Graph Neural Networks

Representations

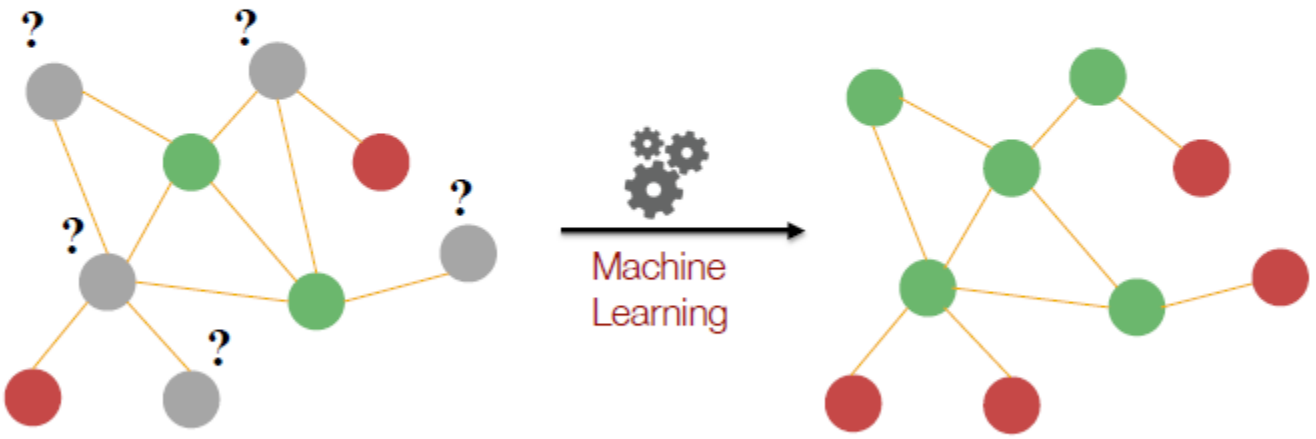
- Different ways of representing information
- Graph
- Matrix
- Vectors

- We can Transform from one representation to another.
- Encode it to obtain an embedded vector representation

Distributed Vector Representations

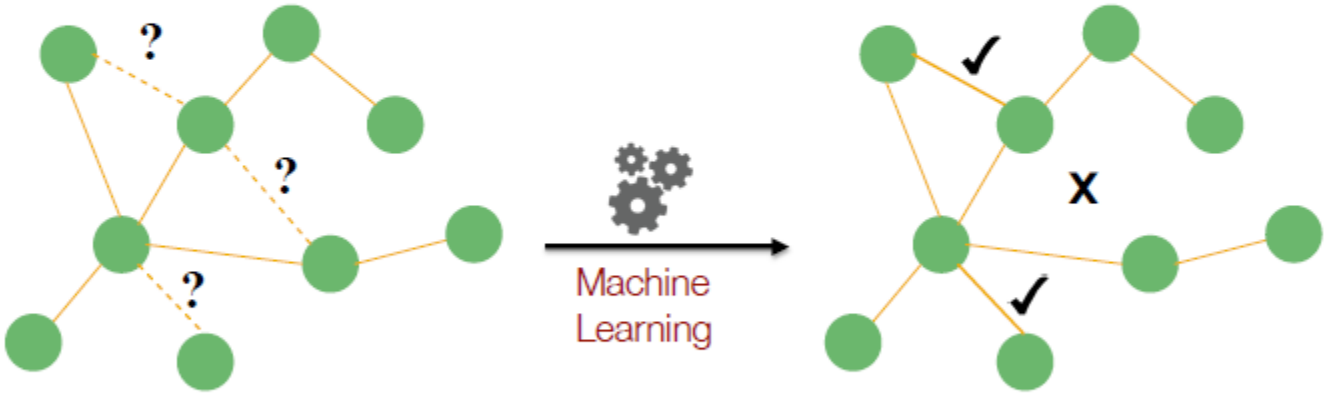
Graph Representation Learning

Machine Learning in Networks



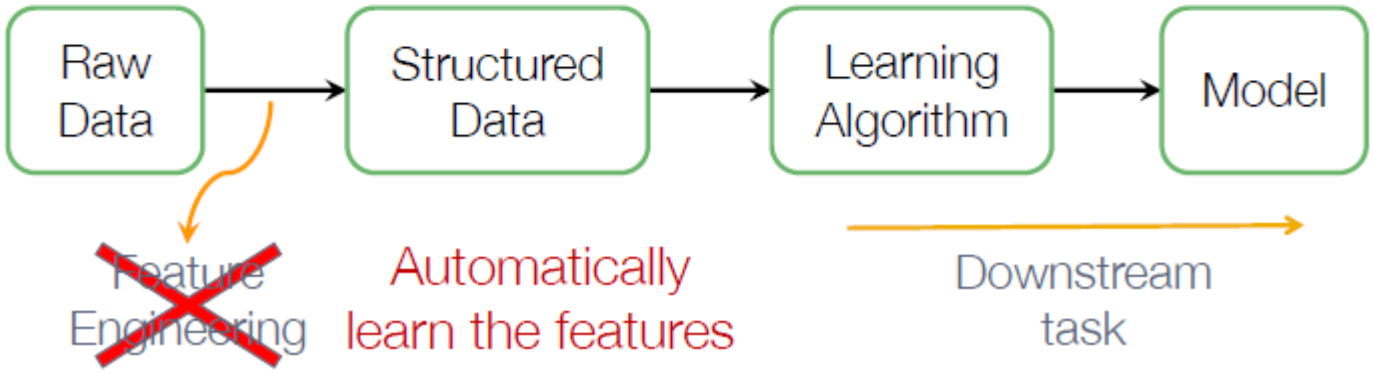
Node classification

Example: Link Prediction



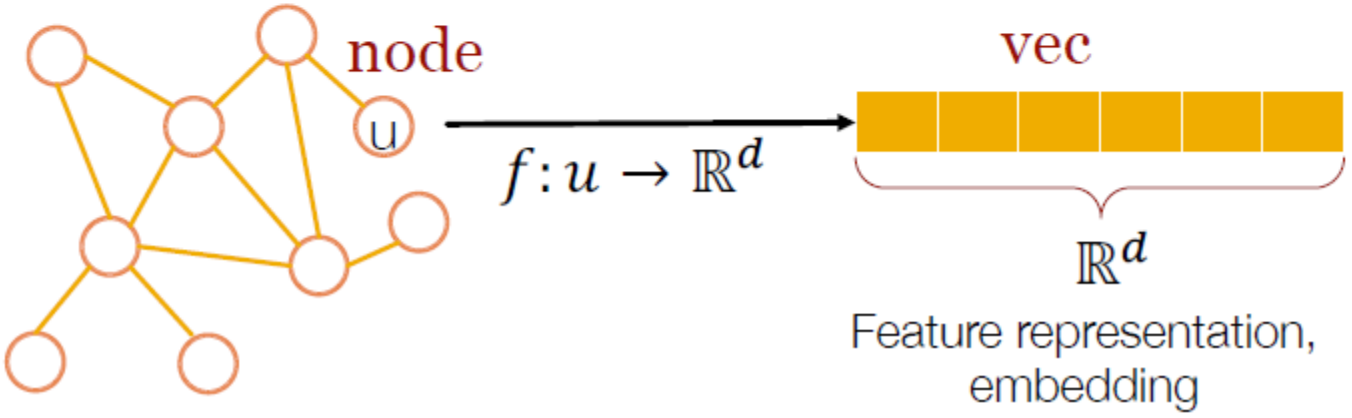
Machine Learning Lifecycle

- **(Supervised) Machine Learning Lifecycle requires feature engineering every single time!**



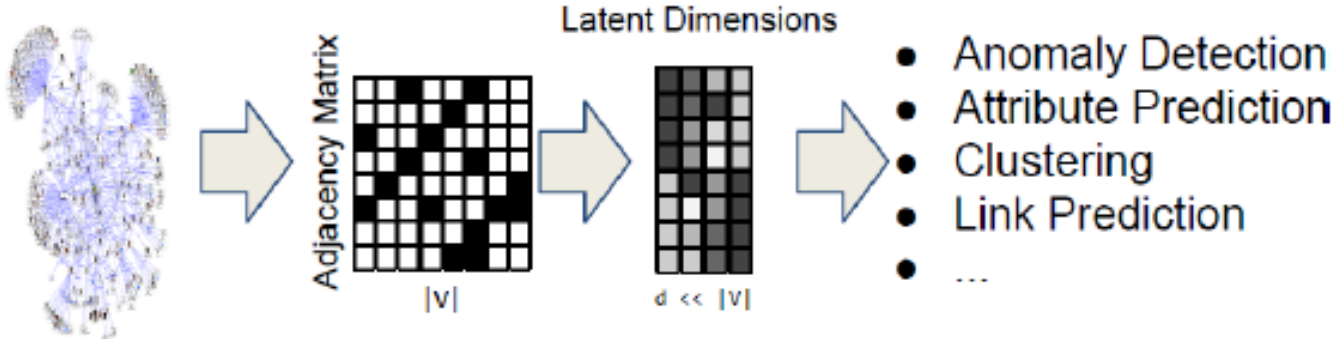
Feature Learning in Graphs

Goal: Efficient task-independent feature learning for machine learning in networks!



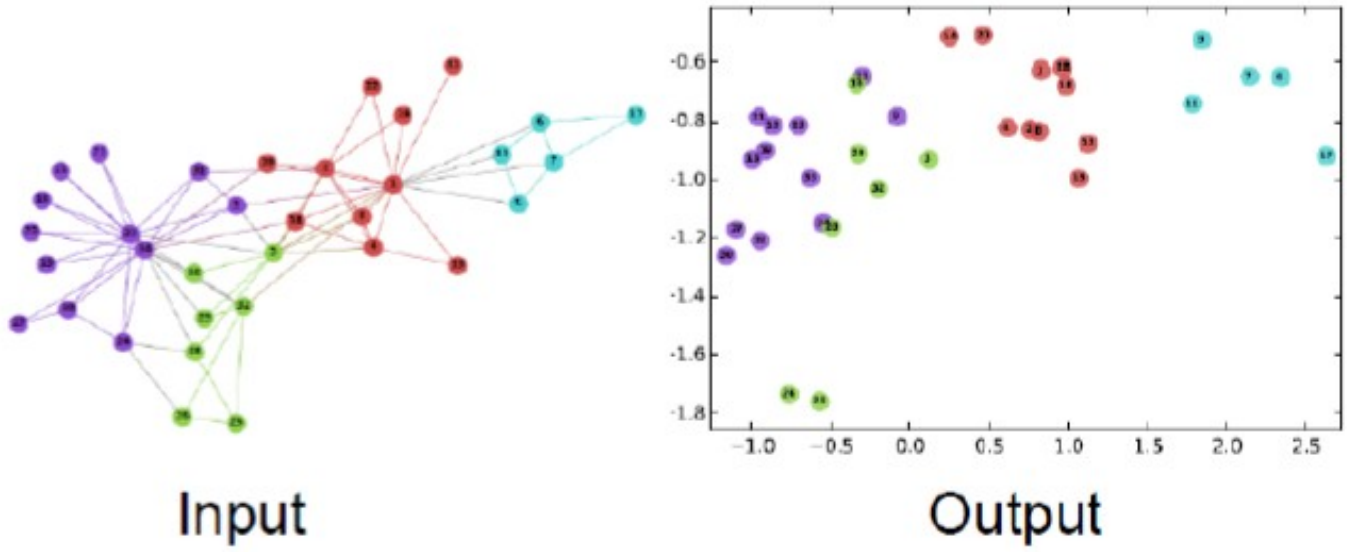
Why network embedding?

- **Task: We map each node in a network into a low-dimensional space**
  - Distributed representation for nodes
  - Similarity of embedding between nodes indicates their network similarity
  - Encode network information and generate node representation



Example Node Embedding

- **2D embedding of nodes of the Zachary's Karate Club network:**

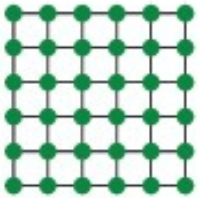
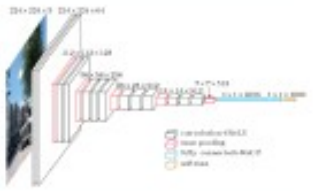


Input Output

Image from: Perozzi et al. DeepWalk: Online Learning of Social Representations. *KDD 2014.*

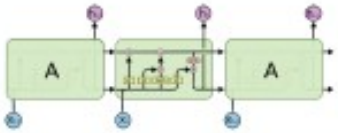
Why Is It Hard?

- Modern deep learning toolbox is designed for simple sequences or grids.
  - CNNs for fixed-size images/grids….

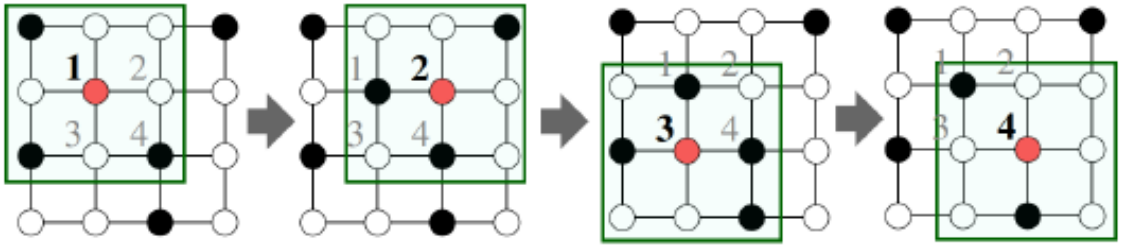
  - RNNs or word2vec for text/sequences…





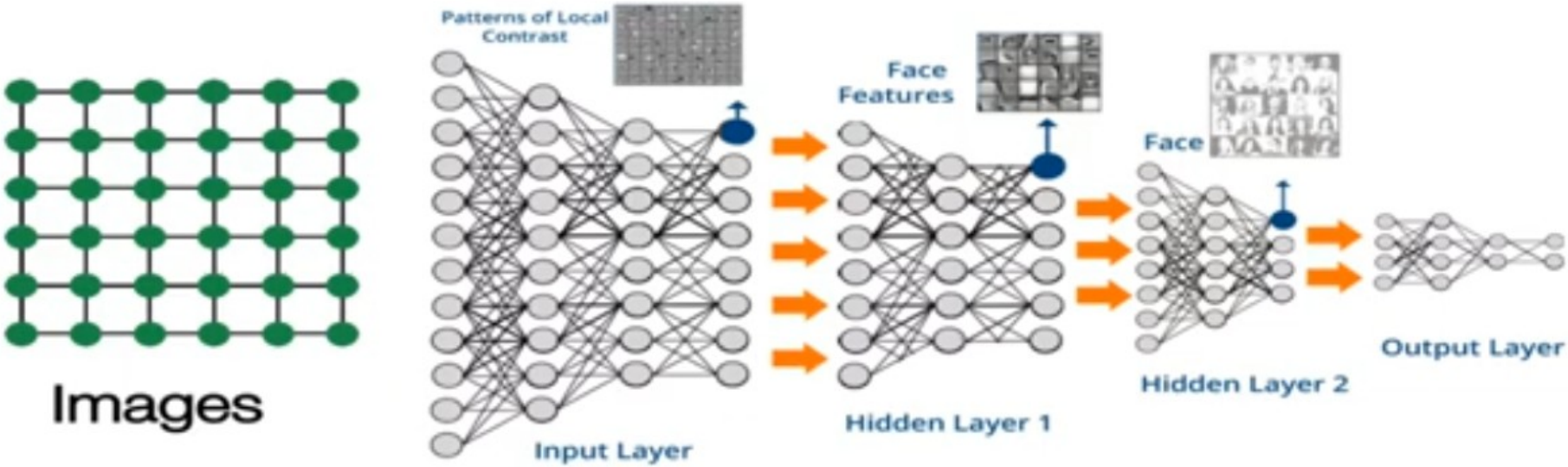
Why Is It Hard?

- But networks are far more complex!
  - Complex topographical structure (i.e., no spatial locality like grids)



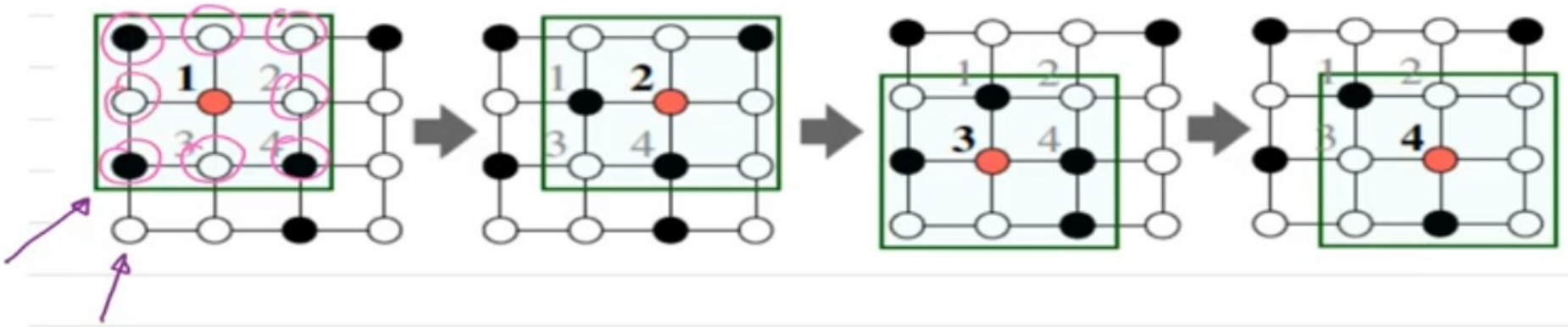
  - No fixed node ordering or reference point (i.e., the isomorphism problem)
  - Often dynamic and have multimodal features.

