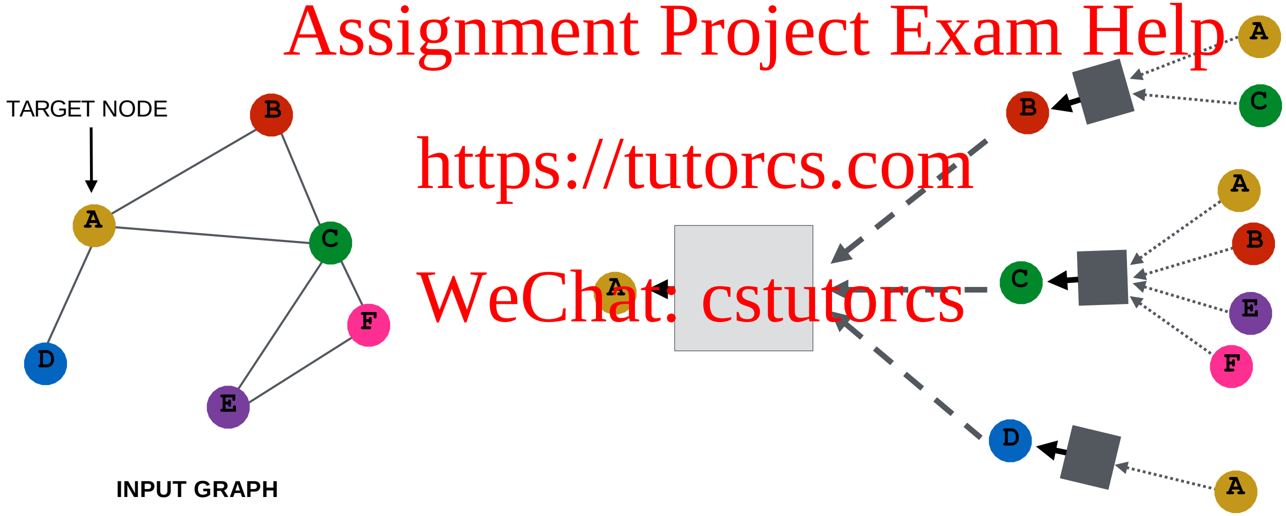
Determine node computation graph

Propagate and transform information

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

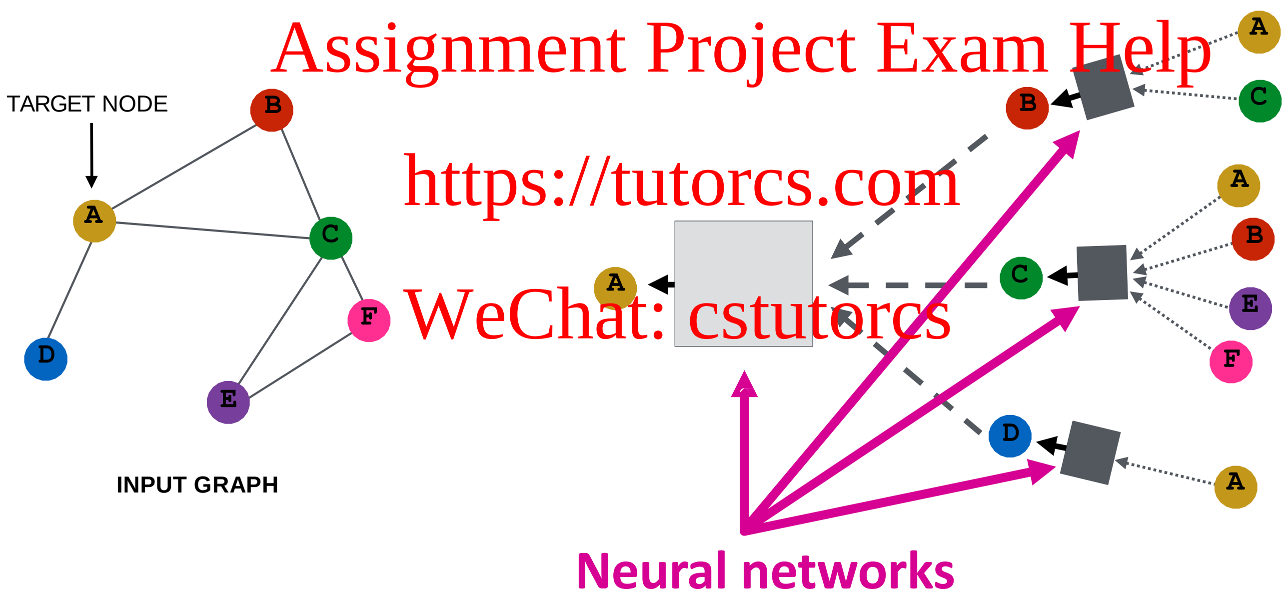
Aggregate Neighbors

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



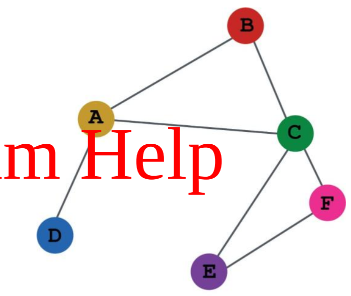
Aggregate Neighbors

Intuition: Nodes aggregate information from their neighbors using neural networks



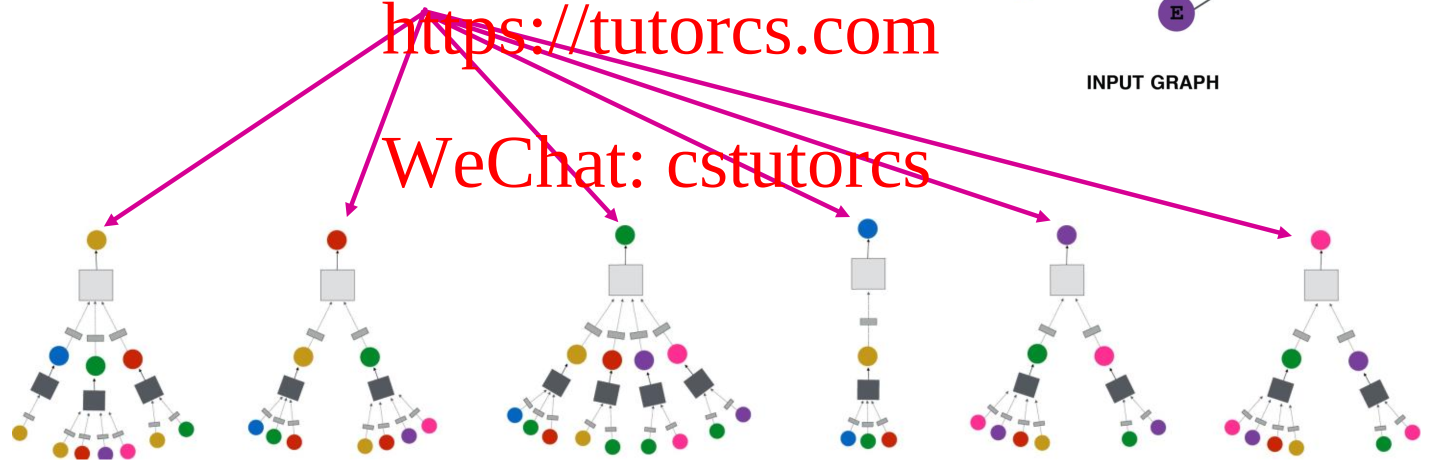
Aggregate Neighbors

Intuition: Network neighborhood defines a computation graph



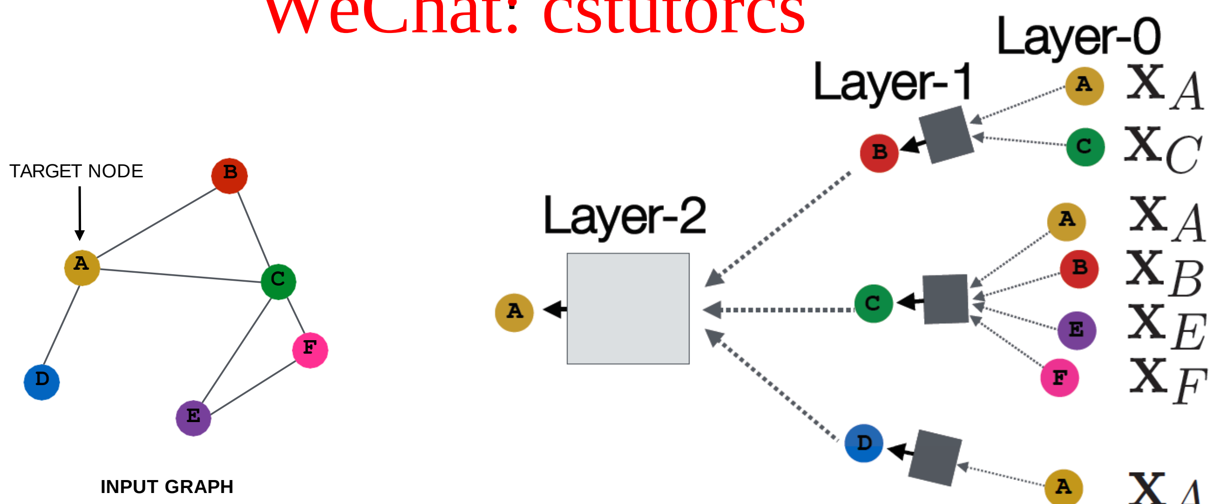
INPUT GRAPH

Every node defines a computation graph based on its neighborhood!