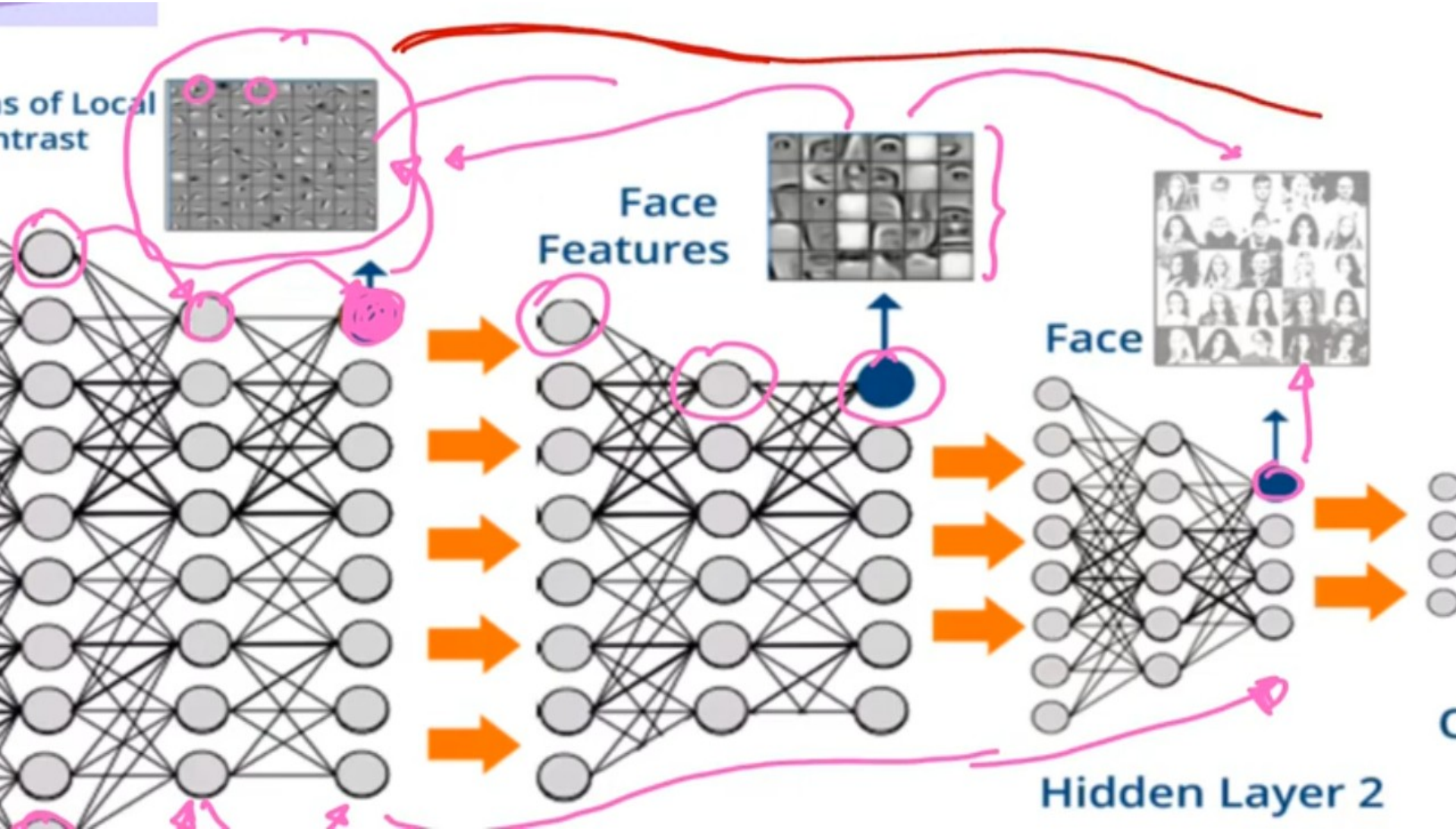
Why Convolution fails on Graphs?

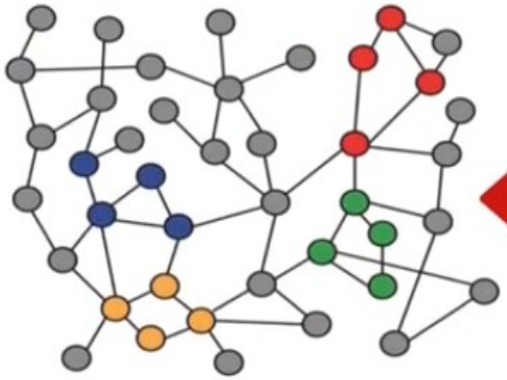


CNN

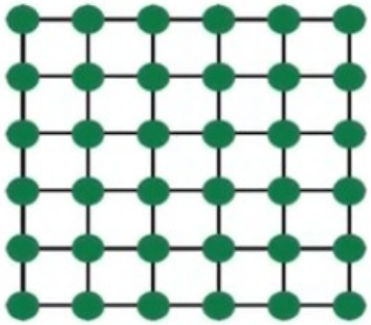
- Locality
- Aggregation
- {Composition (function of a function)

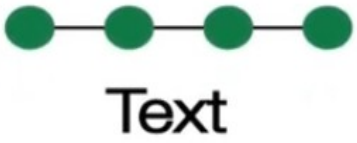



s of Local
ntrast
Face
Features
Face
Hidden Layer 2



VS.





→ Arbitrary size

→ Complex topology (no spatial locality)

→ Graphs are Non-Euclidean

→ No fixed node ordering

→ Grid (spatial locality)

→ Structured data

→ Deep Learning is designed for grides & sequences.

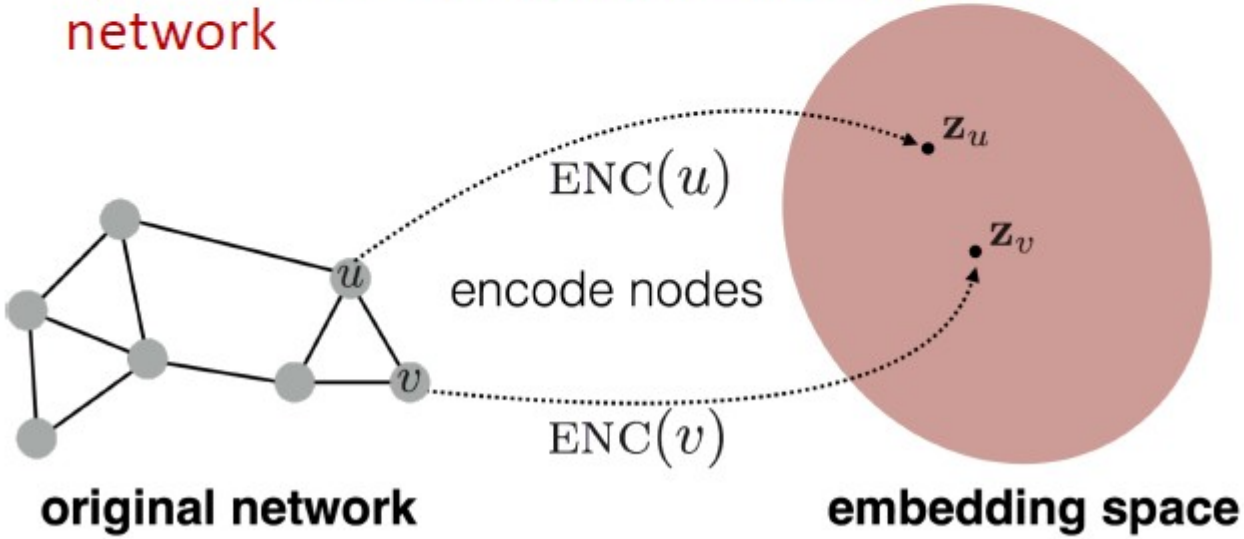
Embedding Nodes

Setup

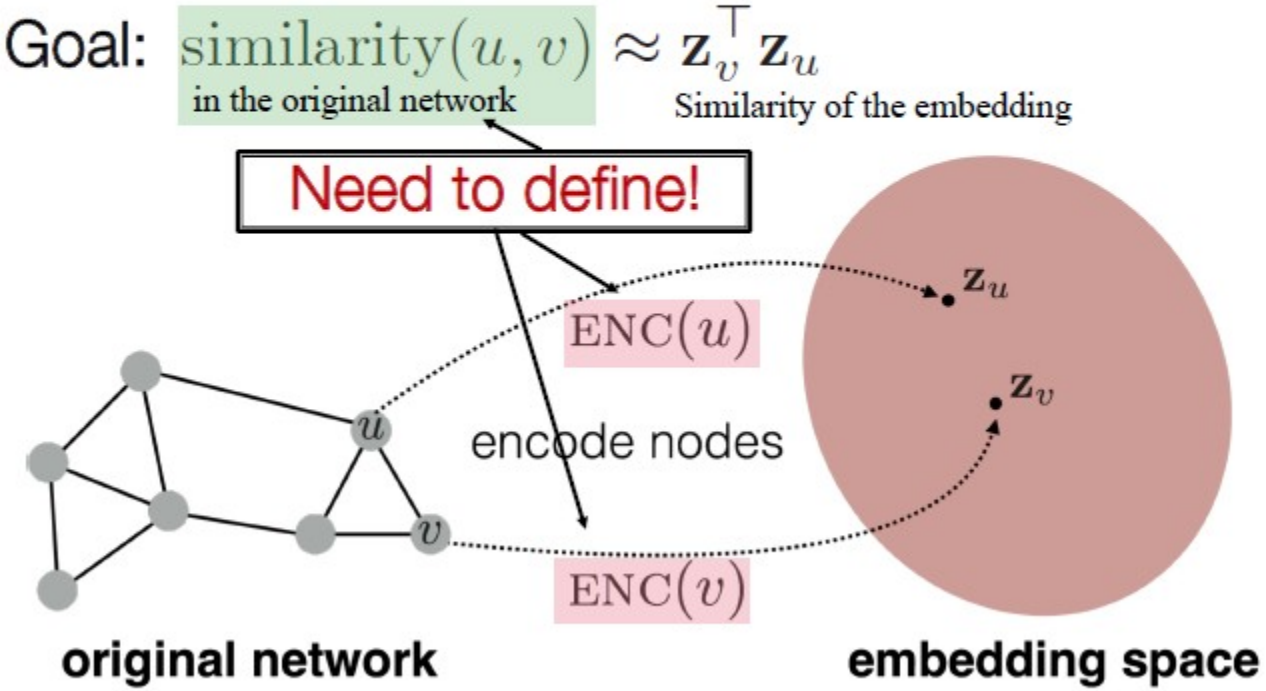
- **Assume we have a graph G:**
  - V is the vertex set.
  - A is the adjacency matrix (assume binary).
  - **No node features or extra information is used!**

Embedding Nodes

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



Embedding Nodes



Learning Node Embeddings

1. **Define an encoder** (i.e., a mapping from nodes to embeddings)
2. **Define a node similarity function** (i.e., a measure of similarity in the original network).
3. **Optimize the parameters of the encoder so that:**

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

Two Key Components

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

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

$\mathbf{d}$-dimensional embedding

node in the input graph

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

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

Similarity of u and v in the original network

dot product between node embeddings

"Shallow" Encoding

- Simplest encoding approach: **encoder is just an embedding-lookup**

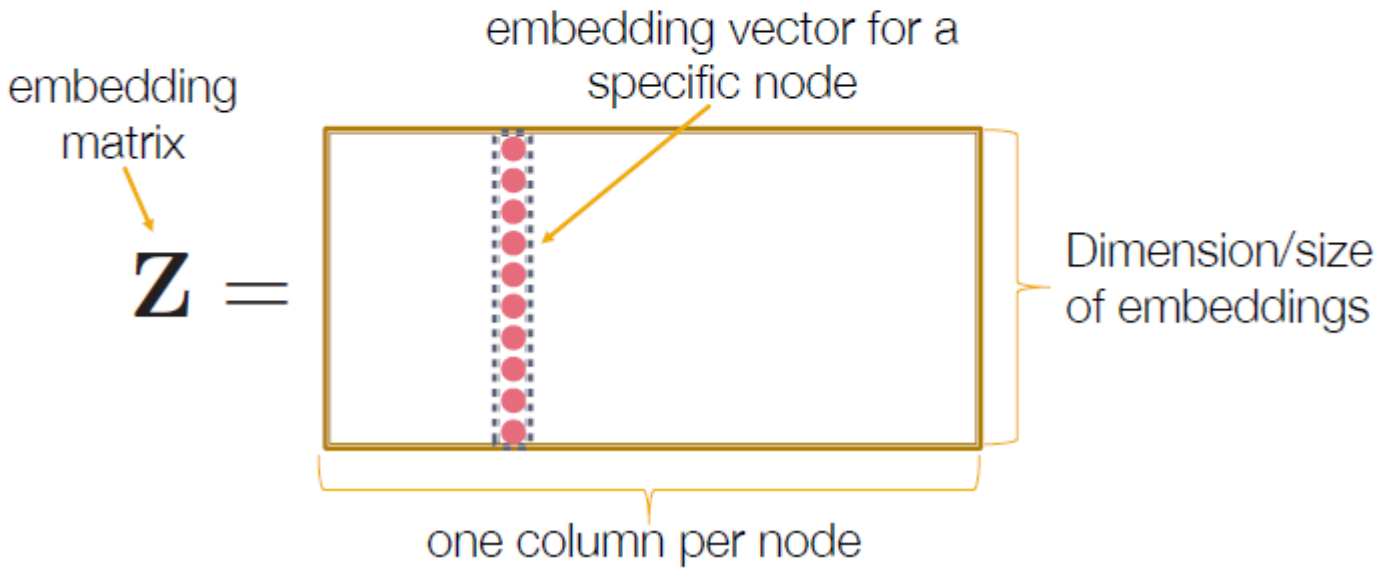
$$\text{ENC}(v) = \mathbf{Z}\mathbf{v}$$

$\mathbf{Z} \in \mathbb{R}^{d \times |\mathcal{V}|}$ matrix, each column is node embedding [what we learn!]

$\mathbf{v} \in \mathbb{I}^{|\mathcal{V}|}$ indicator vector, all zeroes except a one in column indicating node *v*

"Shallow" Encoding

- Simplest encoding approach: **encoder is just an embedding-lookup**



"Shallow" Encoding

Simplest encoding approach: **encoder is just an embedding-lookup**

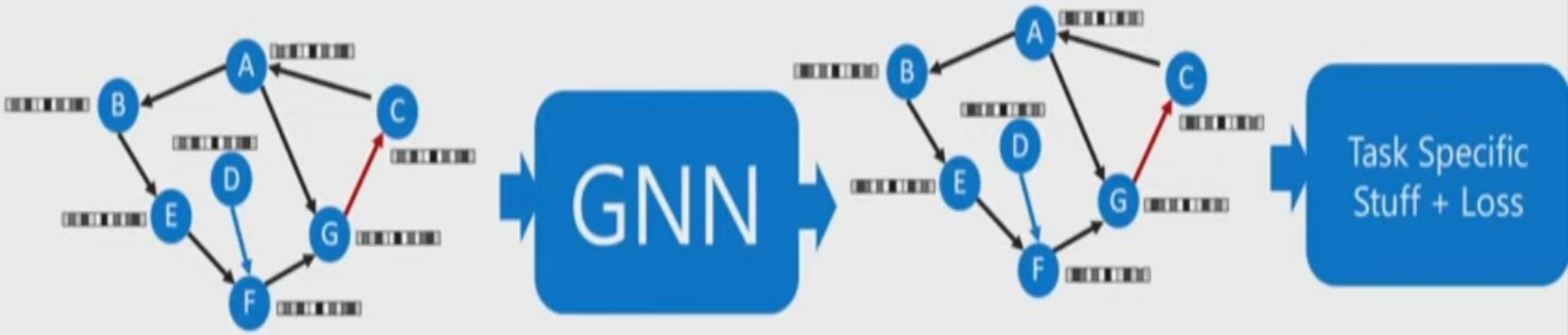
Each node is assigned a unique embedding vector

Many methods: node2vec, DeepWalk, LINE

How to Define Node Similarity?

- Key choice of methods is **how they define node similarity.**
- E.g., should two nodes have similar embeddings if they….
  - are connected?
  - share neighbors?
  - have similar "structural roles"?
  - …?