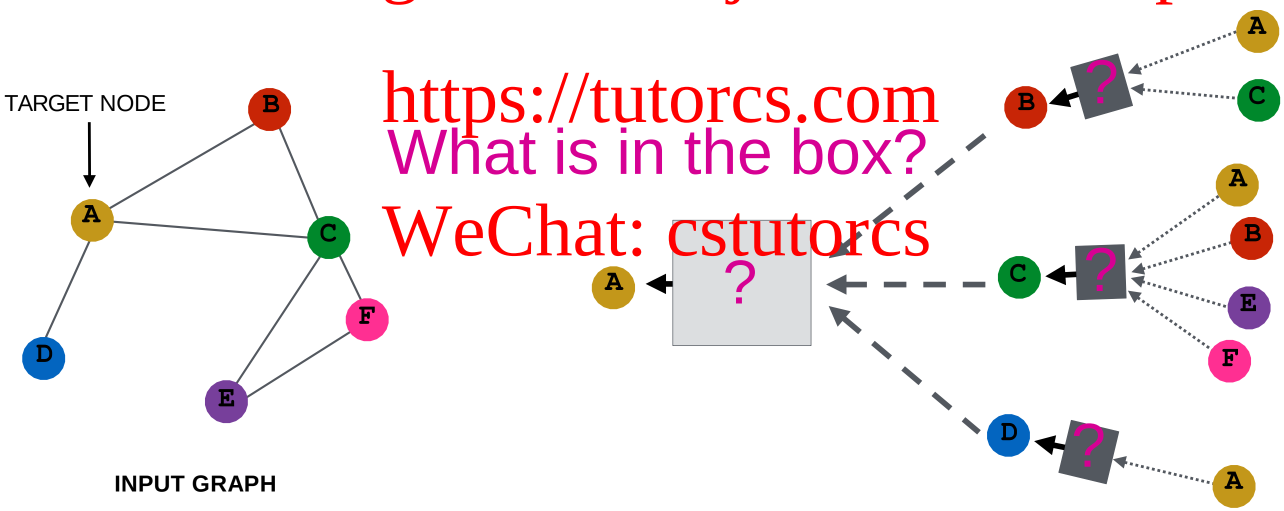
Deep: Many Layers

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



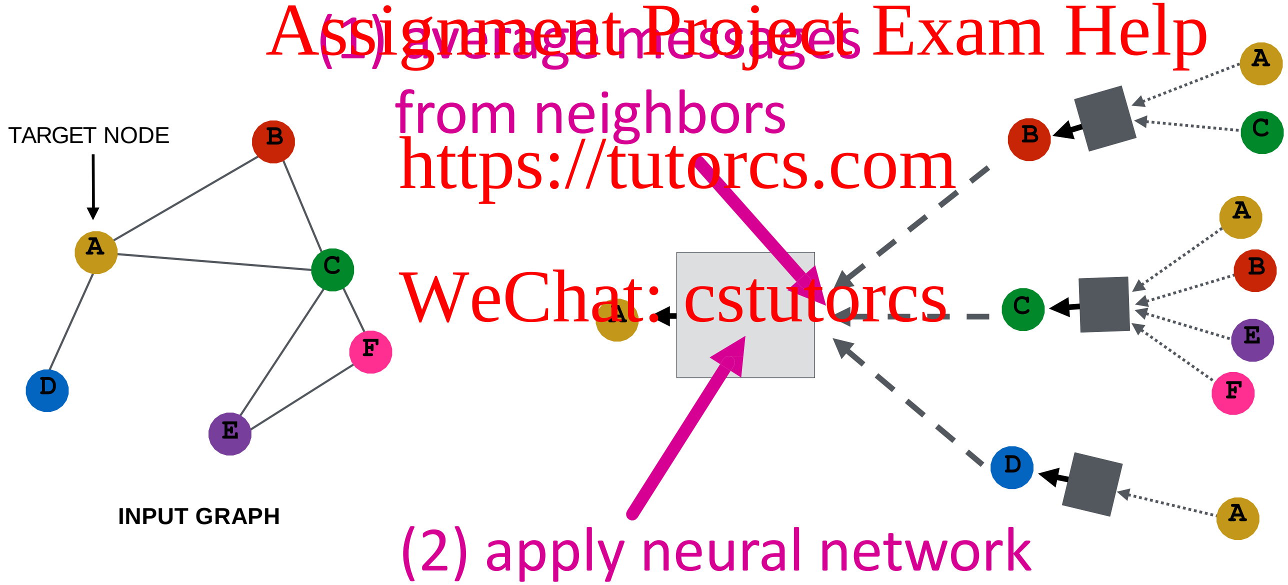
Neighborhood Aggregation

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



Neighborhood Aggregation

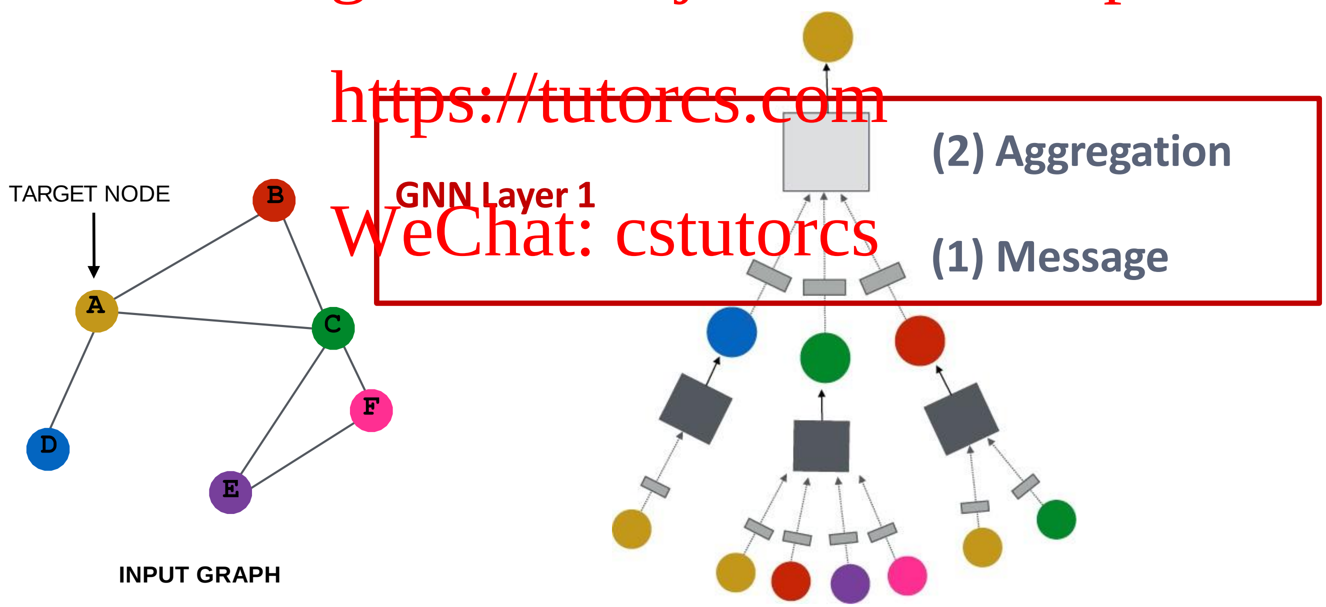
Basic approach: Average information from neighbors and apply a **neural network**



A GNN Layer

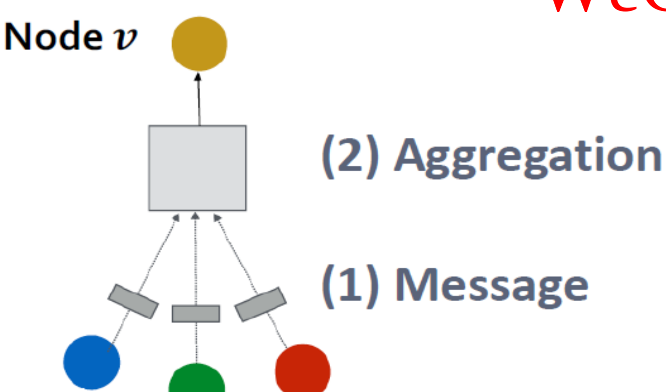
GNN Layer = Message + Aggregation

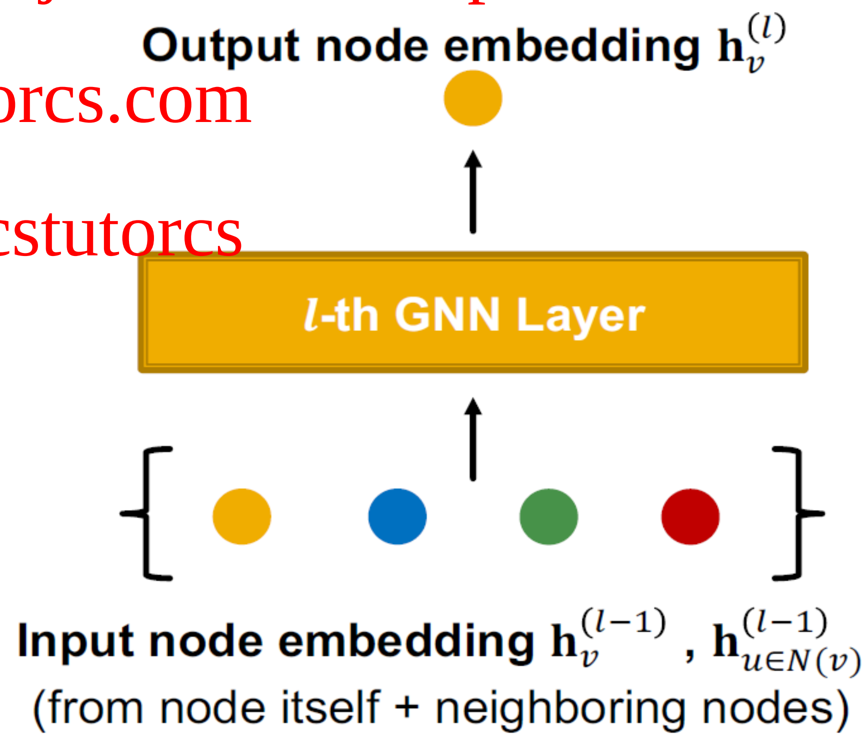
- **Different instantiations under this perspective**
- **GCN, GraphSAGE, GAT, …**



A Single GNN Layer

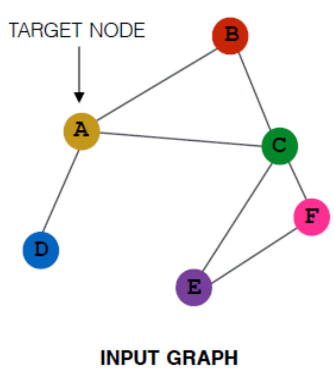
- **Idea of a GNN Layer:**
  - Compress a set of vectors into a single vector
  - **Two step process:**
    - **(1) Message**
    - **(2) Aggregation**

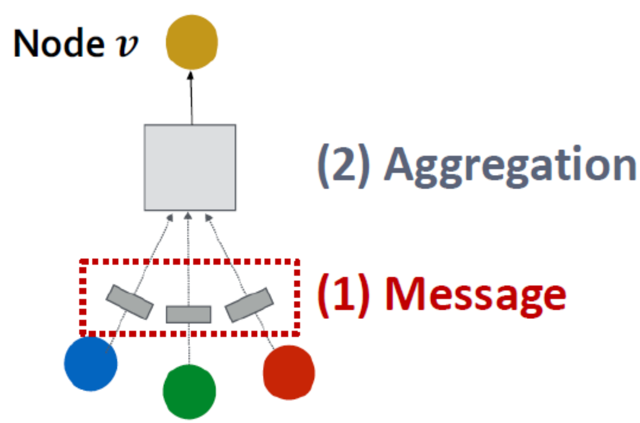




Message Computation

- **(1) Message computation**
  - **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 $\mathbf{W}^{(l)}$



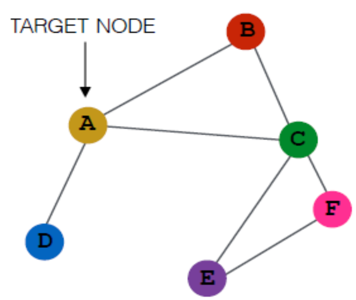


Message Aggregation

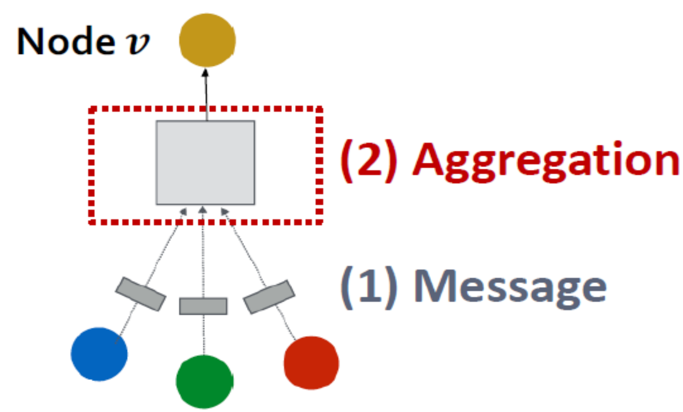
- **(2) Aggregation**
  - **Intuition:** Each node will aggregate the messages from node v's neighbors

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

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



INPUT GRAPH



Message Aggregation Issue

- **Issue:** Information from node v itself **could get lost**
  - Computation of $\mathbf{h}_v^{(l)}$ does not directly depend on $\mathbf{h}_v^{(l-1)}$
- **Solution:** Include $\mathbf{h}_v^{(l-1)}$ when computing $\mathbf{h}_v^{(l)}$

  - **(1) Message: compute message from node v itself**
    - Usually, a **different message computation** will be performed

$$\mathbf{m}_u^{(l)} = \mathbf{W}^{(l)}\mathbf{h}_u^{(l-1)} \qquad \mathbf{m}_v^{(l)} = \mathbf{B}^{(l)}\mathbf{h}_v^{(l-1)}$$

  - **(2) Aggregation:** After aggregating from neighbors, we can **aggregate the message from node v itself**
    - Via **concatenation** or **summation**

Then aggregate from node itself

$$\mathbf{h}_v^{(l)} = \mathrm{CONCAT}\left(\underbrace{\mathrm{AGG}\left(\left\{\mathbf{m}_u^{(l)}, u \in N(v)\right\}\right)}_{\text{First aggregate from neighbors}}, \boxed{\mathbf{m}_v^{(l)}}\right)$$

A Single GNN Layer

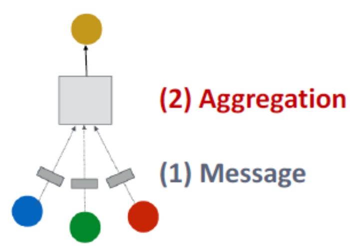
- **Putting things together:**
  - **(1) Message**: each node computes a message

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

  - **(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)**: Adds expressiveness
    - Often written as $\sigma(\cdot)$: $\mathrm{ReLU}(\cdot)$, $\mathrm{Sigmoid}(\cdot)$, ...
    - Can be added to **message or aggregation**



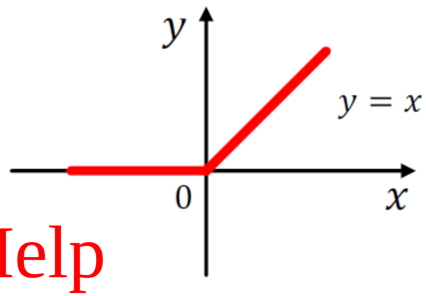
Activation (Non-linearity)

Apply activation to i-th dimension of embedding x

- **Rectified linear unit (ReLU)**

  $$\mathrm{ReLU}(\mathbf{x}_i) = \max(\mathbf{x}_i, 0)$$

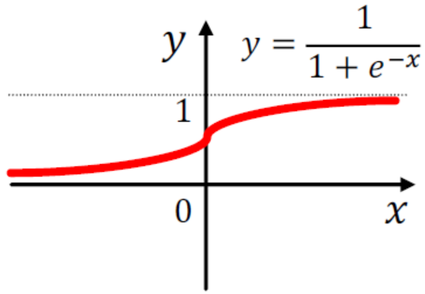
  - Most commonly used



- **Sigmoid**

  $$\sigma(\mathbf{x}_i) = \frac{1}{1 + e^{-\mathbf{x}_i}}$$

  - Used only when you want to restrict the range of your embeddings

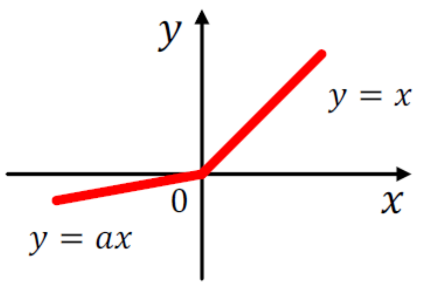


- **Parametric ReLU**

  $$\mathrm{PReLU}(\mathbf{x}_i) = \max(\mathbf{x}_i, 0) + a_i \min(\mathbf{x}_i, 0)$$

  a_i is a trainable parameter

  - Empirically performs better than ReLU



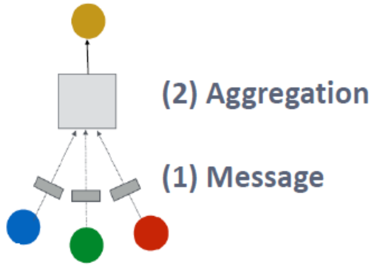
Classical GNN Layers: GCN

- **(1) Graph Convolutional Networks (GCN)**

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

- **How to write this as Message + Aggregation?**

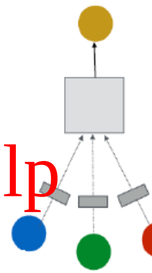
$$\mathbf{h}_v^{(l)} = \sigma\left(\underbrace{\boxed{\sum_{u \in N(v)}}}_{\text{Aggregation}} \overbrace{\boxed{\mathbf{W}^{(l)} \frac{\mathbf{h}_u^{(l-1)}}{|N(v)|}}}^{\text{Message}}\right)$$



Classical GNN Layers: GCN

- **(1) Graph Convolutional Networks (GCN)**

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



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

The Maths: Deep Encoder

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

Initial 0-th layer embeddings are equal to node features

$$\boxed{\mathrm{h}_v^0 = \mathrm{x}_v}$$

Embedding of v at layer l

$$\mathrm{h}_v^{(l+1)} = \boxed{\sigma}\left(\mathrm{W}_l \boxed{\sum_{u \in \mathrm{N}(v)} \frac{\mathrm{h}_u^{(l)}}{|\mathrm{N}(v)|}} + \mathrm{B}_l \boxed{\mathrm{h}_v^{(l)}}\right), \forall l \in \{0, \dots, \boxed{L} - 1\}$$

$$\boxed{\mathrm{z}_v} = \mathrm{h}_v^{(L)}$$

Embedding after L layers of neighborhood aggregation

Non-linearity (e.g., ReLU)

Average of neighbor's previous layer embeddings

Total number of layers

Model Parameters

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

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

$$h_v^{(l+1)} = \sigma\left(\boxed{W_l} \sum_{u \in N(v)} \frac{h_u^{(l)}}{|N(v)|} + \boxed{B_l} h_v^{(l)}\right), \forall l \in \{0, \dots, L-1\}$$