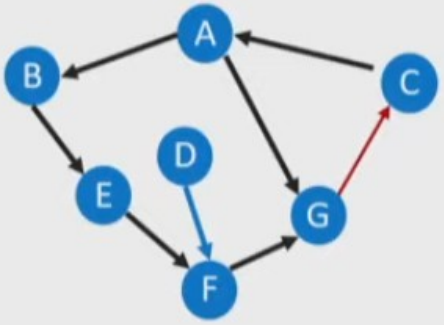
Graph Neural Networks



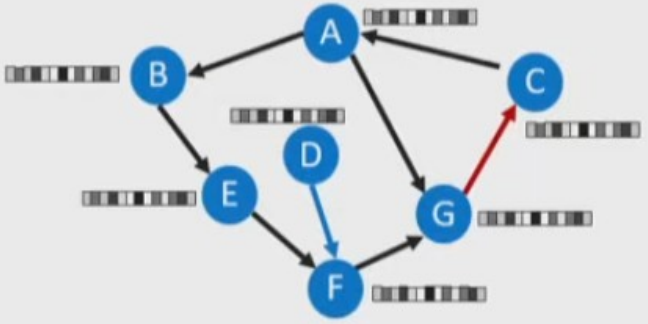
Initial Representation
of each node

Output Representations
of each Node

Graph Neural Networks

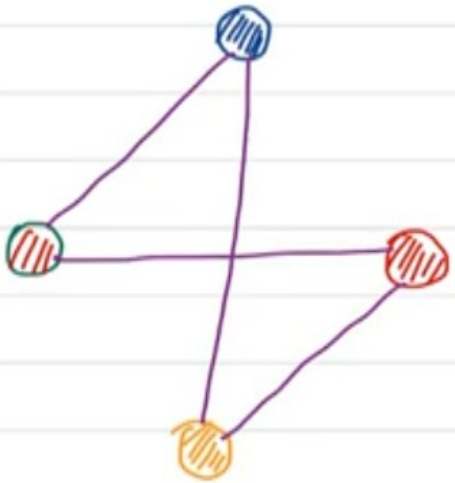


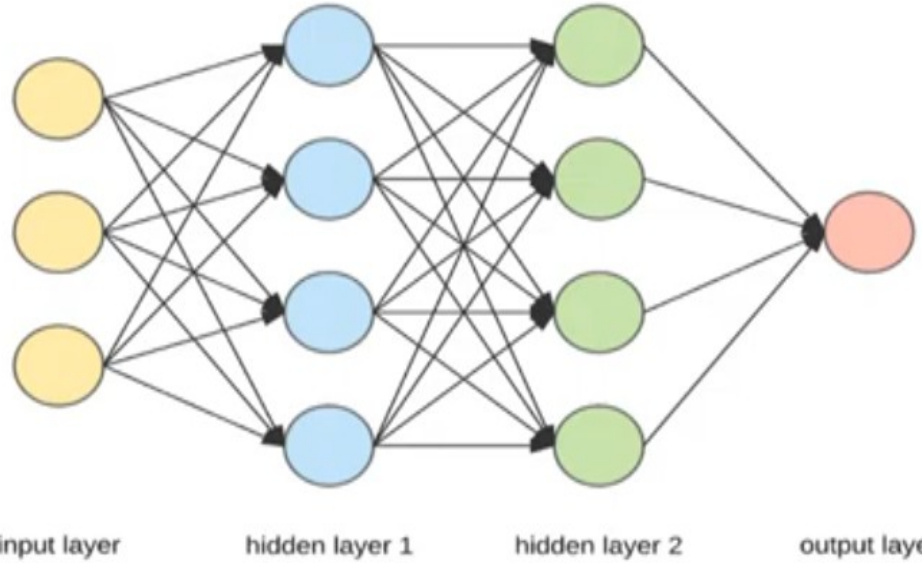
Graph Representation
of Problem

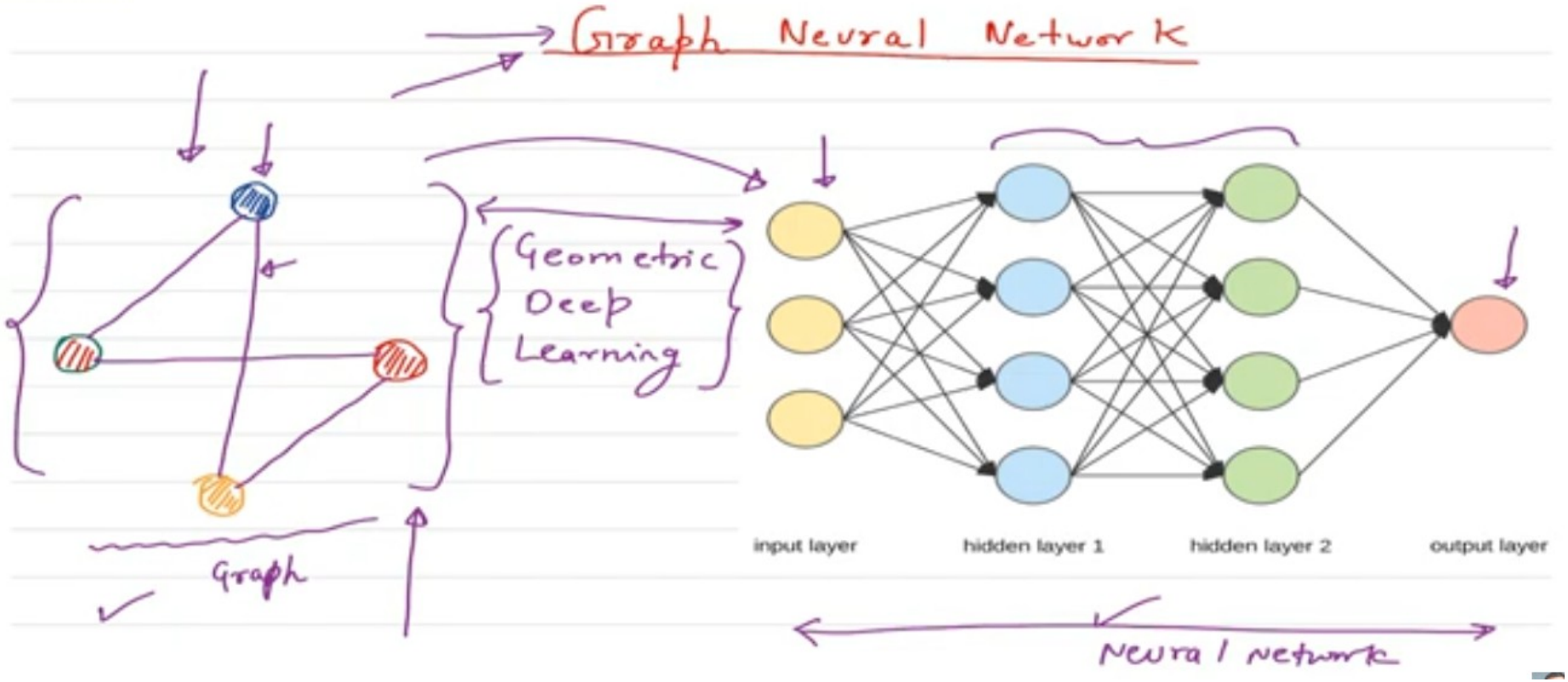


Initial Representation
of each node

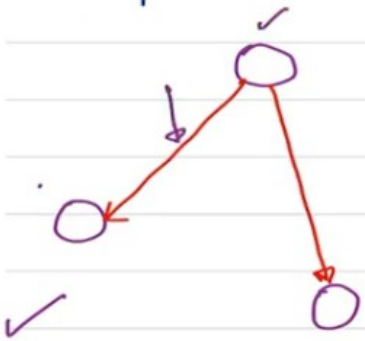
→ Graph Neural Network



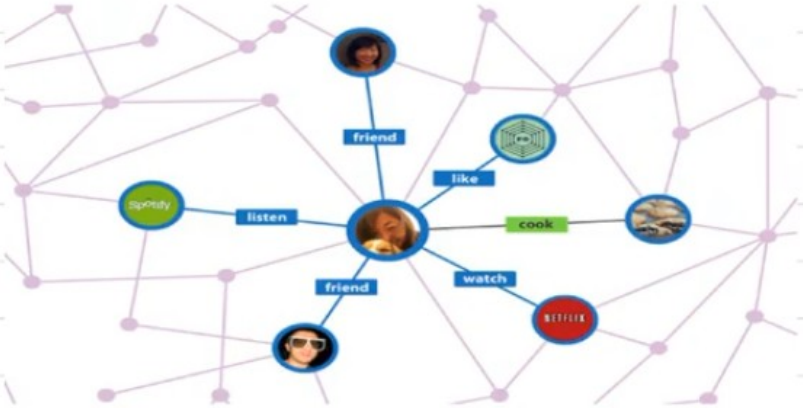


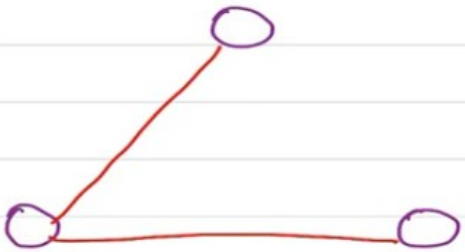

Graph Neural Network
Geometric
Deep
Learning
Graph
input layer
hidden layer 1
hidden layer 2
output layer
Neural network

Graphs can be directed or undirected.



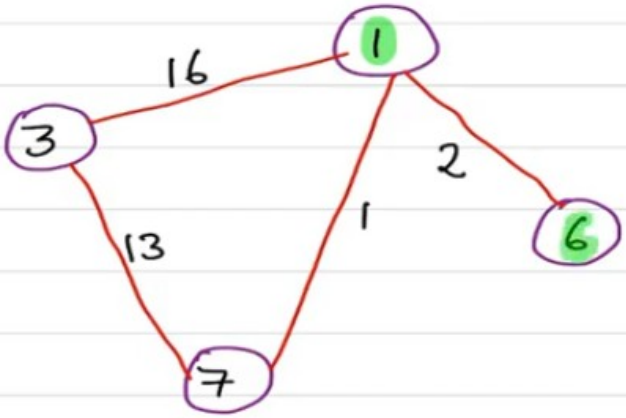
Twitter →



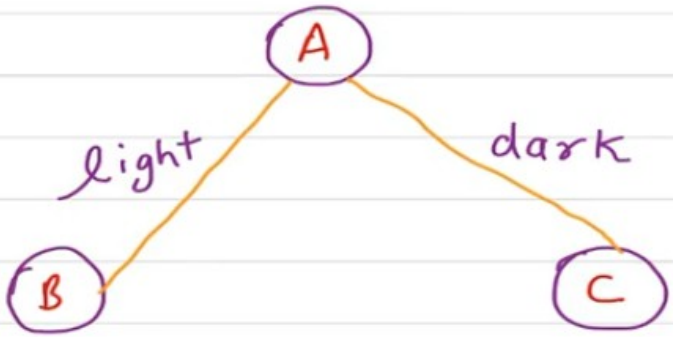


→ Facebook friend request.

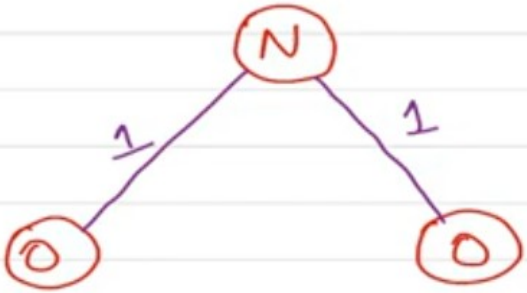
Graphs can have labels on their edges / nodes.



Numerical labels

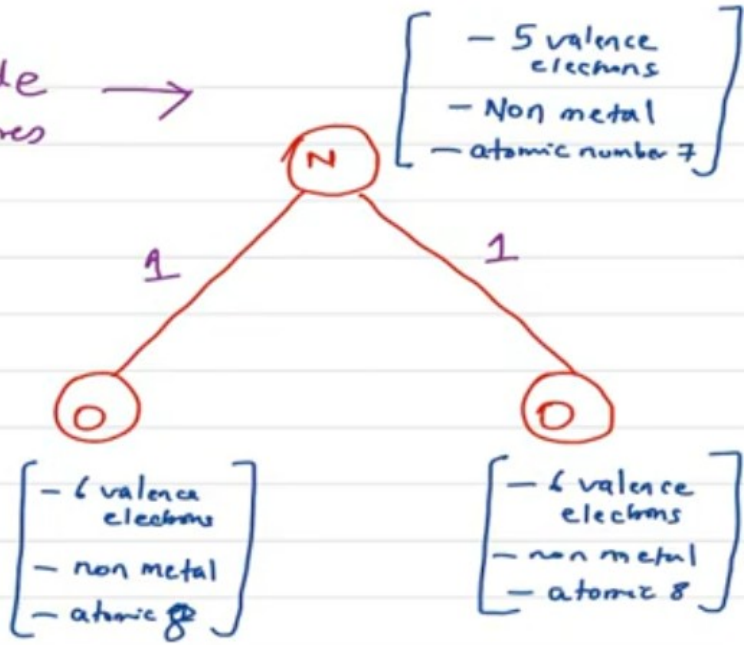


Textual labels



NO_2 : Unique labels

Node Features →



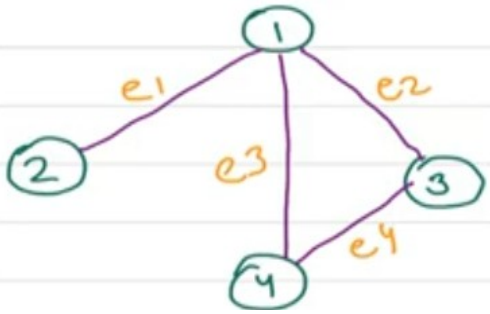
Note: Do not mix labels with features.

Analogy: Node is a person, node label is person name & node feature are person's characteristics.

How to convert graph in a format that computer can accept.

Incidence matrix (I) → A matrix of size $n \times m$ where n are number of nodes & edges of a graph

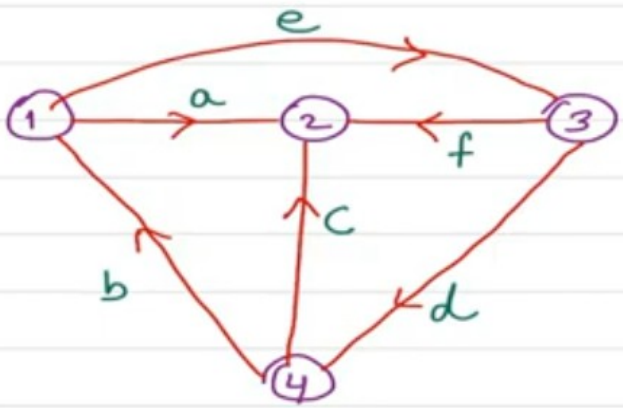
Undirected Graph



Entry is 1 if node & edge are incident (related) & 0 if not.

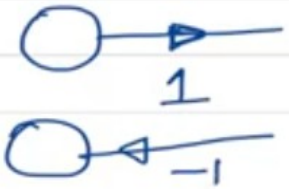
	e_1	e_2	e_3	e_4
1	1	1	1	0
2	1	0	0	0
3	0	1	0	1
4	0	0	1	1

Directed Graph.



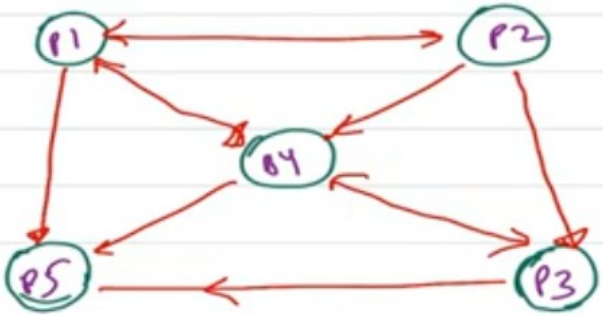
	a	b	c	d	e	f
1	1	-1	0	0	1	0
2	-1	0	-1	0	0	-1
3	0	0	0	1	-1	1
4	0	1	1	-1	0	0

I



Adjacency Matrix (A)

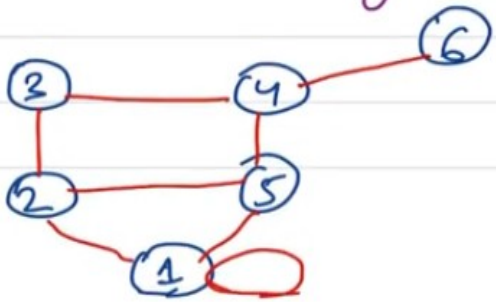
It is also called as connection matrix, is a square matrix. The component of the matrix is 0 or 1 depending upon whether the nodes of the graph are adjacent or not



	P_1	P_2	P_3	P_4	P_5
P_1	0	1	0	1	1
P_2	1	0	1	1	0
P_3	0	0	0	1	1
P_4	1	0	1	0	1
P_5	0	0	0	0	0

An adjacency matrix can be weighted, which means each edge has an associated value. So instead of 1s, the value is put in the respective matrix co-ordinate.

Degree matrix (D) → A diagonal matrix which contains information about the degree of each node — that is the number of edges attached to each node.