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

Final node embedding

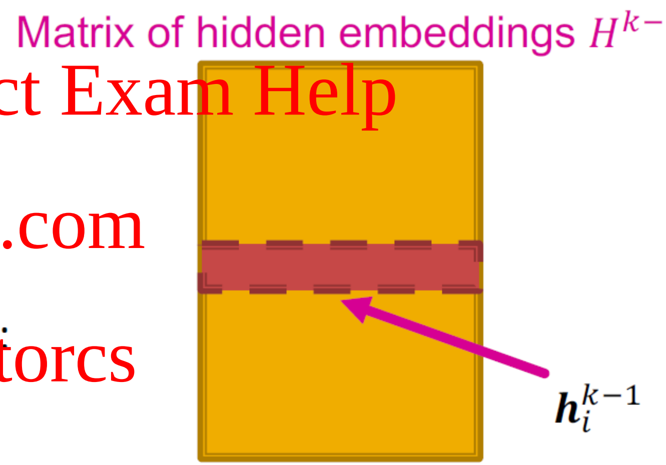
We can feed these **embeddings into any loss function** and run SGD to **train the weight parameters**

- h_v^l :the hidden representation of node v at layer l
- W_k :weight matrix for neighborhood aggregation
- B_k : weight matrix for transforming hidden vector of self

Matrix Formulation

- **Many aggregations can be performed efficiently by (sparse) matrix operations**
- Let $H^{(l)} = [h_1^{(l)} \dots h_{|V|}^{(l)}]^{\mathrm{T}}$
- Then: $\sum_{u \in N_v} h_u^{(l)} = A_{v,:} H^{(l)}$
- Let D be diagonal matrix where $D_{v,v} = \mathrm{Deg}(v) = |N(v)|$
  - The inverse of D: D^{-1} is also diagonal: $D_{v,v}^{-1} = 1/|N(v)|$
- **Therefore,**

$$\boxed{\sum_{u \in N(v)} \frac{h_u^{(l-1)}}{|N(v)|}} \Rightarrow H^{(l+1)} = D^{-1} A H^{(l)}$$

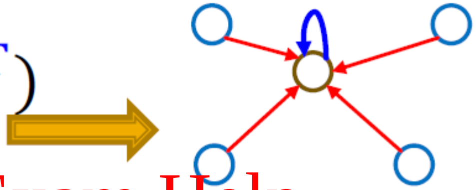


Matrix Formulation

- Re-writing update function in matrix form:

$$H^{(l+1)} = \sigma(\tilde{A}H^{(l)}W_l^{\mathrm{T}} + H^{(l)}B_l^{\mathrm{T}})$$

where $\tilde{A} = D^{-1}A$



$$H^{(l)} = [h_1^{(l)} \dots h_{|V|}^{(l)}]^T$$

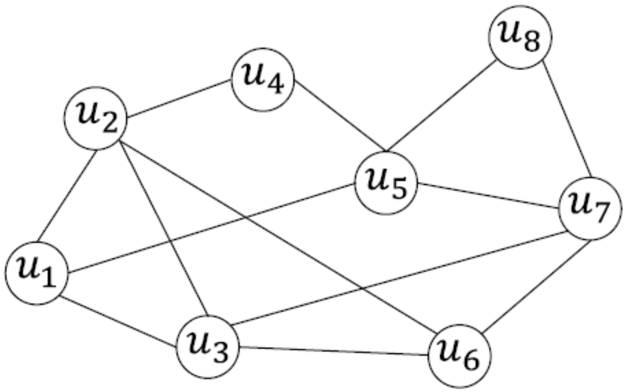
  - Red: neighborhood aggregation
  - Blue: self transformation

- In practice, this implies that efficient sparse matrix multiplication can be used ($\tilde{A}$ is sparse)

- **Note**: not all GNNs can be expressed in matrix form, when aggregation function is complex

Example

$$H^{(l+1)} = \sigma(\tilde{A}H^{(l)}W_l^{\mathrm{T}} + H^{(l)}B_l^{\mathrm{T}})$$

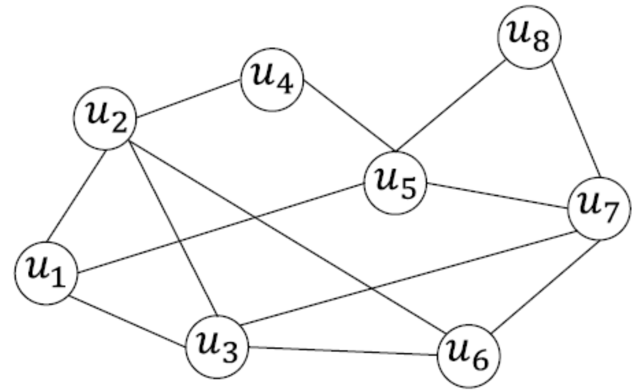


Compute the output of the first graph convolutional layer based on the above formula

$$H_0 = \begin{bmatrix} 0.20 & 0.60 & 0.30 & -0.40 \\ 0.40 & 0.30 & -0.20 & -0.60 \\ 0.20 & -0.60 & 0.50 & -0.30 \\ -0.40 & 0.20 & 0.20 & -0.40 \\ 0.70 & -0.90 & 0.10 & -0.50 \\ 0.30 & 0.50 & -0.30 & -0.70 \\ 0.90 & -0.60 & 0.20 & -0.80 \\ -0.10 & 0.70 & 0.10 & -0.90 \end{bmatrix} \quad W^0 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix} \quad B^0 = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix}$$

Example

$$H^{(l+1)} = \sigma(\tilde{A} H^{(l)} W_l^{\mathrm{T}} + H^{(l)} B_l^{\mathrm{T}})$$



Adjacent matrix A:

```
[[0 1 1 0 1 0 0 0]
 [1 0 1 1 0 1 0 0]
 [1 1 0 0 0 1 1 0]
 [0 1 0 0 1 0 0 0]
 [1 0 0 1 0 0 1 1]
 [0 1 1 0 0 0 1 0]
 [0 0 1 0 1 1 0 1]
 [0 0 0 0 1 0 1 0]]
```

The matrix D^{-1}:

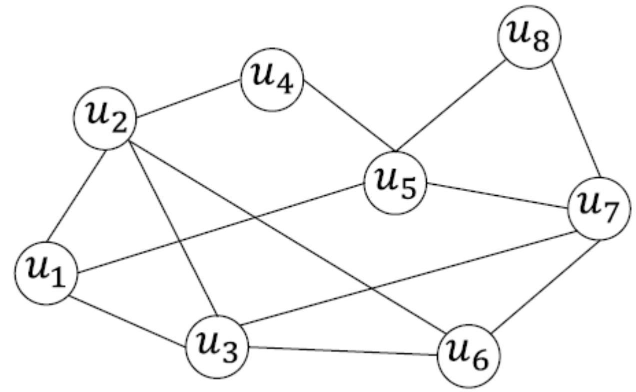
```
[[0.33333334 0.         0.         0.         0.         0.         0.         0.        ]
 [0.         0.25       0.         0.         0.         0.         0.         0.        ]
 [0.         0.         0.25       0.         0.         0.         0.         0.        ]
 [0.         0.         0.         0.5        0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.25       0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.33333334 0.         0.        ]
 [0.         0.         0.         0.         0.         0.         0.25       0.        ]
 [0.         0.         0.         0.         0.         0.         0.         0.5       ]]
```

The matrix $D^{-1}A$:

```
[[0.         0.33333334 0.33333334 0.         0.33333334 0.         0.         0.        ]
 [0.25       0.         0.25       0.25       0.         0.25       0.         0.        ]
 [0.25       0.25       0.         0.         0.         0.25       0.25       0.        ]
 [0.         0.5        0.         0.         0.5        0.         0.         0.        ]
 [0.25       0.         0.         0.25       0.         0.         0.25       0.25      ]
 [0.         0.33333334 0.33333334 0.         0.         0.         0.33333334 0.        ]
 [0.         0.         0.25       0.         0.25       0.25       0.         0.25      ]
 [0.         0.         0.         0.         0.5        0.         0.5        0.        ]]
```

Example

$$H^{(l+1)} = \sigma(\tilde{A}H^{(l)}W_l^{\mathrm{T}} + H^{(l)}B_l^{\mathrm{T}})$$



Matrix H^0 :

Matrix $D^{-1}A$:

$$\begin{bmatrix} 0.20 & 0.60 & 0.30 & -0.40 \\ 0.40 & 0.30 & -0.20 & -0.60 \\ 0.20 & -0.60 & 0.50 & -0.30 \\ -0.40 & 0.20 & 0.20 & -0.40 \\ 0.70 & -0.90 & 0.10 & -0.50 \\ 0.30 & 0.50 & -0.30 & -0.70 \\ 0.90 & -0.60 & 0.20 & -0.80 \\ -0.10 & 0.70 & 0.10 & -0.90 \end{bmatrix}$$

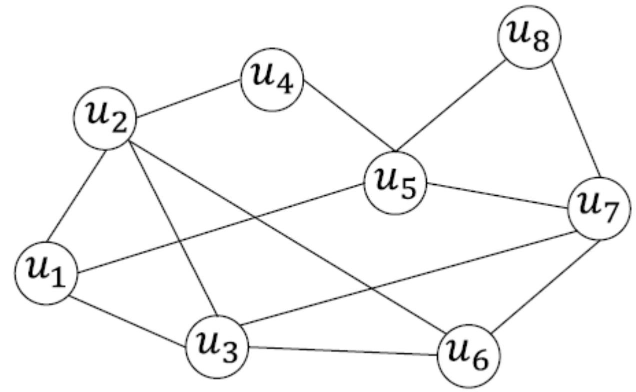
```
[[0.         0.33333334 0.33333334 0.         0.33333334 0.         0.         0.        ]
 [0.25       0.         0.25       0.25       0.         0.25       0.         0.        ]
 [0.25       0.25       0.         0.         0.         0.25       0.25       0.        ]
 [0.         0.5        0.         0.         0.5        0.         0.         0.        ]
 [0.25       0.         0.         0.25       0.         0.         0.25       0.25      ]
 [0.         0.33333334 0.33333334 0.         0.         0.         0.33333334 0.        ]
 [0.         0.         0.25       0.         0.25       0.25       0.         0.25      ]
 [0.         0.         0.         0.         0.5        0.         0.5        0.        ]]
```