$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{bmatrix} 4 & & & & & \\ & 3 & & & & \\ & & 2 & & & \\ & & & 3 & & \\ & & & & 3 & \\ & & & & & 1 \end{bmatrix} \end{array}$$

Laplacian matrix (Graph Laplacian)

$$L = \{D - A\}$$

→ Also Known as Laplace–Beltrami operator is a measure of smoothness of a vertex i.e how quickly it changes b/w adjacent vertex

Laplacian matrix (Graph Laplacian)

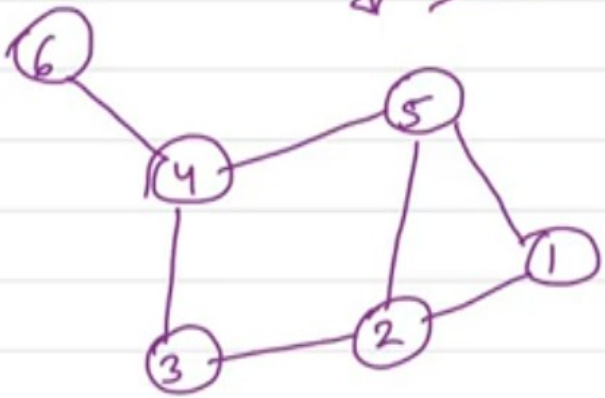
$$L = \{D - A\}$$

Also known as {Laplace–Beltrami operator} is a measure of smoothness of a vertex i.e how quickly it changes b/w adjacent vertex.

$$\{\nabla f\} = f(x+1, y) - f(x, y)$$

{∇f} → smoothness of the image

"2D" images



Degree matrix

$$\begin{bmatrix} 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Adjacency

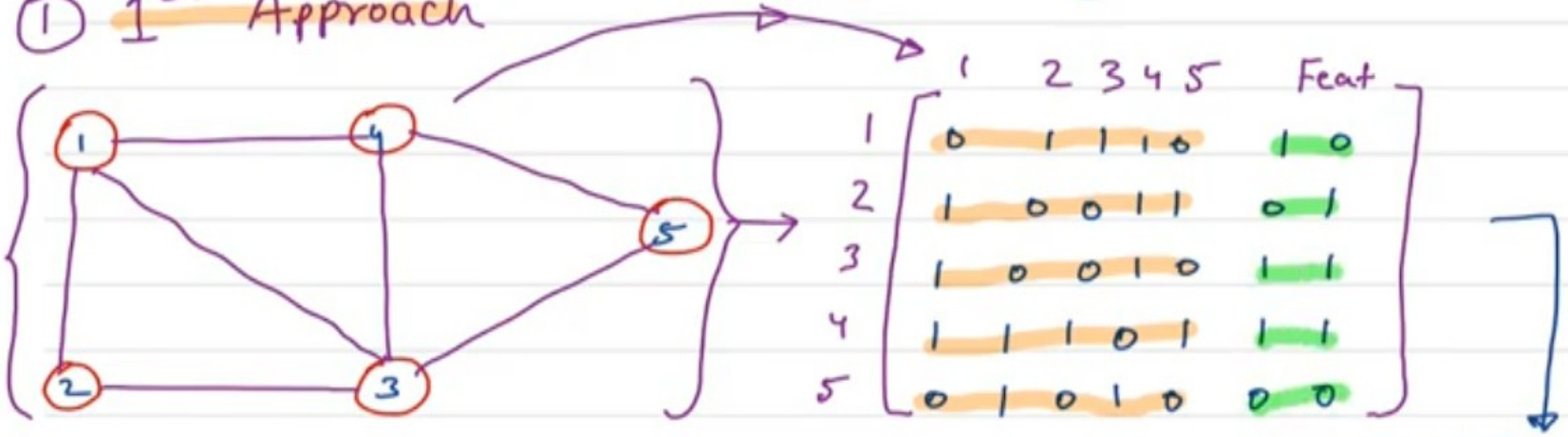
$$\begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

Laplacian: $D - A$

$$\begin{bmatrix} 2 & -1 & 0 & 0 & -1 & 0 \\ -1 & 3 & -1 & 0 & -1 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & -1 & 3 & -1 & -1 \\ -1 & -1 & 0 & -1 & 3 & 0 \\ 0 & 0 & 0 & -1 & 0 & 1 \end{bmatrix}$$

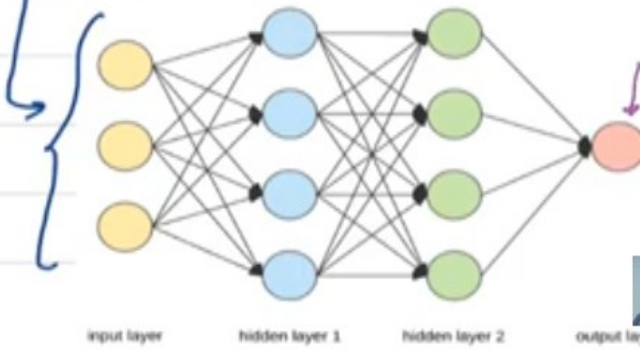
How to train Graphs using Neural Network ?

① 1st Approach



⊗ Not invariant to node ordering

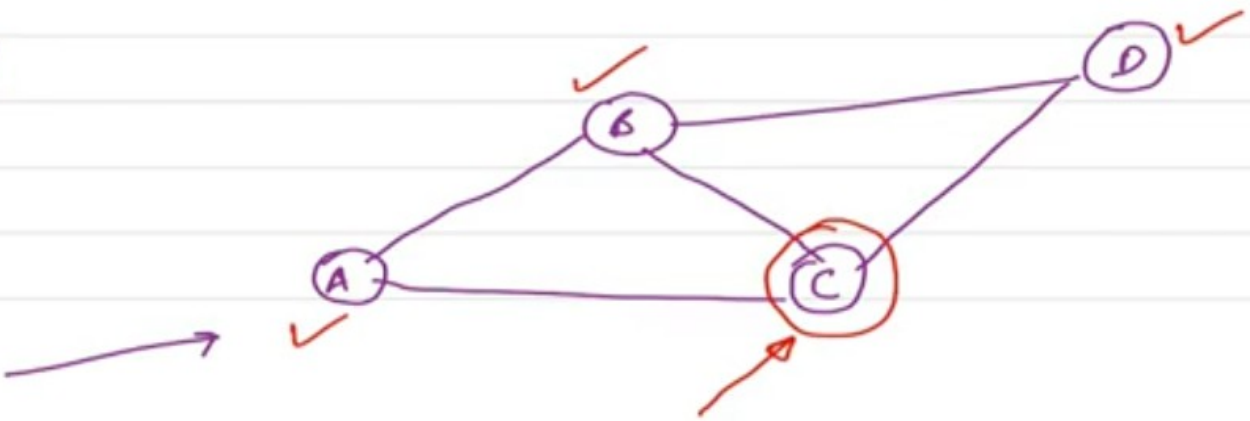
⊗ Not applicable to graphs of different sizes.



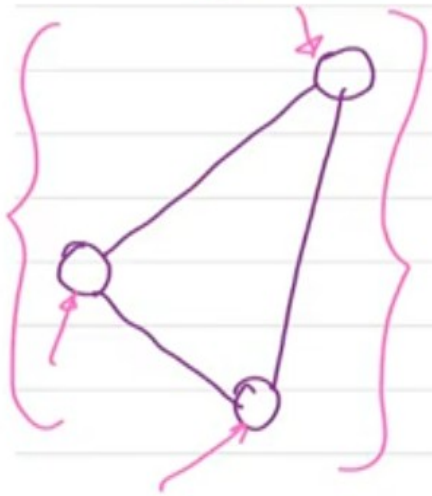
2nd Approach Use same approach as in {CNN}

GNN
- locality (neighborhood)
- Aggregation
- Stacking layers (composition)

Ex



For any graph



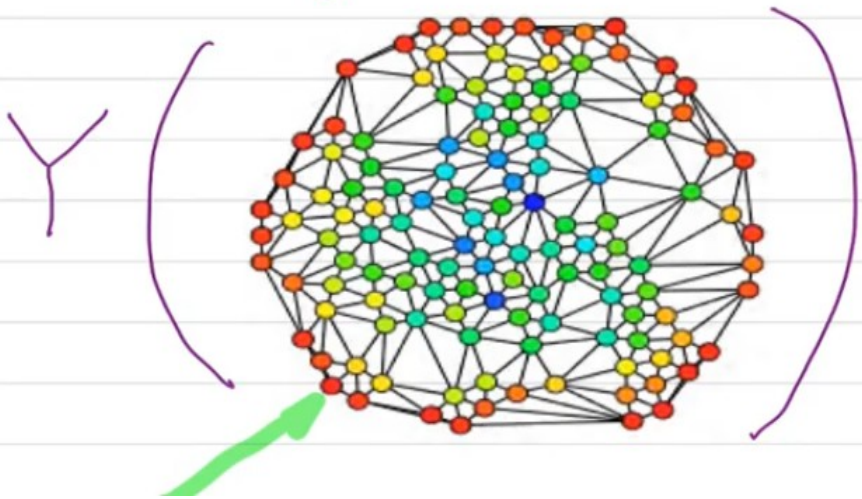
→ Graph $G = (V, E)$

* A is adjacency matrix of size $N \times N$

* X is the node feature matrix of size $(N \times M)$

Node Embedding.

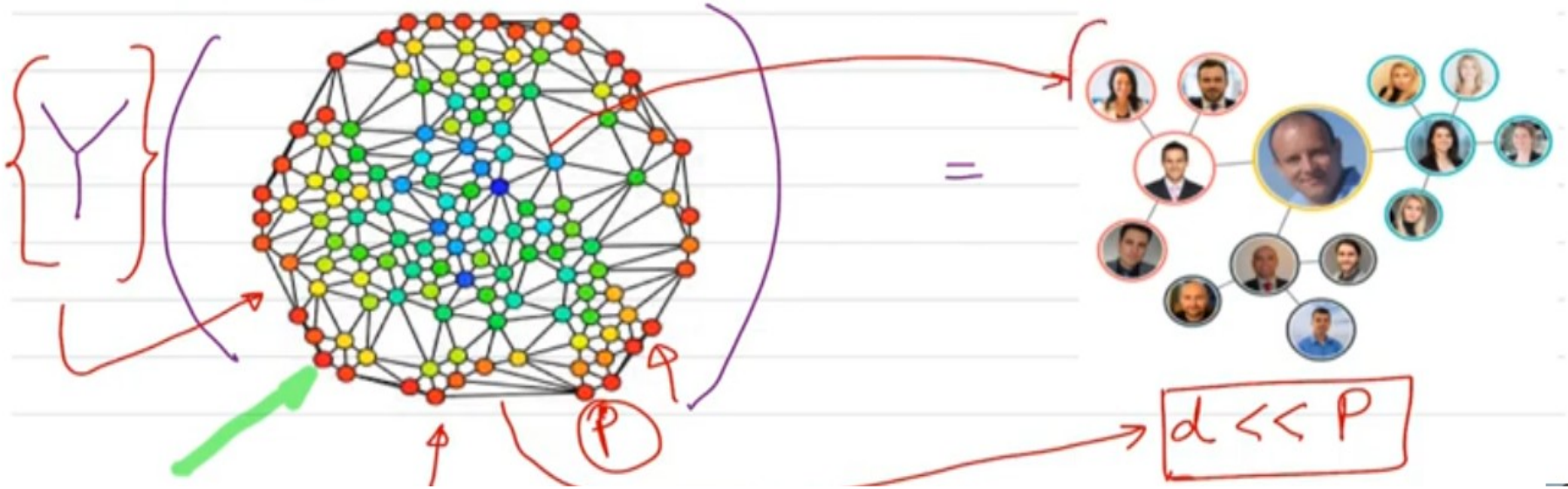
Target: Map nodes to d dimensional embeddings such that similar nodes in the graph are embedded close together



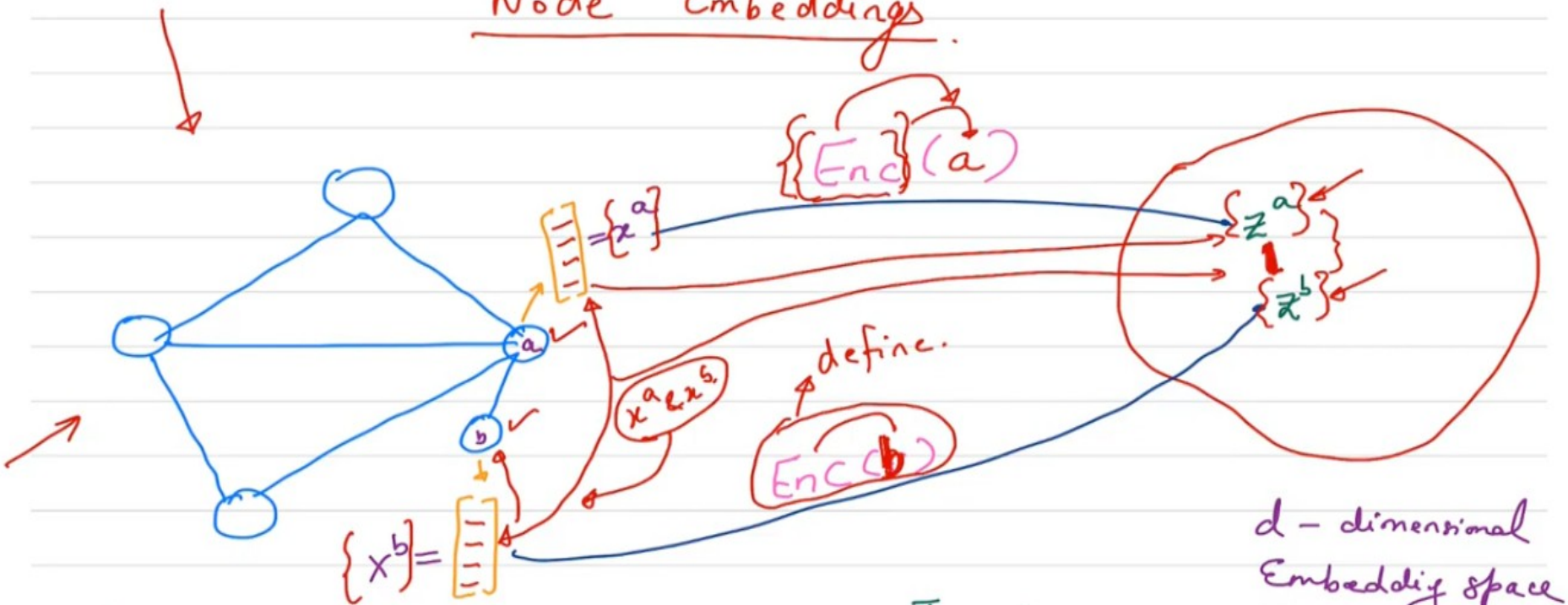
=

Node Embedding

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

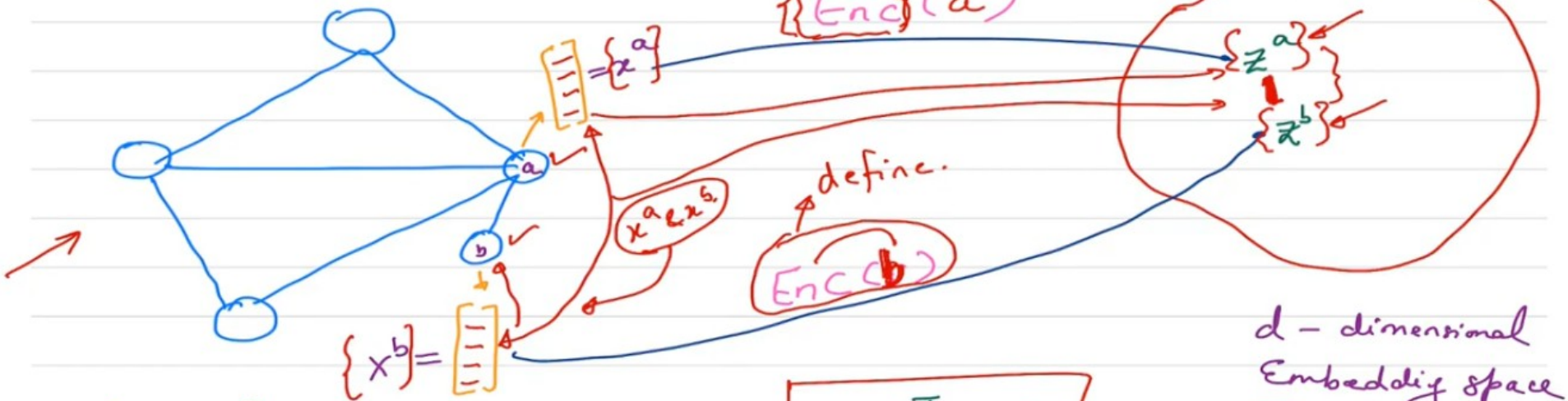


Node Embeddings



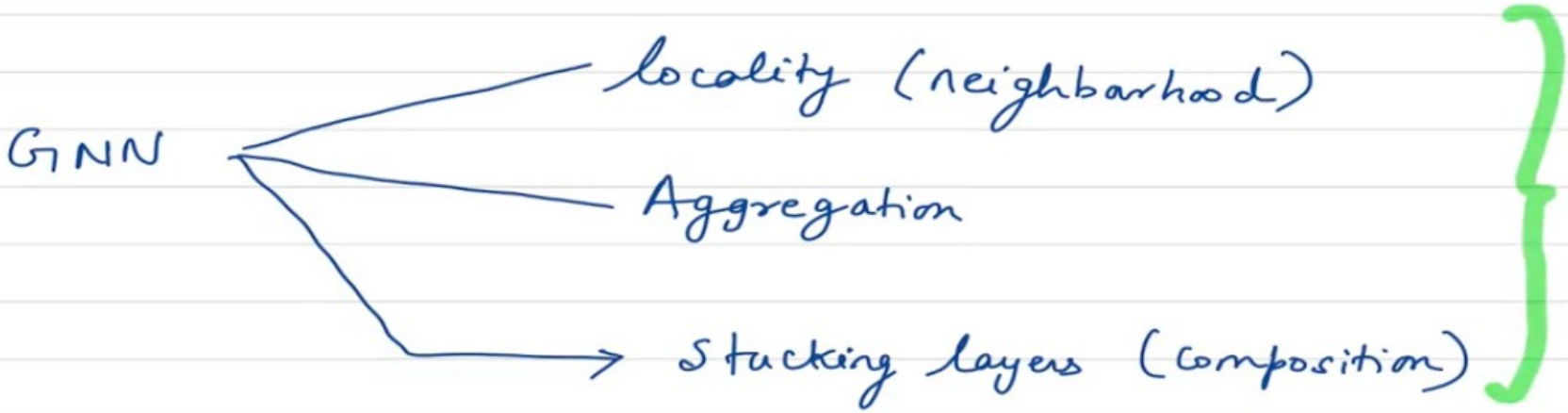
Goal: Similarity $(a, b) \approx (z^a)^T (z^b)$

Node Embeddings



Goal: Similarity $(a, b) = (z^a)^T (z^b) = (x^a)^T (x^b)$

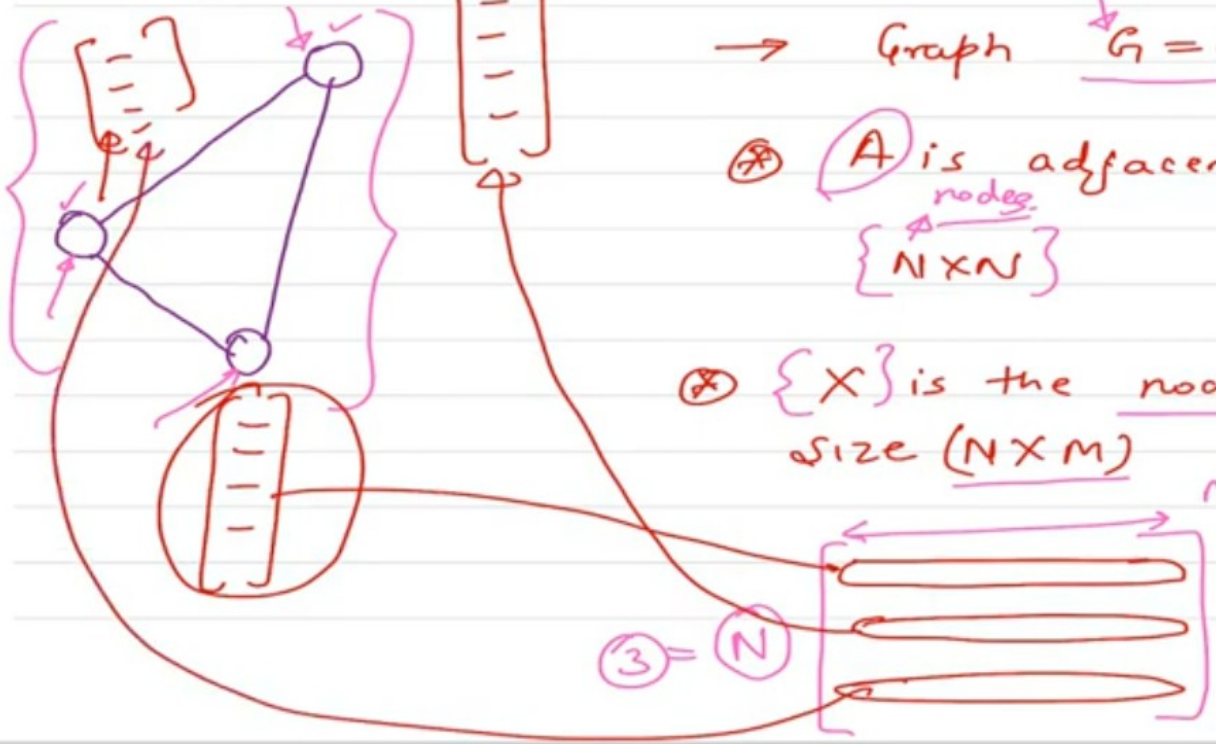
How to come up with this Encoder function.



Graph Neural Networks

and Neural Message Passing

For any graph



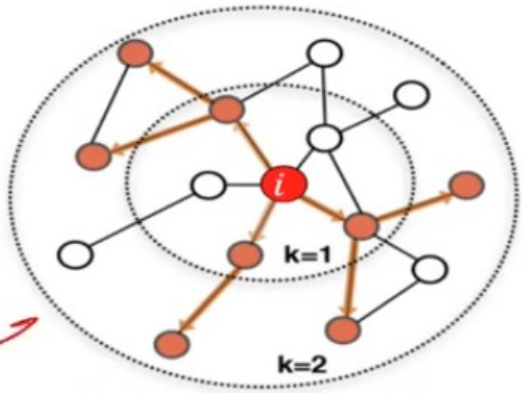
→ Graph $G = (V, E)$

* A is adjacency matrix of size (nodes) $\{N \times N\}$

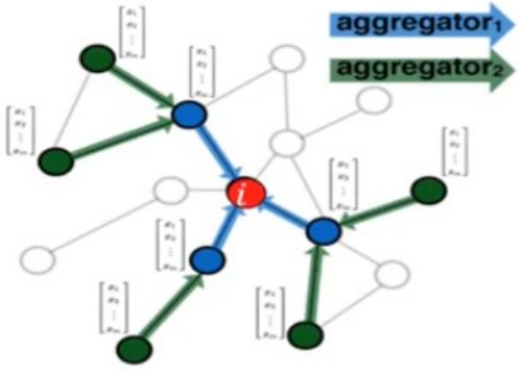
* $\{X\}$ is the node feature matrix of size $(N \times M)$

$M = 4$

⊛ Locality information using computational graph.

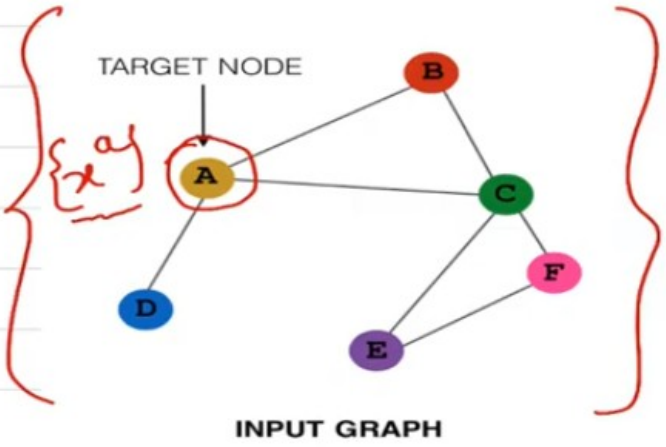


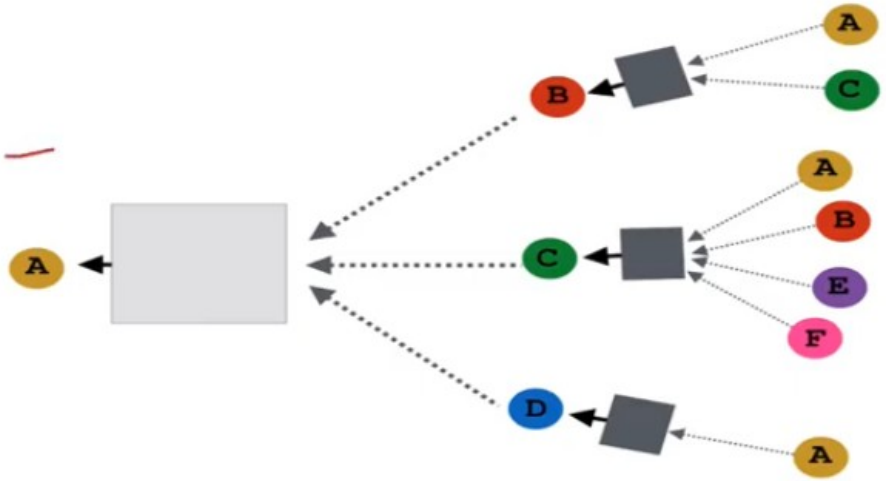
Determine node computation graph



Propagate and transform information

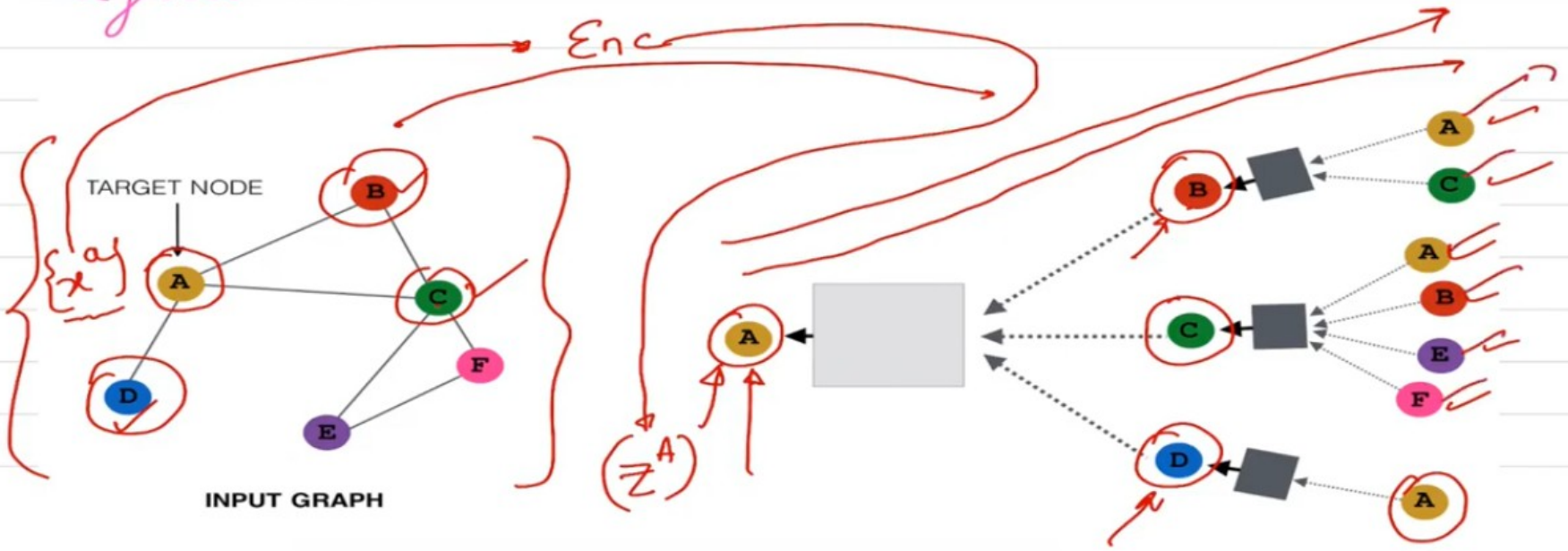
→ Generate node embedding based on local neighborhood.





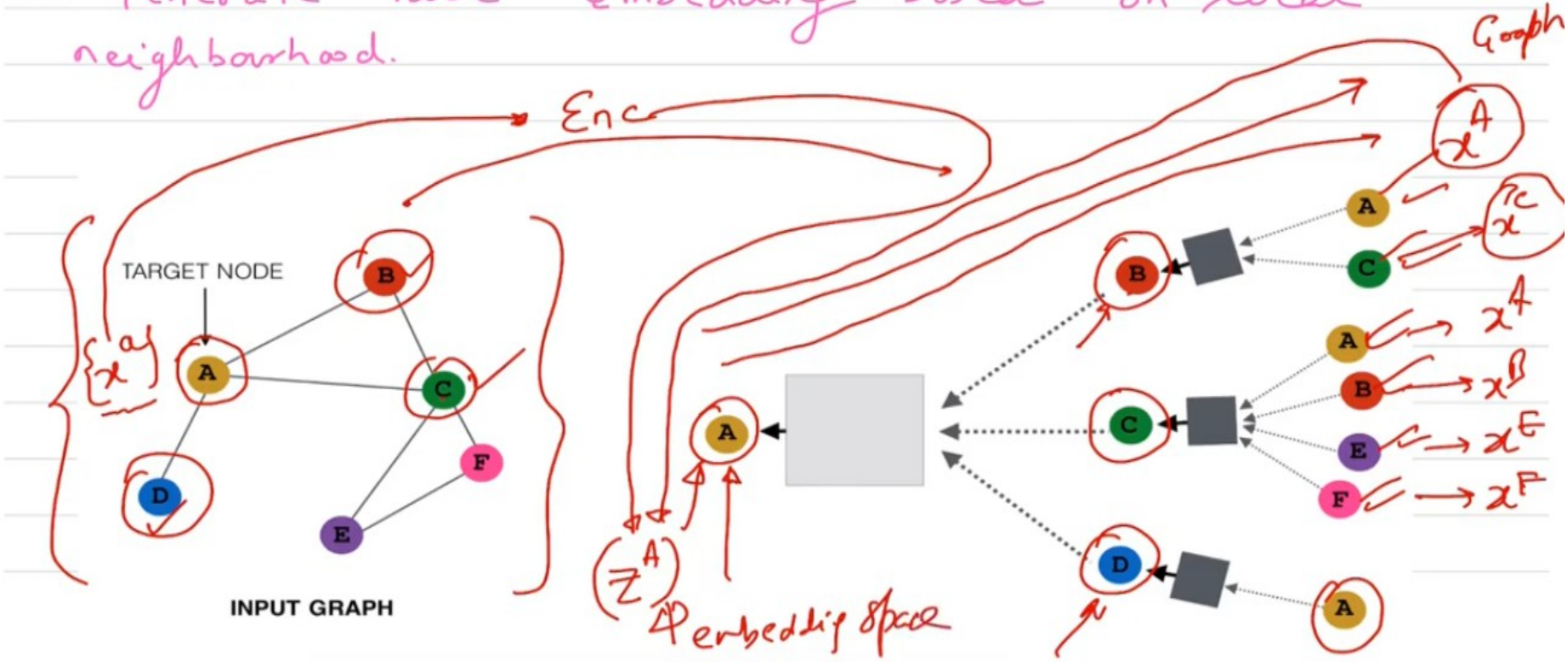
Aggregate (Neighbors)

→ Generate node embedding based on local neighborhood.

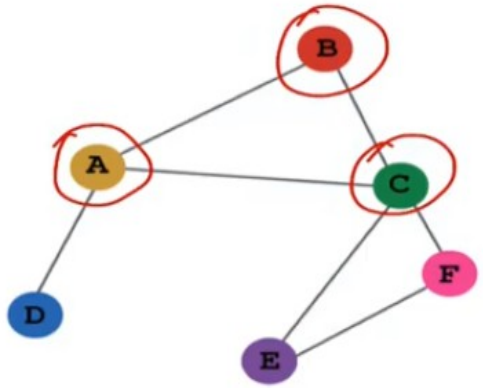


Aggregate (Neighbors)

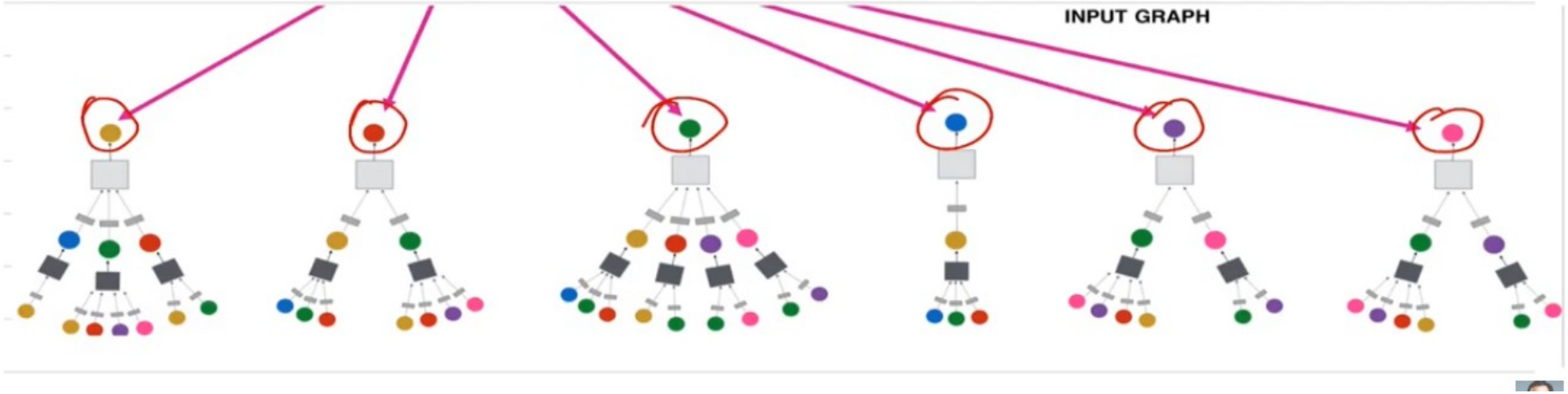
→ Generate node embedding based on local neighborhood.