Matrix $D^{-1}AH$:

```
[[ 0.43333335 -0.40000001  0.13333334 -0.46666668]
 [ 0.075       0.175       0.175      -0.1       ]
 [ 0.45        0.2         0.         -0.275     ]
 [ 0.55       -0.3        -0.05       -0.55      ]
 [ 0.15        0.225       0.2        -0.625     ]
 [ 0.50000001 -0.30000001  0.16666667 -0.56666668]
 [ 0.275      -0.075       0.1        -0.25      ]
 [ 0.8        -0.75        0.15       -0.65      ]]
```

Example

$$H^{(l+1)} = \sigma(\tilde{A}H^{(l)}W_l^{\mathrm{T}} + H^{(l)}B_l^{\mathrm{T}})$$



Matrix $D^{-1}AH$:

Matrix W^0 :

```
[[ 0.43333335 -0.40000001  0.13333334 -0.46666668]
 [ 0.075       0.175       0.175      -0.1       ]
 [ 0.45        0.2         0.         -0.275     ]
 [ 0.55       -0.3        -0.05       -0.55      ]
 [ 0.15        0.225       0.2        -0.625     ]
 [ 0.50000001 -0.30000001  0.16666667 -0.56666668]
 [ 0.275      -0.075       0.1        -0.25      ]
 [ 0.8        -0.75        0.15       -0.65      ]]
```

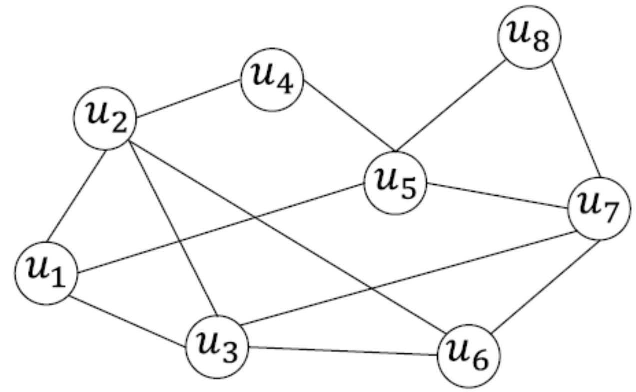

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

Matrix $D^{-1}AHW^T$:

```
[[ 0.43333335  0.03333333  0.16666667 -0.30000001]
 [ 0.075       0.25        0.425       0.325     ]
 [ 0.45        0.65        0.65        0.375     ]
 [ 0.55        0.25        0.2        -0.35      ]
 [ 0.15        0.375       0.575      -0.05      ]
 [ 0.50000001  0.20000001  0.36666668 -0.20000001]
 [ 0.275       0.2         0.3         0.05      ]
 [ 0.8         0.05        0.2        -0.45      ]]
```

Example

$$H^{(l+1)} = \sigma(\tilde{A}H^{(l)}W_l^{\mathrm{T}} + H^{(l)}B_l^{\mathrm{T}})$$



Matrix B^0 :

$$\begin{bmatrix} 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix}$$

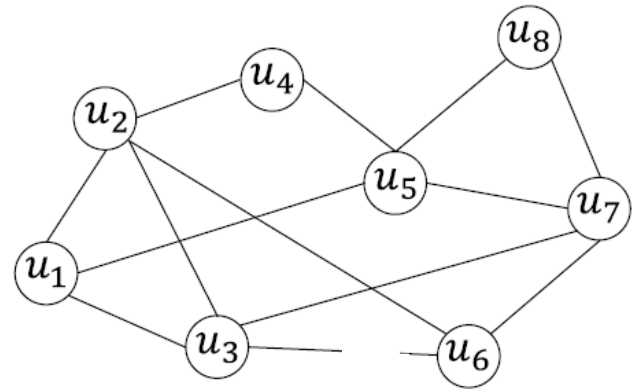
Matrix H^0 :

$$\begin{bmatrix} 0.20 & 0.60 & 0.30 & -0.40 \\ 0.40 & 0.30 & -0.20 & -0.60 \\ 0.20 & -0.60 & 0.50 & -0.30 \\ -0.40 & 0.20 & 0.20 & -0.40 \\ 0.70 & -0.90 & 0.10 & -0.50 \\ 0.30 & 0.50 & -0.30 & -0.70 \\ 0.90 & -0.60 & 0.20 & -0.80 \\ -0.10 & 0.70 & 0.10 & -0.90 \end{bmatrix}$$

Matrix HB^T :

```
[[-0.2  0.5 -0.1  0.5]
 [-0.2  0.2 -0.8  0.2]
 [-0.1  0.7  0.2  0.7]
 [-0.8 -0.2 -0.2 -0.2]
 [ 0.2  0.8 -0.4  0.8]
 [ 1.   0.   0.4  0. ]
 [ 0.1  1.1 -0.6  1.1]
 [-1.   0.  -0.8  0. ]]
```

Example



$$H^{(l+1)} = \sigma\left(\tilde{A}H^{(l)}W_l^{\mathrm{T}} + H^{(l)}B_l^{\mathrm{T}}\right)$$

Matrix $D^{-1}AHW^T$:

```
[[ 0.43333335  0.03333333  0.16666667 -0.30000001]
 [ 0.075       0.25        0.425       0.325     ]
 [ 0.45        0.65        0.65        0.375     ]
 [ 0.55        0.25        0.2        -0.35      ]
 [ 0.15        0.375       0.575      -0.05      ]
 [ 0.50000001  0.20000001  0.36666668 -0.20000001]
 [ 0.275       0.2         0.3         0.05      ]
 [ 0.8         0.05        0.2        -0.45      ]]
```

Matrix HB^T :

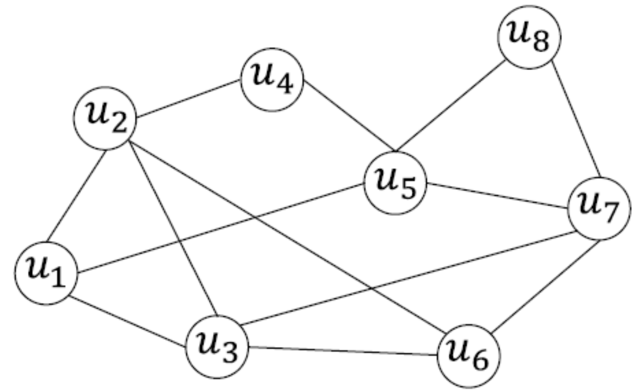
```
[[-0.2  0.5 -0.1  0.5]
 [-0.2  0.2 -0.8  0.2]
 [-0.1  0.7  0.2  0.7]
 [-0.8 -0.2 -0.2 -0.2]
 [ 0.2  0.8 -0.4  0.8]
 [ 1.   0.   0.4  0. ]
 [ 0.1  1.1 -0.6  1.1]
 [-1.   0.  -0.8  0. ]]
```


Matrix $D^{-1}AHW^T + HB^T$:

```
[[ 0.23333335  0.53333333  0.06666667  0.19999999]
 [-0.125       0.45       -0.375       0.525     ]
 [ 0.35        1.35        0.85        1.075     ]
 [-0.25        0.05        0.         -0.55      ]
 [ 0.35        1.175       0.175       0.75      ]
 [ 1.50000001  0.20000001  0.76666668 -0.20000001]
 [ 0.375       1.3        -0.3         1.15      ]
 [-0.2        0.05       -0.6        -0.45      ]]
```

Example

$$H^{(l+1)} = \sigma(\tilde{A}H^{(l)}W_l^{\mathrm{T}} + H^{(l)}B_l^{\mathrm{T}})$$



Matrix $D^{-1}AHW^T + HB^T$:


```
[[ 0.23333335  0.53333333  0.06666667  0.19999999]
 [-0.125       0.45       -0.375       0.525     ]
 [ 0.35        1.35        0.85        1.075     ]
 [-0.25        0.05        0.         -0.45      ]
 [ 0.35        1.175       0.175       0.75      ]
 [ 1.50000001  0.20000001  0.76666668 -0.20000001]
 [ 0.375       1.3        -0.3         1.15      ]
 [-0.2         0.05       -0.6        -0.45      ]]
```


Matrix $\sigma(D^{-1}AHW^T + HB^T)$:

```
[[0.23333335 0.53333333 0.06666667 0.19999999]
 [0.         0.45       0.         0.525     ]
 [0.35       1.35       0.85       1.075     ]
 [0.         0.05       0.         0.        ]
 [0.35       1.175      0.175      0.75      ]
 [1.50000001 0.20000001 0.76666668 0.        ]
 [0.375      1.3        0.         1.15      ]
 [0.         0.05       0.         0.        ]]
```