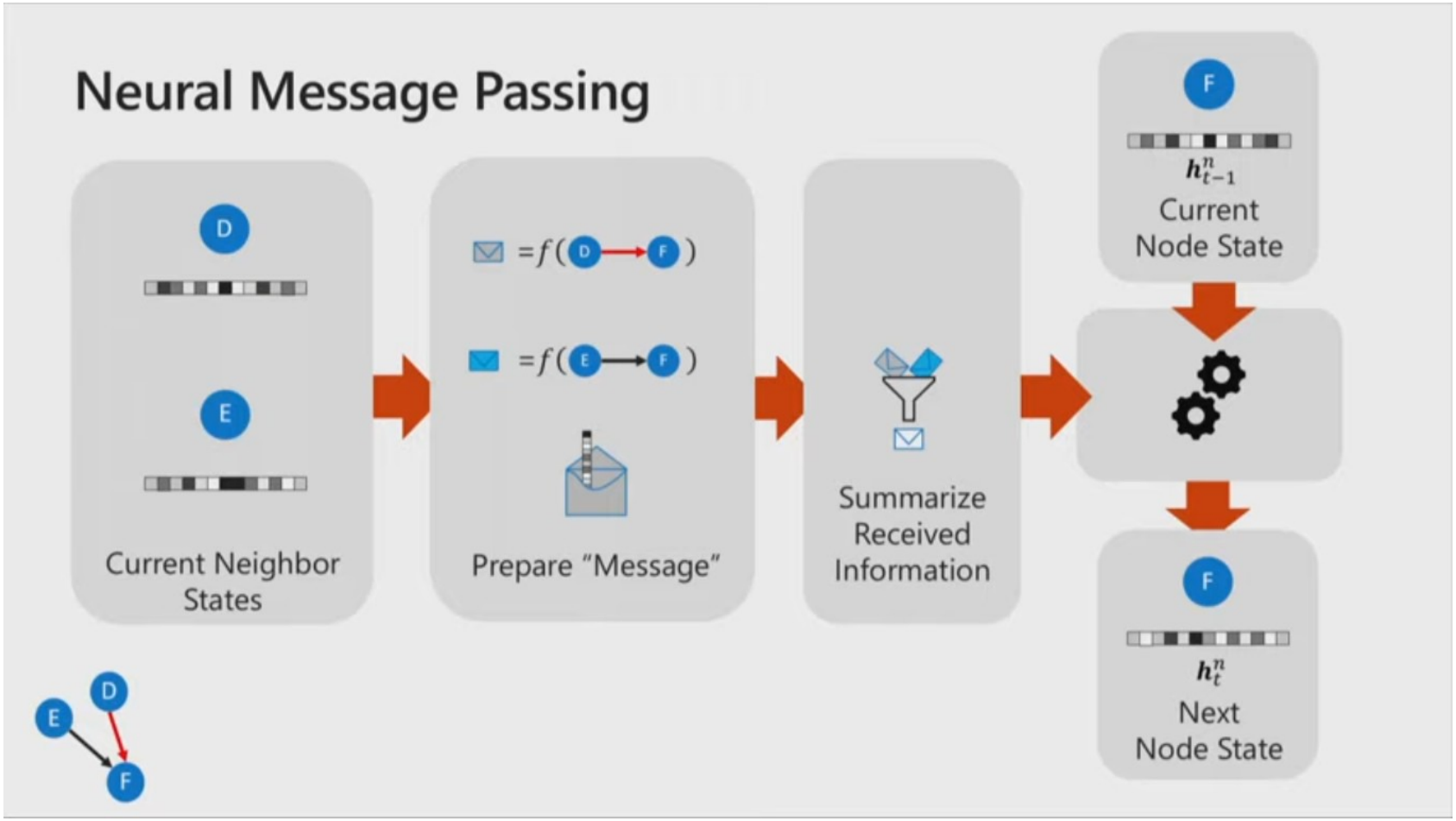


Aggregate (Neighbors)

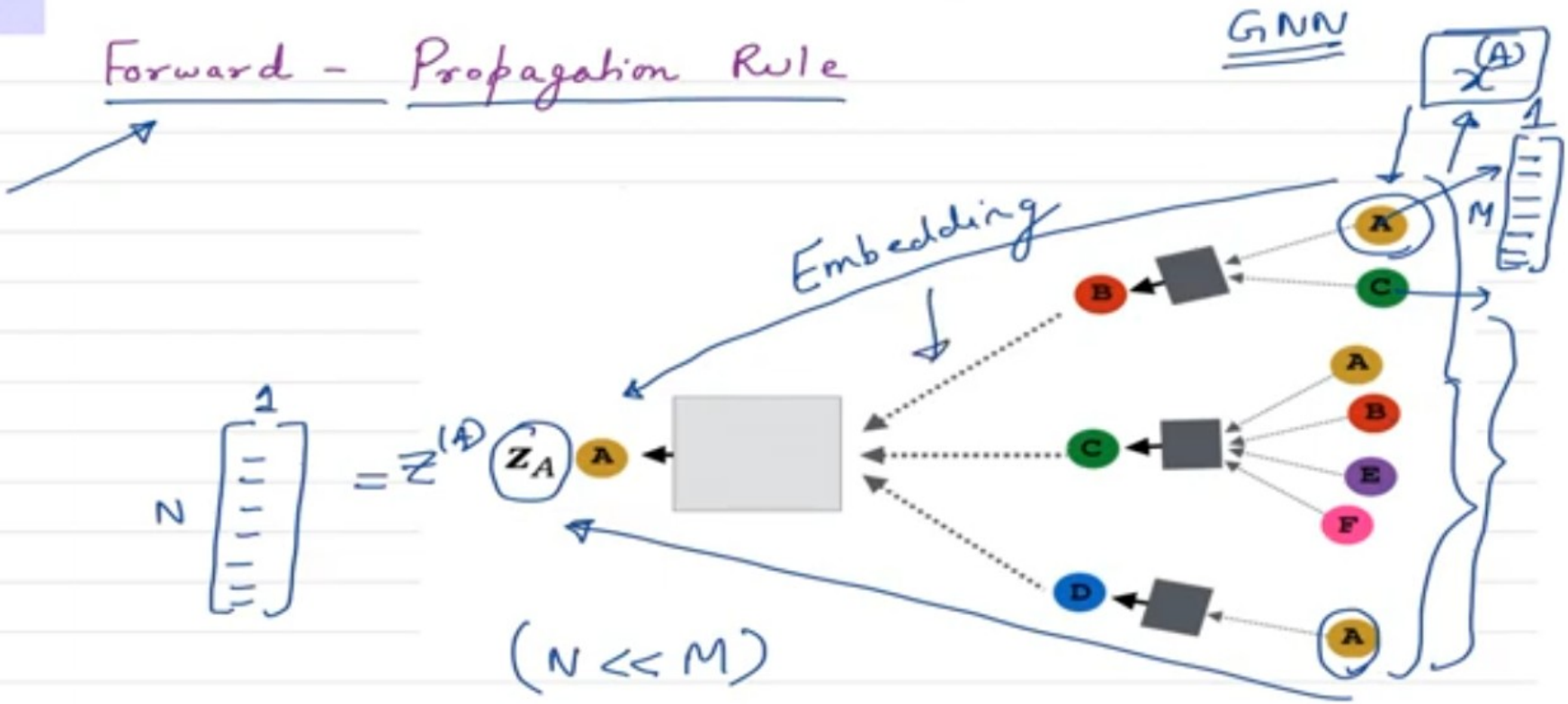


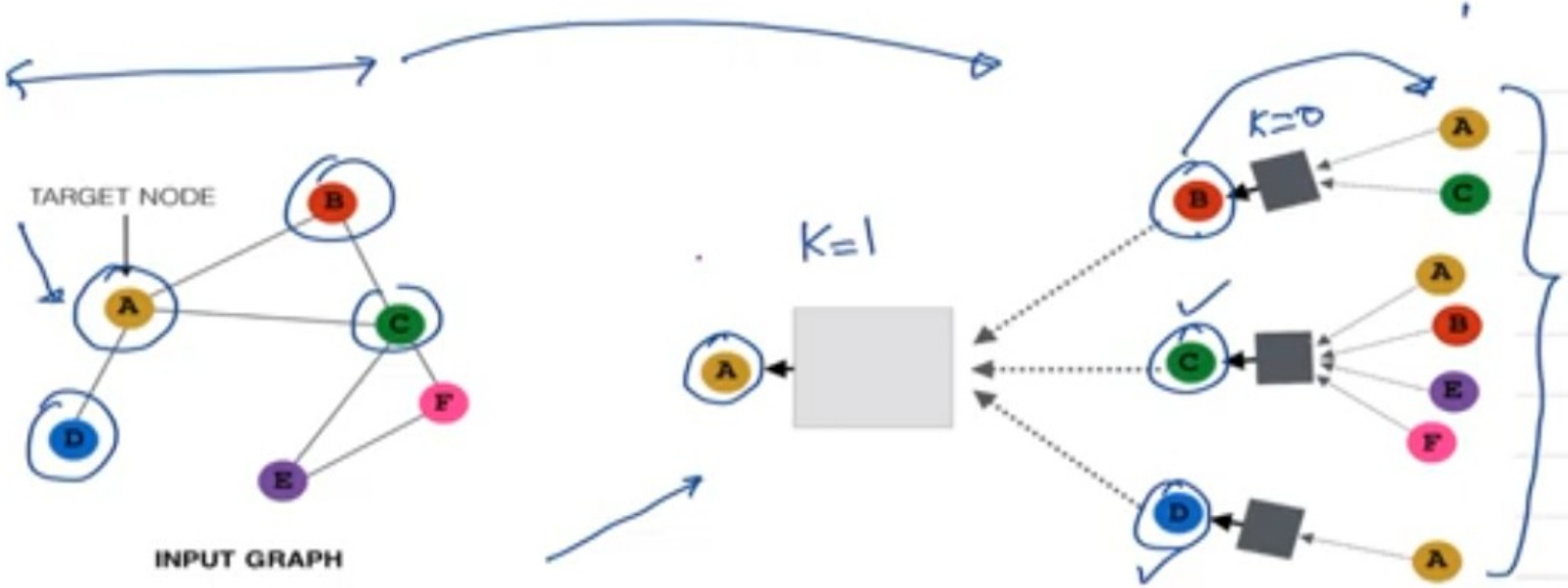
Every node has a computational graph




Neural Message Passing
D
E
Current Neighbor States
$=f($ D → F $)$
$=f($ E → F $)$
Prepare "Message"
Summarize Received Information
F
h_{t-1}^n
Current Node State
F
h_t^n
Next Node State
D
E
F

Forward - Propagation Rule

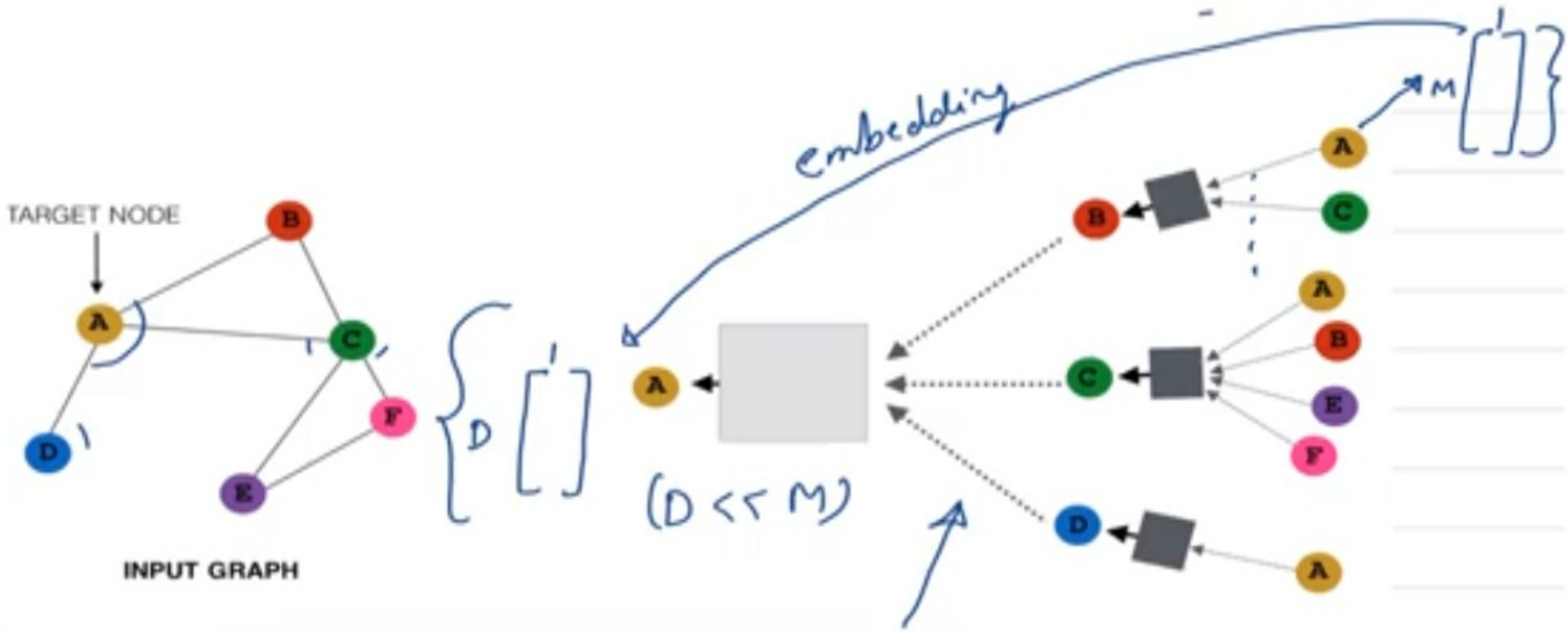




$$h_v^0 = x_v$$

$$h_v^k = \sigma\left[W_k \sum_{u \in N(v)} \left[\frac{h_u^{[k-1]}}{N(v)} \right] + B_k h_v^{[k-1]} \right] \qquad k = 1, \ldots K-1$$

$$z^{(A)} = h_v^{[K]}$$



$$h_v^0 = x_v$$

$$h_v^k = \sigma\left[W_k \sum_{u \in N(v)} \left[\frac{h_u^{[k-1]}}{N(v)} \right] + B_k h_v^{[k-1]} \right] \qquad k = 1, \ldots K-1$$

$$z^{(A)} = h_v^{[K]}$$

Graph $G = (V, E)$

MATRIX NOTATION

Spectral analysis

* Input feature matrix $(N \times M) = X$
* Adjacency matrix $(N \times N) = A$

Generalized Neural Network is represented as

$$H^{(l+1)} = f(H^{(l)}, A)$$

with $H^{(0)} = X$ & $H^{(L)} = Z$, L being number of layers.

MATRIX NOTATION

Graph $G = (V, E)$

* Input feature matrix $(N \times M) = X$
* Adjacency matrix $(N \times N) = (A)$

Spectral analysis

Generalized Neural Network is represented as

$$H^{(l+1)} = f(H^{(l)}, A)$$

with $H^{(0)} = \{X\}$ & $H^{(L)} = \{Z\}$, L being number of layers.

$K=0$

→ A simple layer wise propagation rule is

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

- σ : Relu { non linear activation unit }
- A : $N \times N$
- $H^{(l)}$: $N \times M$ for $l=0$
- $W^{(l)}$: $M \times P$

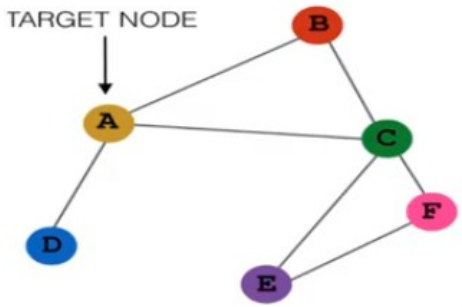
where $(P \ll N)$ for dim. reduction needed for Node embedding.

Limitations :

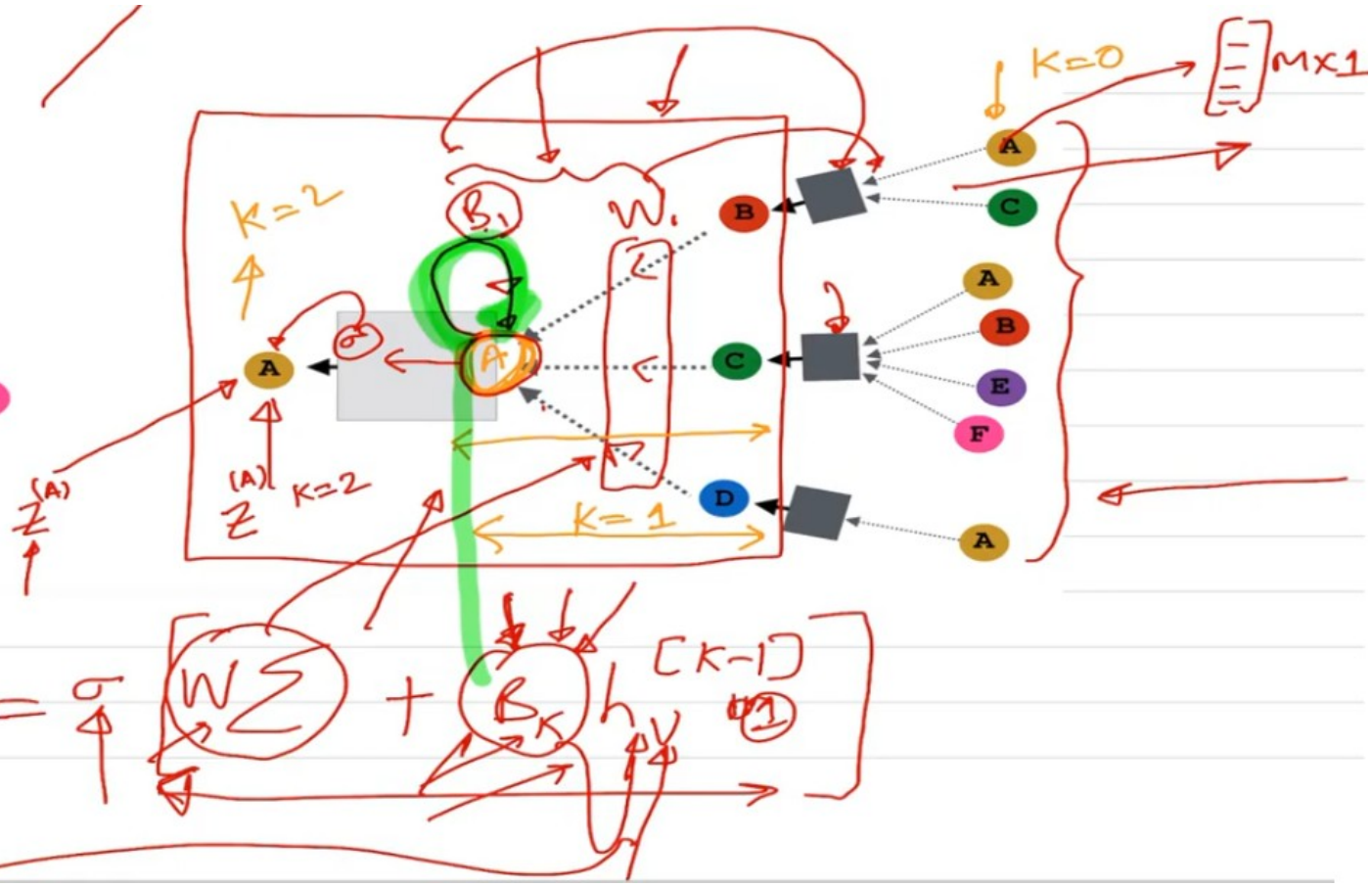
① No self loops in adjacency Matrix. Fixed by adding I

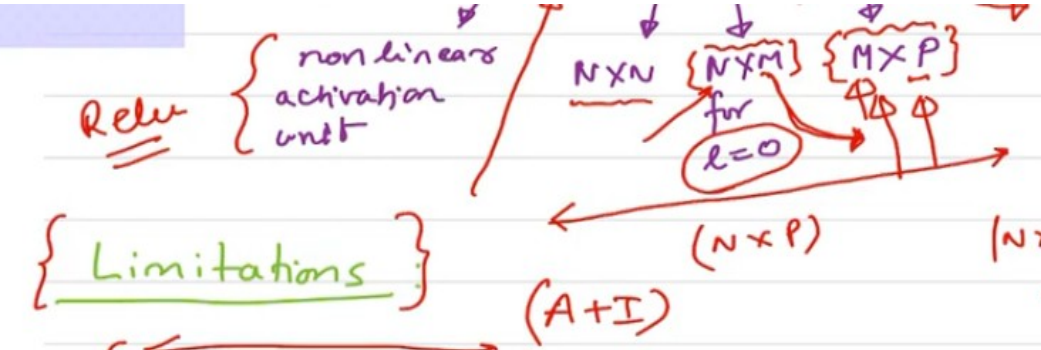
② A is not normalized. results → in change of feature vectors → explodes gradient & optimizer sensitive to features

Normalize A with Degree matrix ⇒ $D^{-1}A$



INPUT GRAPH





(1) {No self loops} in adjacency Matrix. Fixed by adding (I)

(2) A is not normalized. results in change of feature vectors
↓
explodes gradients & optimizer sensitive to feature scaling

Normalize A with Degree matrix $\Rightarrow D^{-1}A$

$D^{-1}A$ → Corresponds to taking average of neighboring node features.

① {No self loops} in adjacency Matrix. Fixed by adding ($\hat{A} = A + I$)

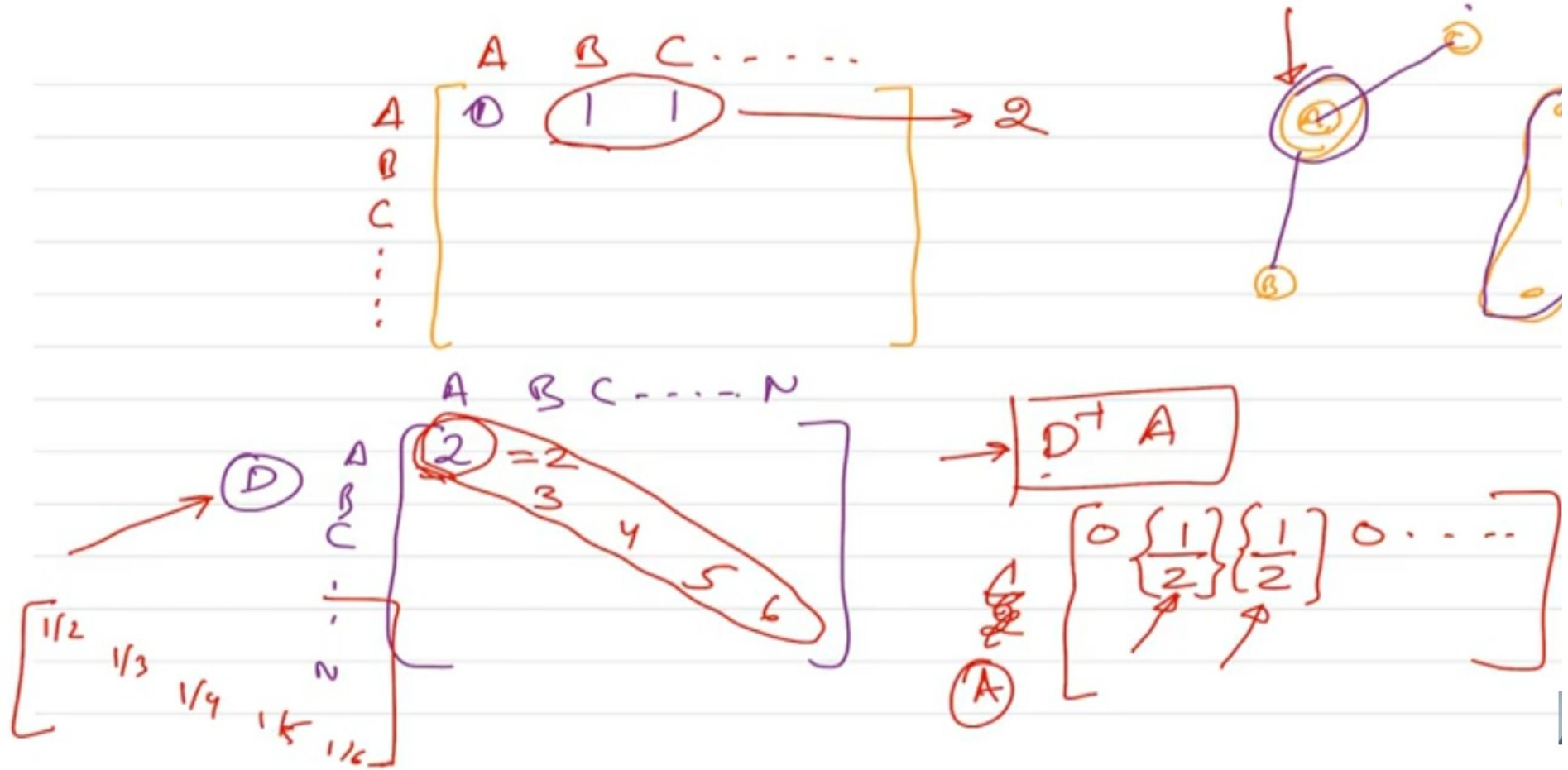
② A is not normalized. results → in change of feature vectors

Normalize A with Degree matrix ⇒ $D^{-1}A$ → explodes gradients & optimizer sensitive to feature scaling

$D^{-1}A$ → Corresponds to taking average of neighboring node features.

Paper use Symmetric normalization $D^{-1/2} A D^{-1/2}$. With this new rule is

$$H^{(l+1)} = \sigma\left(D^{-1/2} \hat{A} D^{-1/2} H^{(l)} W^{(l)} \right)$$


A B C . . . . .
A
B
C
1 1
2
A B C . . . . N
D
2 =2
3
4
5
6
1/2
1/3
1/4
1/5
1/6
N
D⁻¹ A
0 1/2 1/2 0 . . . .
A

$\{D^{-1} A\}$ → Corresponds to {taking average of neighboring node features}.

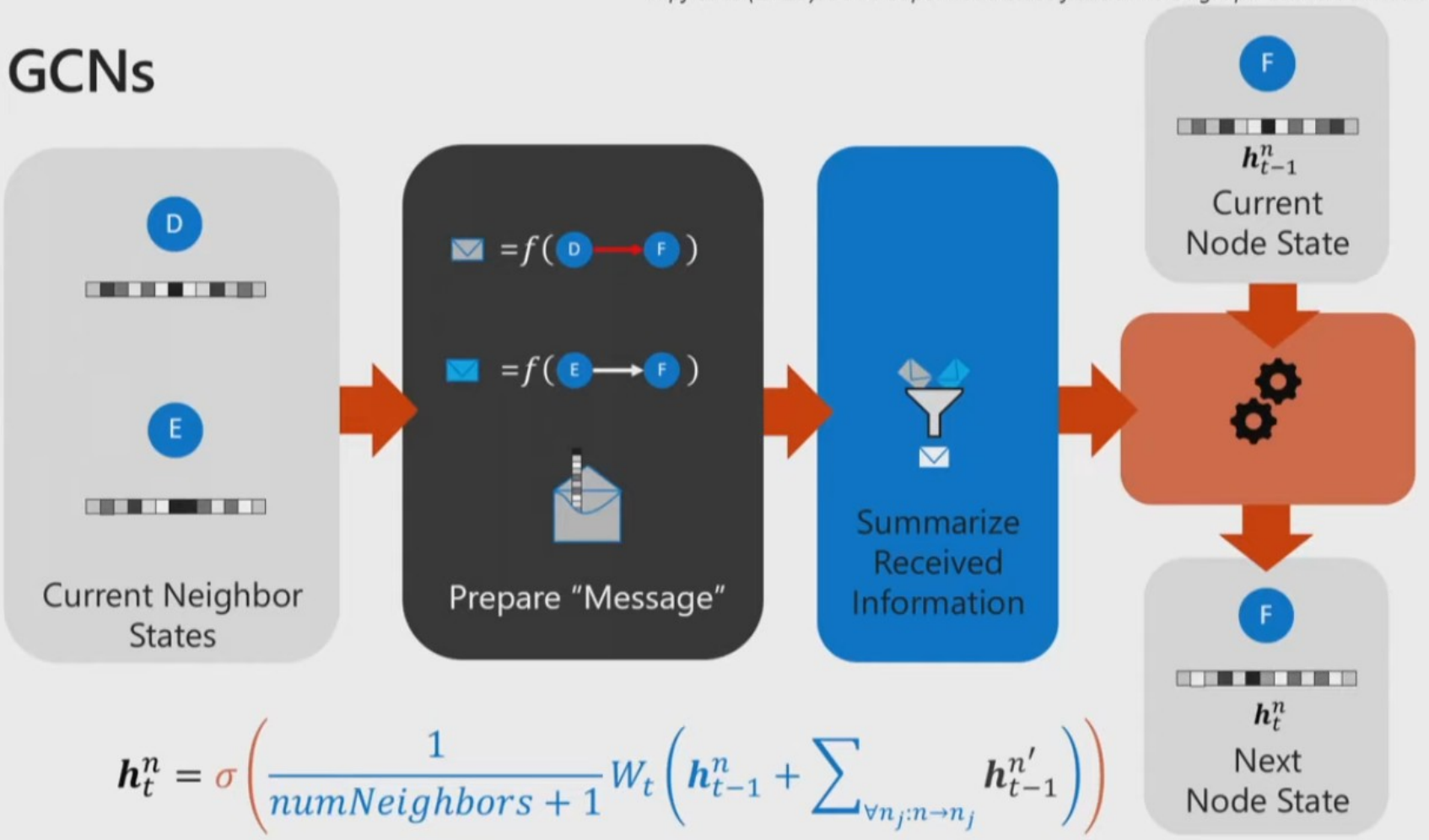
Paper use Symmetric normalization $D^{-1/2} A D^{-1/2}$. With this new rule is

$$H^{(l+1)} = \sigma\left(D^{-1/2} \hat{A} D^{-1/2} H^{(l)} W^{(l)} \right)$$

where $\hat{A} = A + I$, D is degree matrix of $\hat{A}$

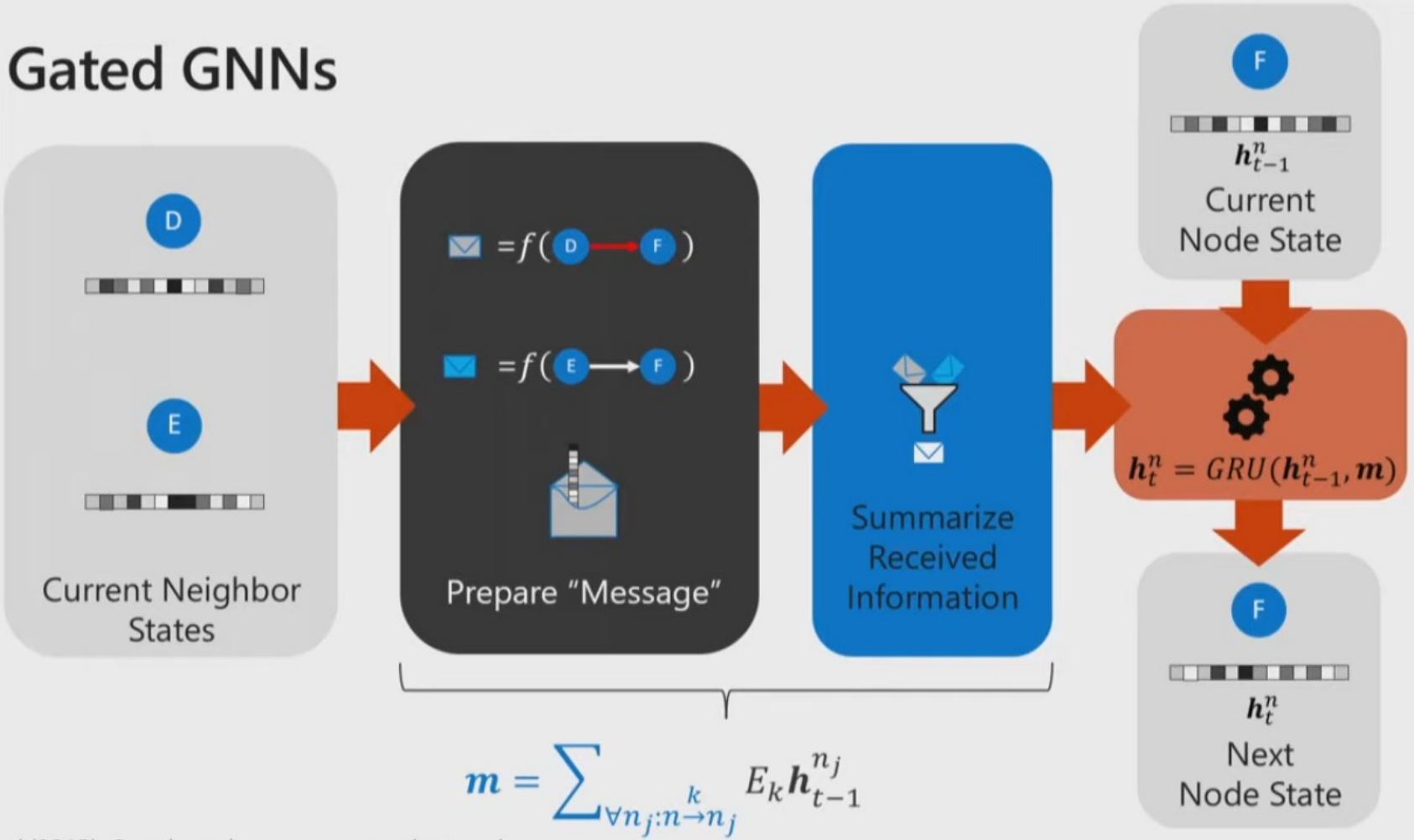
Kipf et al (2016). Semi-supervised classification with graph convolutional networks.

GCNs



$$h_t^n = \sigma\left(\frac{1}{numNeighbors + 1} W_t \left(h_{t-1}^n + \sum_{\forall n_j : n \to n_j} h_{t-1}^{n'}\right)\right)$$

Gated GNNs



Li et al (2015). Gated graph sequence neural networks.

Expressing GGNNs as Matrix Operations

GGNN as Matrix Operation

Node States

$$H_t = \begin{bmatrix} \boldsymbol{h}_t^{n_0} \\ \vdots \\ \boldsymbol{h}_t^{n_K} \end{bmatrix} \quad \texttt{(num\_nodes x D)}$$