Train a GNN

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

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

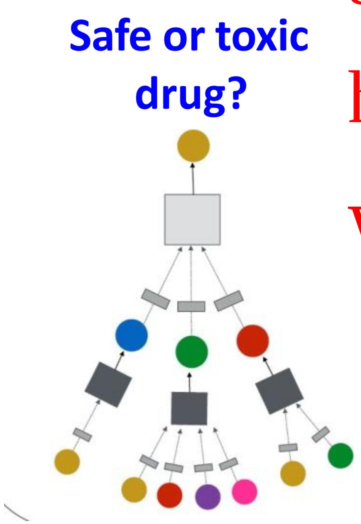
  - $\boldsymbol{y}$: node label

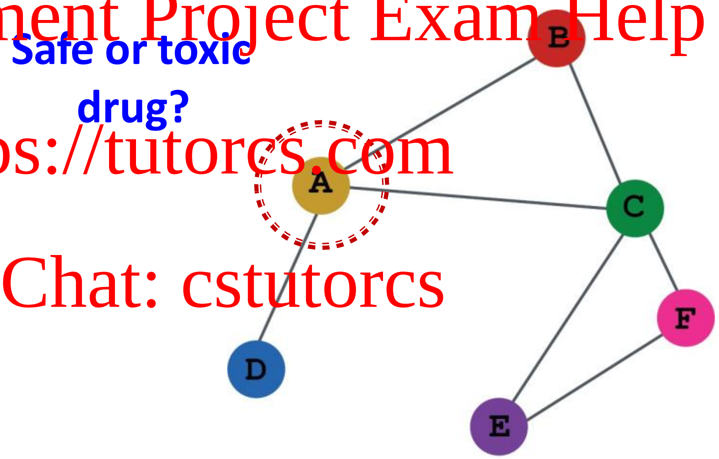
  - $\mathcal{L}$ could be L2 if $\boldsymbol{y}$ is real number, or cross entropy if $\boldsymbol{y}$ is categorical

- **Unsupervised setting:**
  - No node label available
  - **Use the graph structure as the supervision!**

Supervised Training

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





E.g., a drug-drug interaction network

Supervised Training

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

- Use cross entropy loss

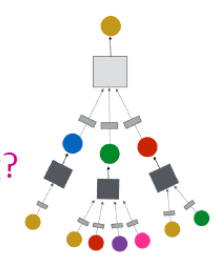
$$\mathcal{L} = -\sum_{v \in V} y_v \log(\sigma(\mathrm{z}_v^{\mathrm{T}}\theta)) + (1 - y_v)\log(1 - \sigma(\mathrm{z}_v^{\mathrm{T}}\theta))$$

Encoder output: node embedding

Classification weights

Node class label

Safe or toxic drug?



Unsupervised Training

- **"Similar" nodes have similar embeddings**

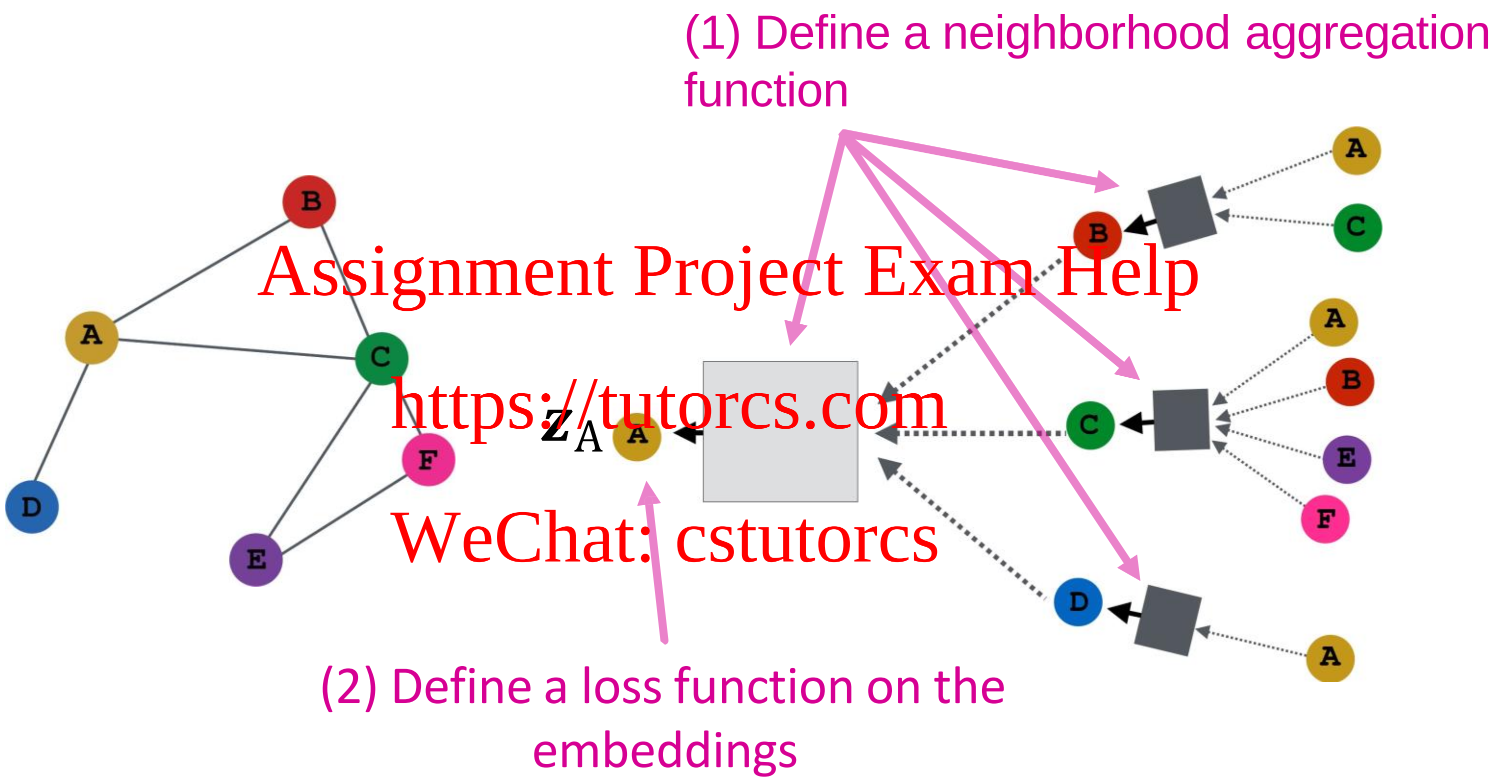
$$\mathcal{L} = \sum_{z_u, z_v} \mathrm{CE}(y_{u,v}, \mathrm{DEC}(z_u, z_v))$$

  - Where $y_{u,v}\ 1 =$ when node u and v are **similar**

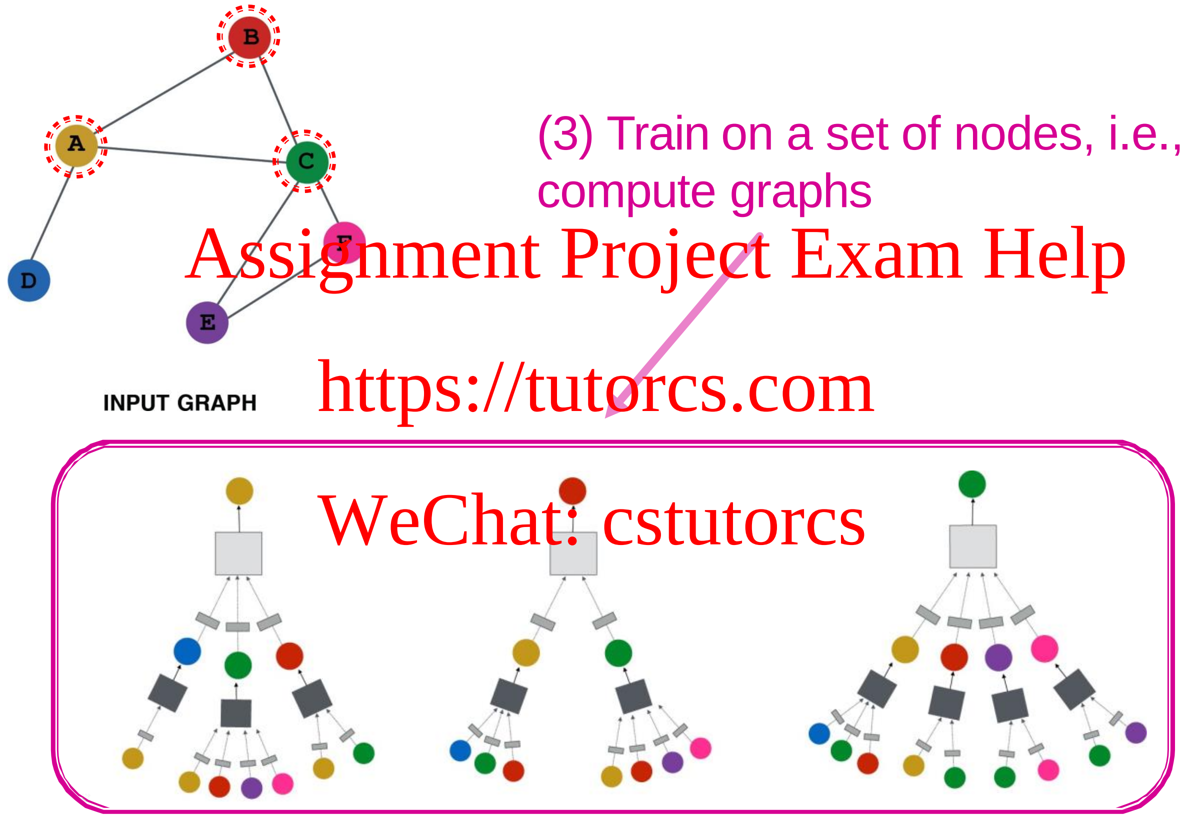
  - CE is the cross entropy
  - DEC is the decoder such as inner product

- **Node similarity** can be anything from previous lectures, e.g., a loss based on:
  - **Random walks** (node2vec, DeepWalk, struc2vec)
  - **Node proximity in the graph**

Model Design: Overview

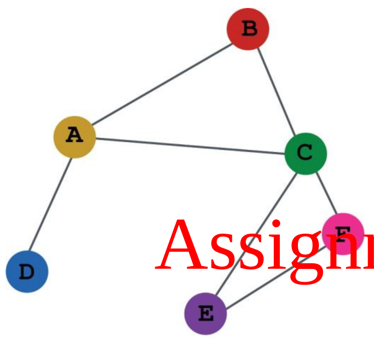


Model Design: Overview



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

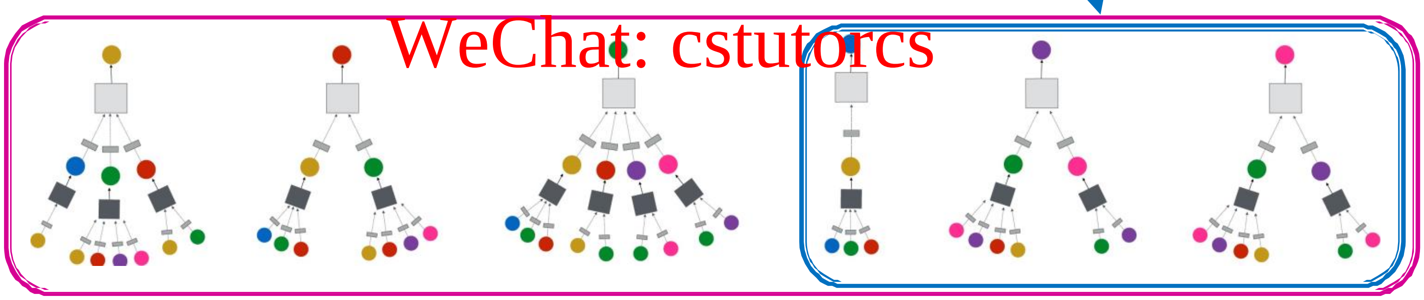
Model Design: Overview



INPUT GRAPH

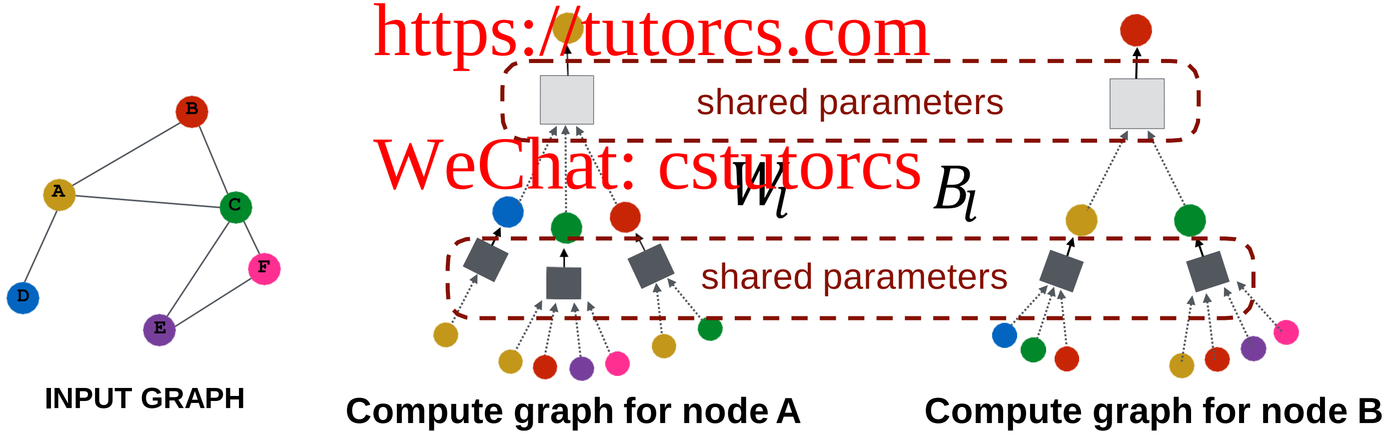
(4) Generate embeddings for nodes as needed

Even for nodes we never trained on!

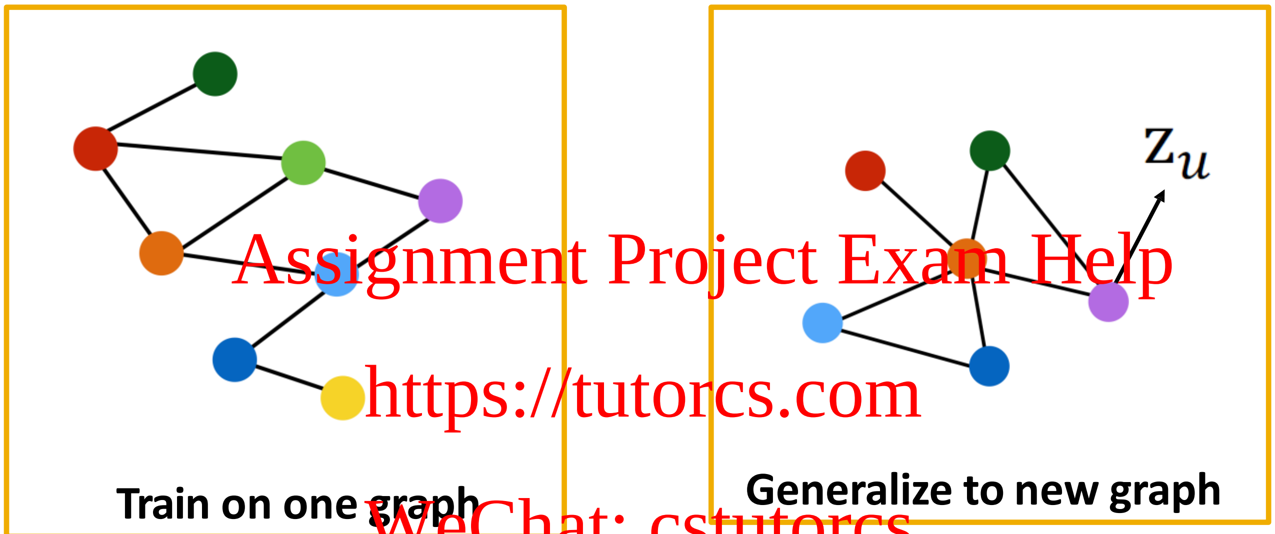


Inductive Capability

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



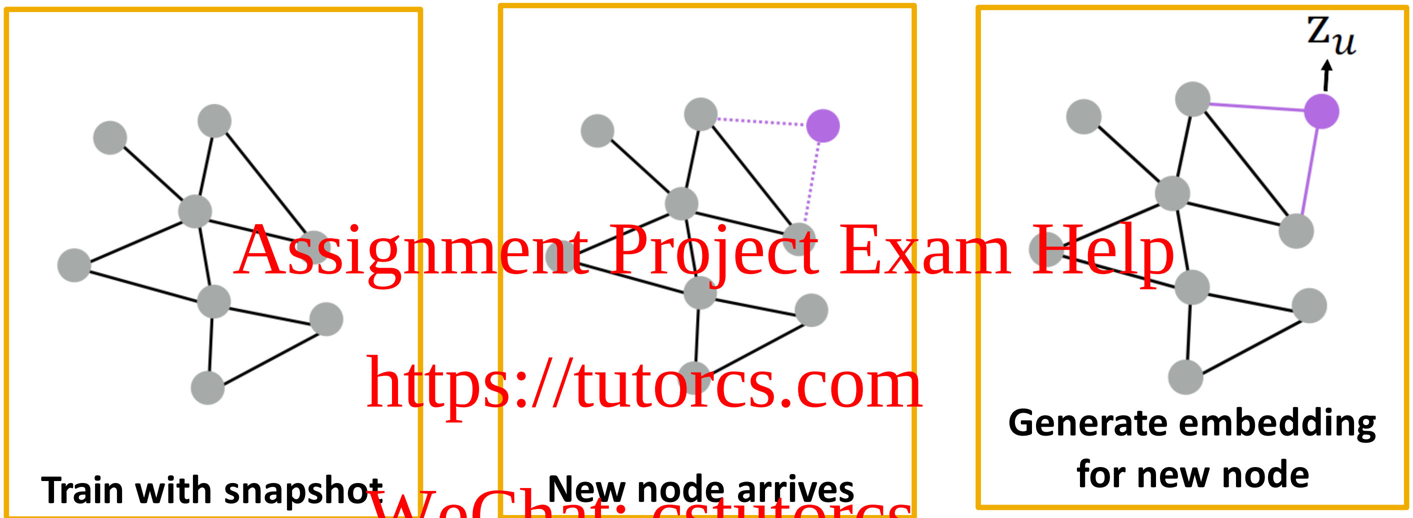
Inductive Capability: New Graphs



Inductive node embedding → Generalize to entirely unseen graphs

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

Inductive Capability: New Nodes

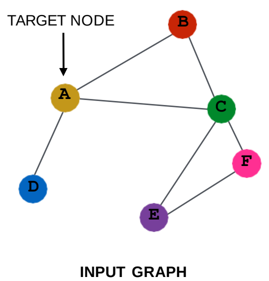


- Many application settings constantly encounter previously unseen nodes:
  - E.g., Reddit, YouTube, Google Scholar
- Need to generate new embeddings "on the fly"

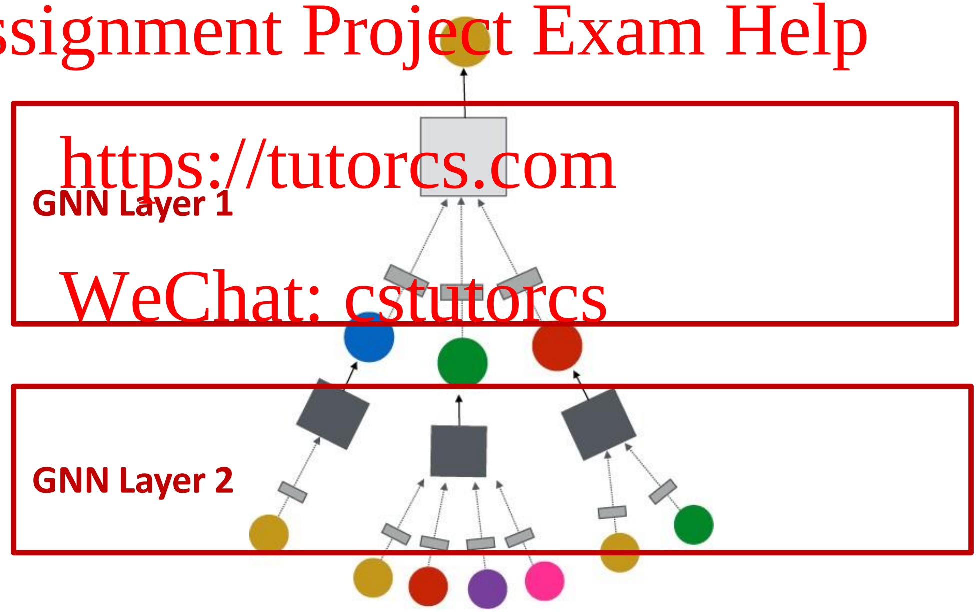
Stacking GNN Layers

How to connect GNN layers into a GNN?

1. Stack layers sequentially

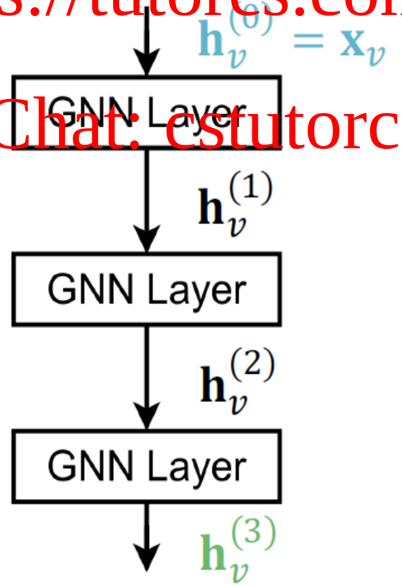


(3) Layer connectivity



Stacking GNN Layers

- **How to construct a Graph Neural Network?**
  - **The standard way:** Stack GNN layers sequentially
  - **Input:** Initial raw node feature $\mathbf{x}_v$
  - **Output:** Node embeddings $\mathbf{h}_v^{(L)}$ after L GNN layers

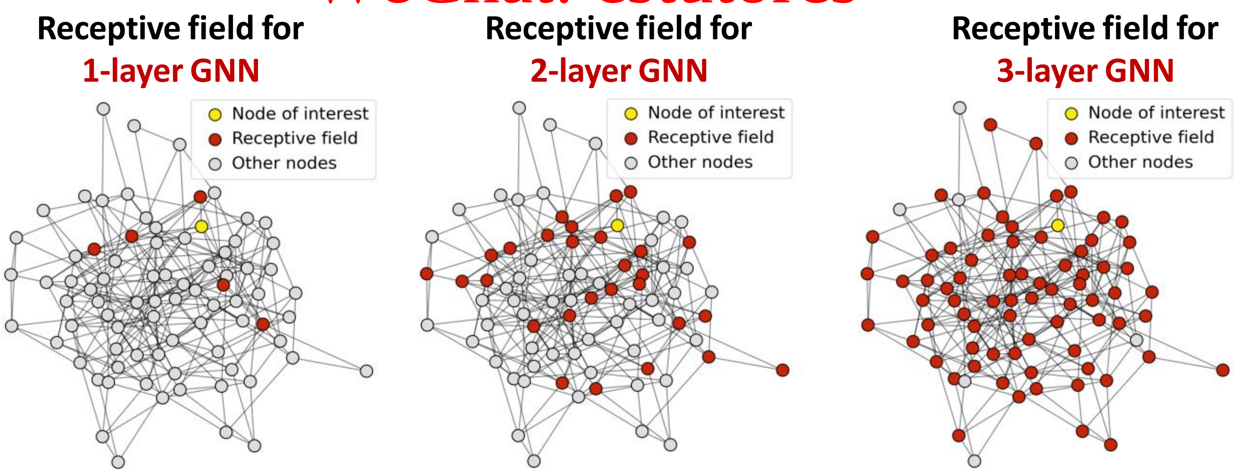


An Over-smoothing Problem

- **The Issue of stacking many GNN layers**
  - GNN suffers from **the over-smoothing problem**
- **The over-smoothing problem: all the node embeddings converge to the same value**
  - This is bad because we **want to use node embeddings to differentiate nodes**
- **Why does the over-smoothing problem happen?**

Receptive Field of a GNN

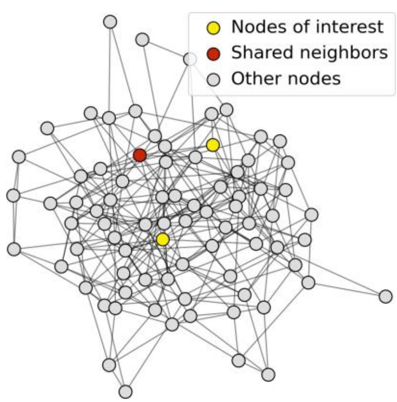
- **Receptive field:** the set of nodes that determine the embedding of a node of interest
  - **In a K-layer GNN, each node has a receptive field of K-hop neighborhood**



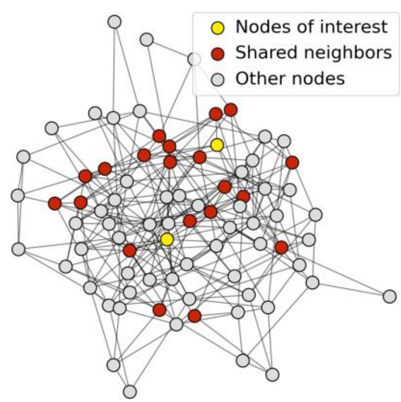
Receptive Field of a GNN

- **Receptive field overlap for two nodes**
  - **The shared neighbors quickly grows** when we increase the number of hops (num of GNN layers)

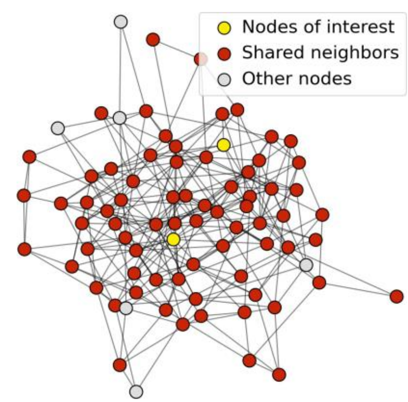
1-hop neighbor overlap **Only 1 node**



2-hop neighbor overlap **About 20 nodes**



3-hop neighbor overlap **Almost all the nodes!**



Receptive Field &Over-smoothing

- **We can explain over-smoothing via the notion of receptive field**
  - We knew **the embedding of a node is determined by its receptive field**
    - If two nodes **have highly-overlapped receptive fields, then their embeddings are highly similar**
  - **Stack many GNN layers → nodes will have highly-overlapped receptive fields → node embeddings will be highly similar → suffer from the over-smoothing problem**
- **Next:** how do we overcome over-smoothing problem?

Over-smoothing

Model	2-Layer	4-Layer	8-Layer	16-Layer	32-Layer	64-Layer
GCN-res	88.18±1.59	86.50±1.87	84.83±1.93	78.60±4.28	59.82±7.74	39.71±5.15
PairNorm	79.98±3.80	82.32±2.79	81.52±3.66	82.29±2.62	81.91±2.45	81.72±2.82
NodeNorm	**89.53**±1.29	88.60±1.36	88.02±1.67	88.41±1.25	88.30±1.30	87.40±2.06

Typical results of node classification accuracy on CoautorCS dataset

Design GNN Layer Connectivity

- **What do we learn from the over-smoothing problem?**
- **Lesson 1: Be cautious when adding GNN layers**
  - Unlike neural networks in other domains (CNN for image classification), **adding more GNN layers do not always help**
  - **Step 1: Analyze the necessary receptive field** to solve your problem. E.g., by computing the diameter of the graph
  - **Step 2:** Set number of GNN layers L to be a bit more than the receptive field we like. **Do not set $\boldsymbol{L}$ to be unnecessarily large**!

Expressive Power for Shallow GNNs

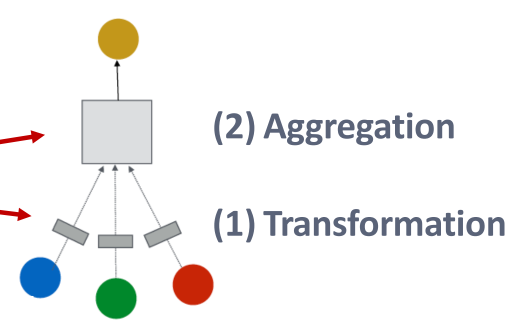
- **Question:** How to enhance the expressive power of a GNN, if the number of GNN layers is small?

- **Solution 1:** Increase the expressive power **within each GNN layer**

  - In our previous examples, each transformation or aggregation function only include one linear layer

  - We can **make aggregation / transformation become a deep neural network**!

If needed, each box could include a 3-layer MLP



Learning Outcome

- Generate node embeddings by aggregating neighborhood information
- Key distinctions are in how different approaches aggregate information across the layers