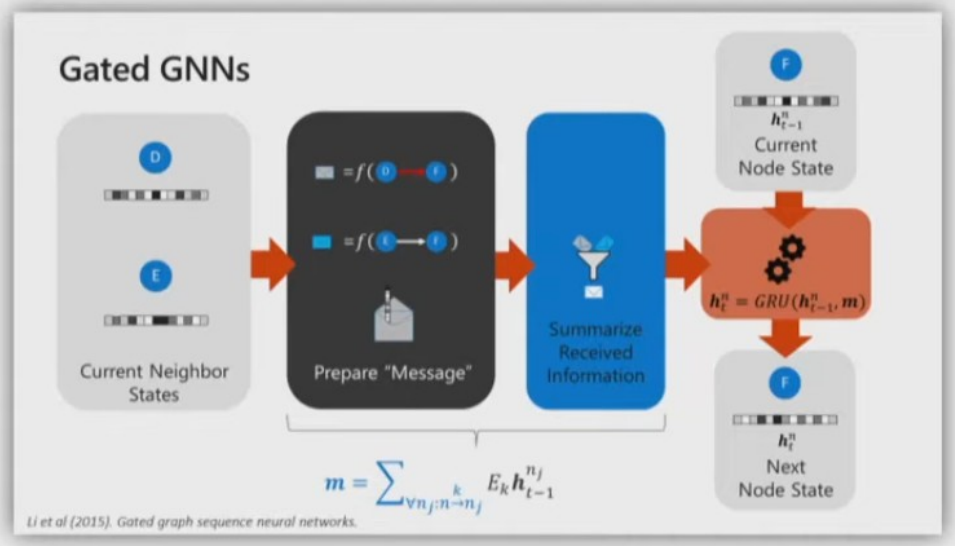
Messages to-be sent

$$M_t^k = E_k H_t \quad \texttt{(num\_nodes x M)}$$

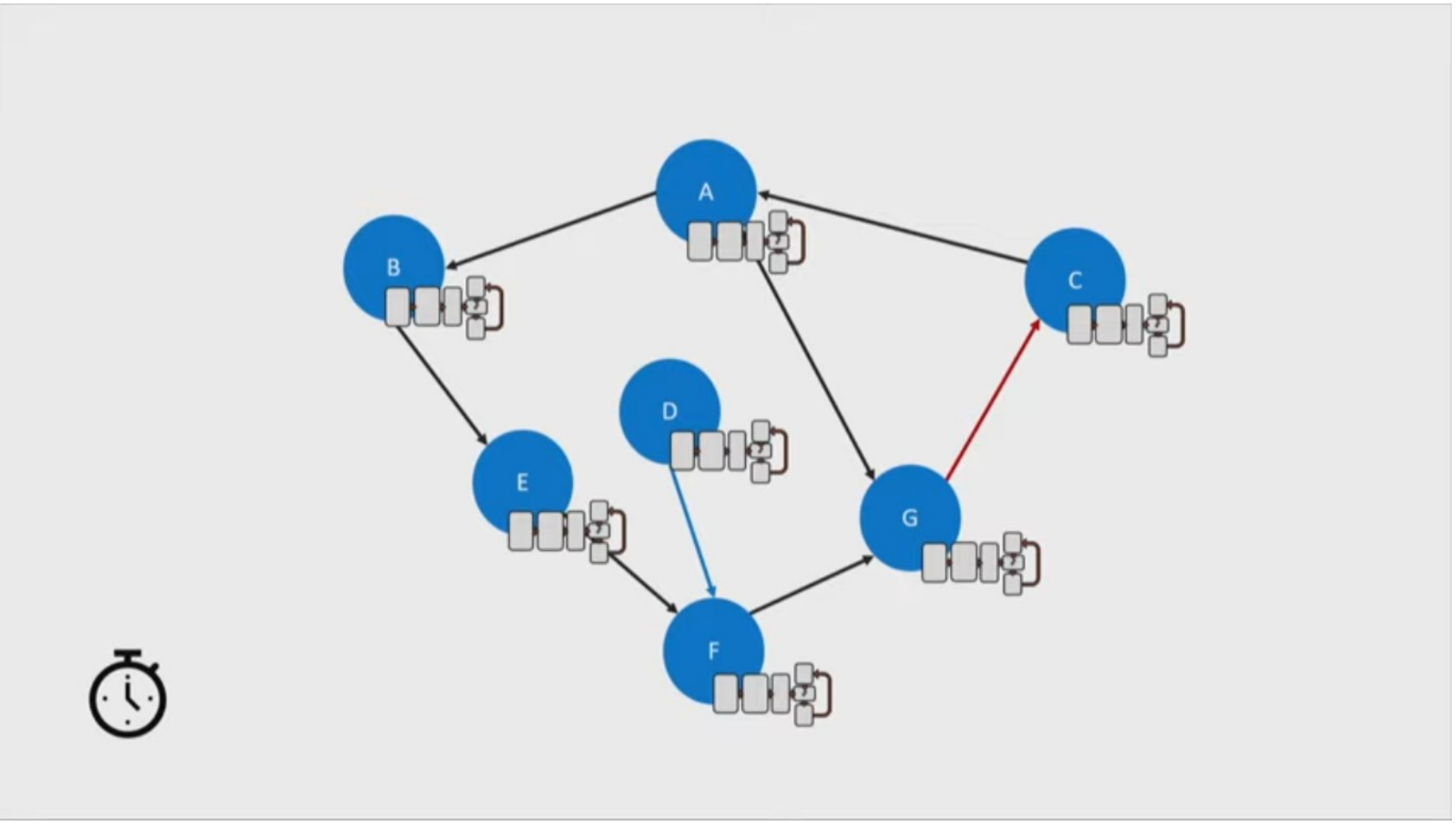
Received Messages

$$R_t = \sum_k A M_t^k \quad \texttt{(num\_nodes x M)}$$

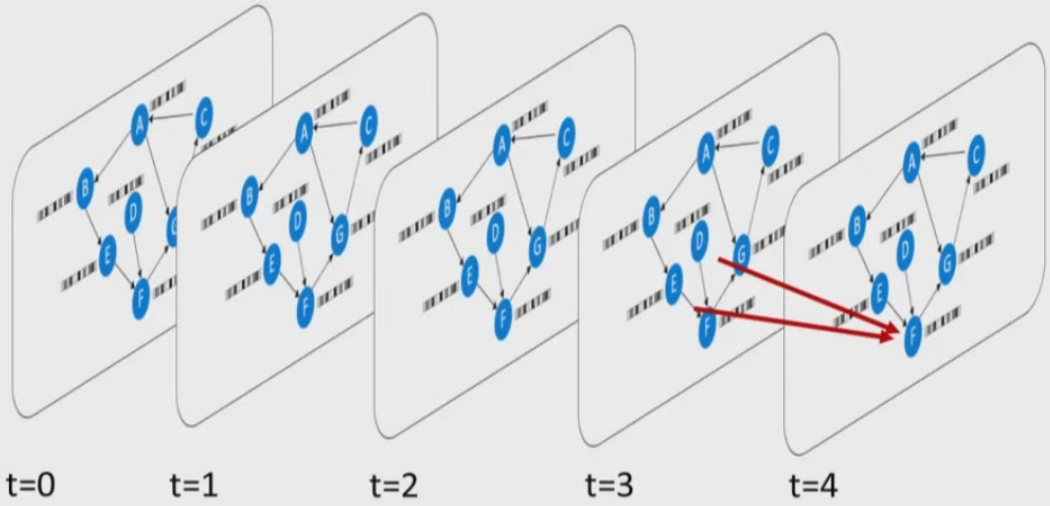
Update $H_{t+1} = GRU(H_t, R_t)$



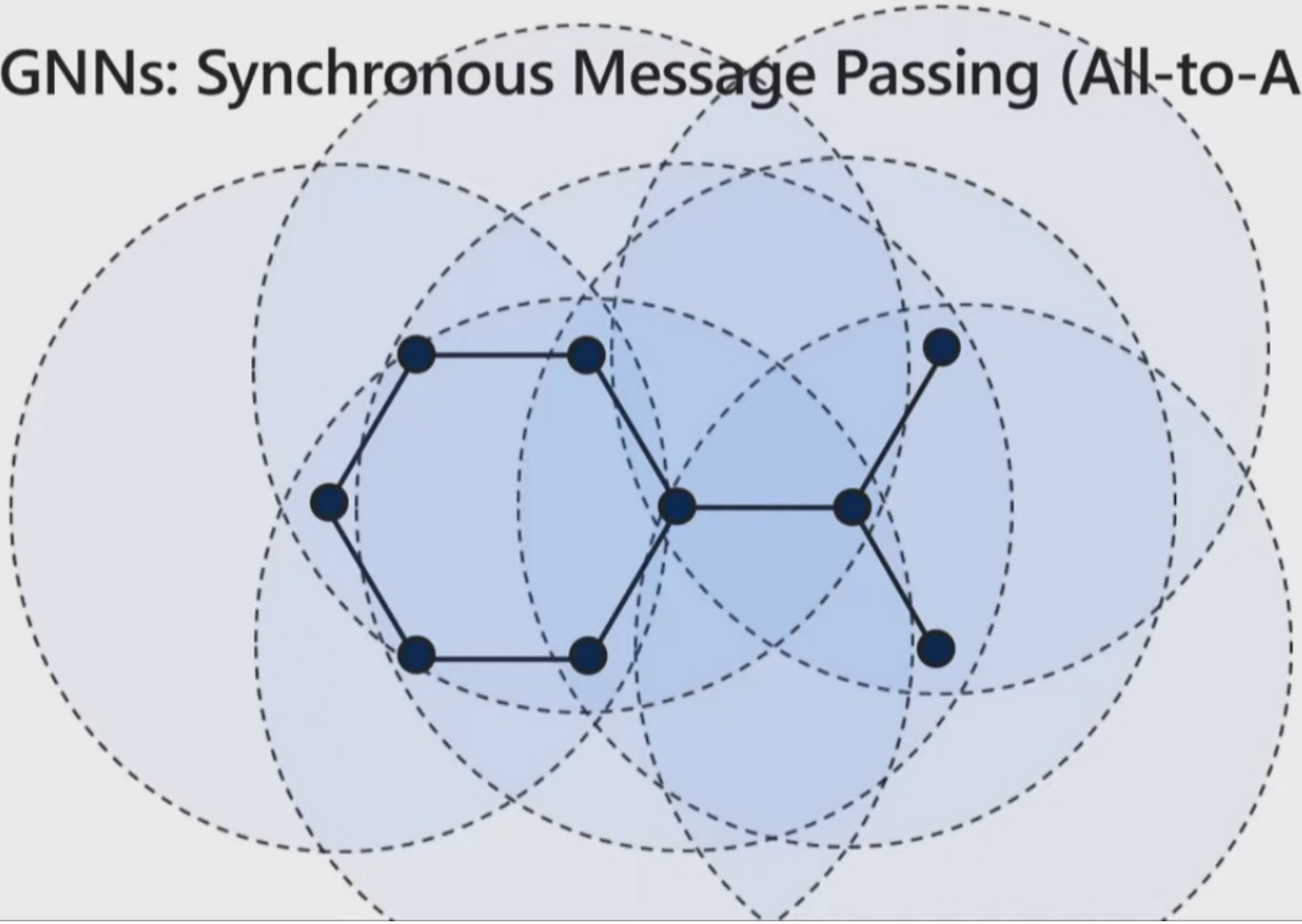
Trick 1: Backwards Edges


A
B
C
D
E
F
G

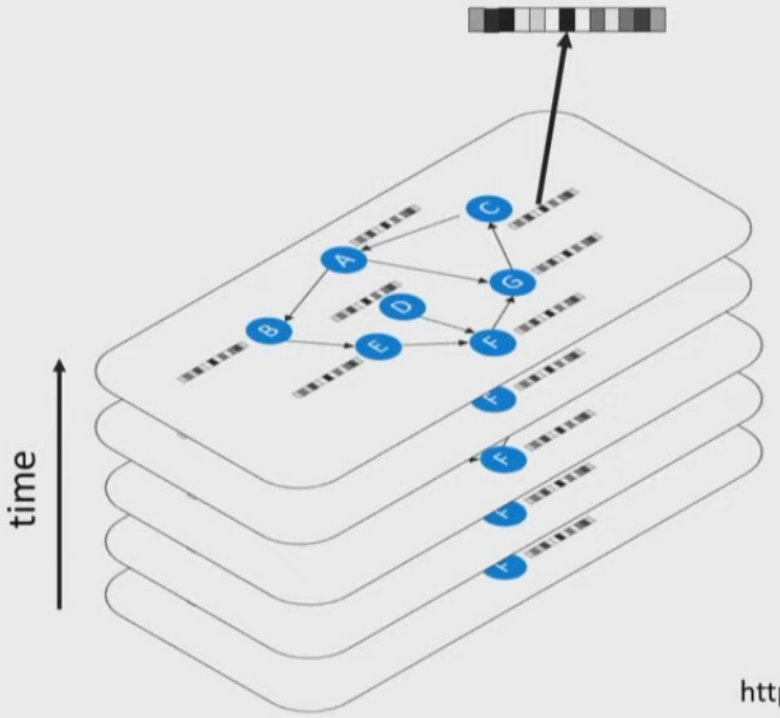
Graph Neural Networks: Message Passing



GNNs: Synchronous Message Passing (All-to-All)



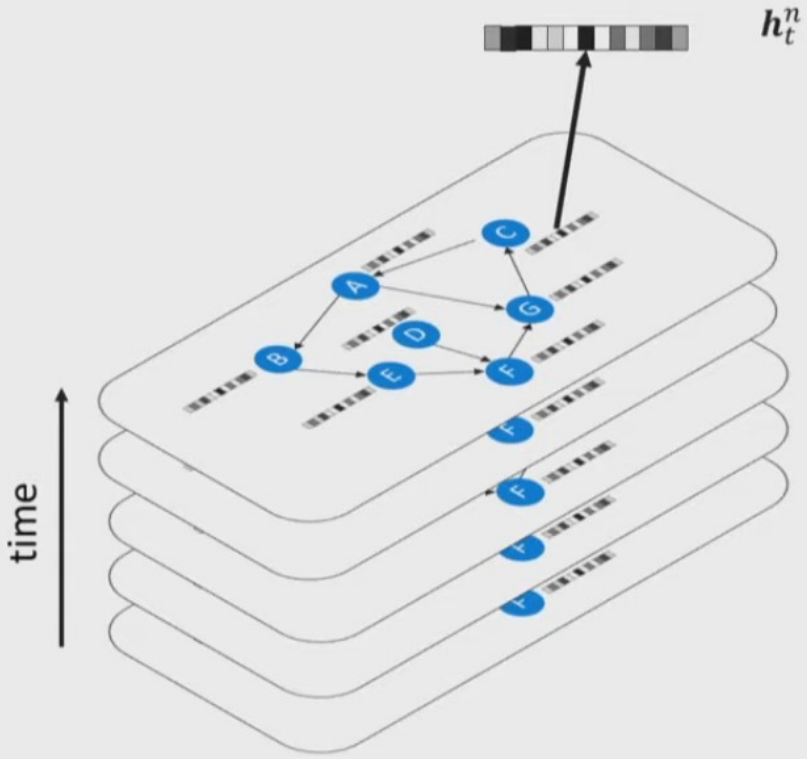
Graph Neural Networks: Output



- node selection
- node classification
- graph classification

https://github.com/microsoft/tf-gnn-samples/

Example: Node [Binary] Classification



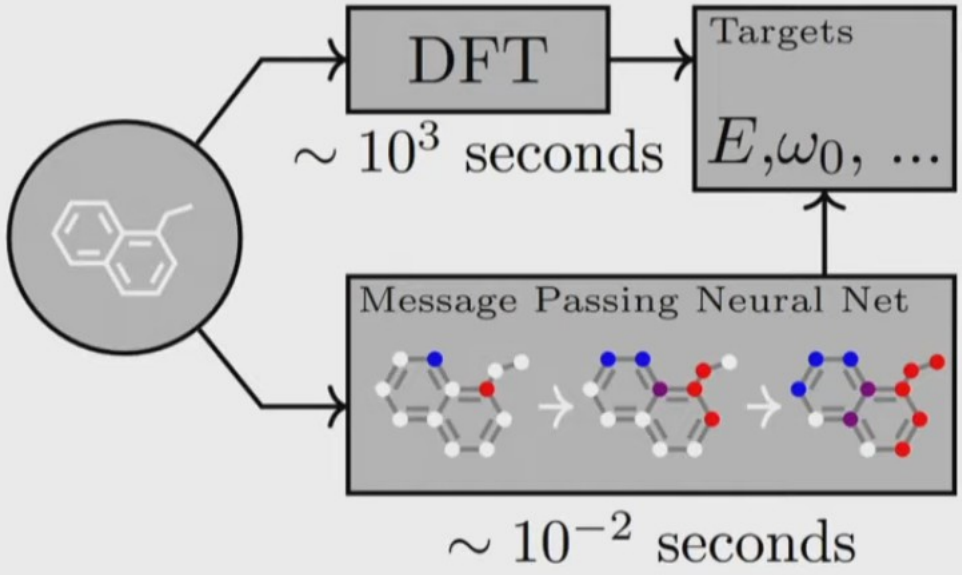
$$x_n = \sigma\left(\boldsymbol{w}^T \boldsymbol{h}_t^n + b\right)$$

Binary cross entropy

$$\mathcal{L} = y_n \cdot \log x_n + (1 - y_n) \log(1 - x_n)$$

Two Sample Applications

...with graph neural networks.



Gilmer, Justin, et al. "Neural message passing for quantum chemistry." 2017.

Variable Misuse Task

```
var clazz=classTypes["Root"].Single() as JsonCodeGenerator.ClassType;
Assert.NotNull(clazz);

var first=classTypes["RecClass"].Single() as JsonCodeGenerator.ClassType;
Assert.NotNull(      );

Assert.Equal("string", first.Properties["Name"].Name);
Assert.False(clazz.Properties["Name"].IsArray);
```

Possible type-correct options: `clazz, first`

Variable Misuse Task

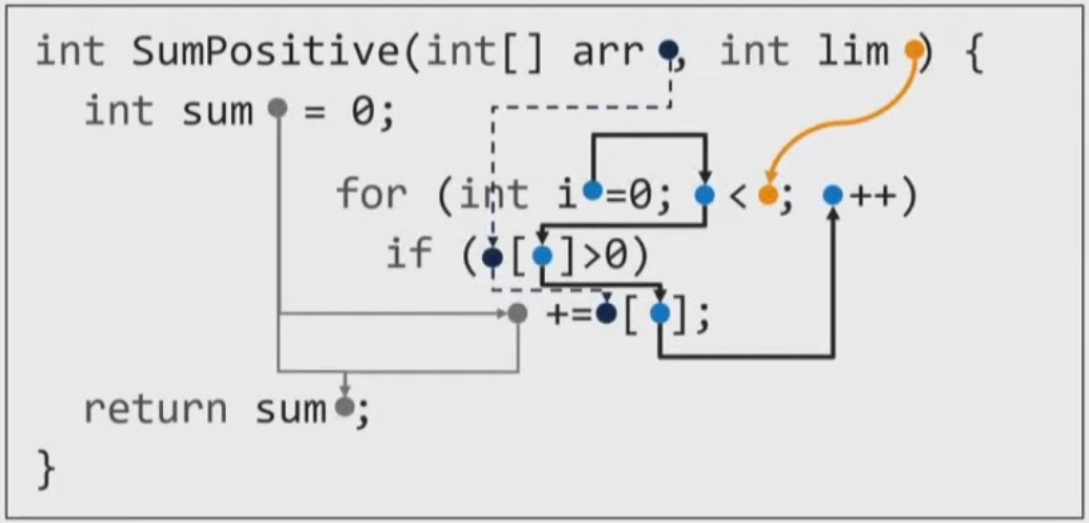
```
var clazz=classTypes["Root"].Single() as JsonCodeGenerator.ClassType;
Assert.NotNull(clazz);

var first=classTypes["RecClass"].Single() as JsonCodeGenerator.ClassType;
Assert.NotNull( clazz );

Assert.Equal("string", first.Properties["Name"].Name);
Assert.False(clazz.Properties["Name"].IsArray);
```

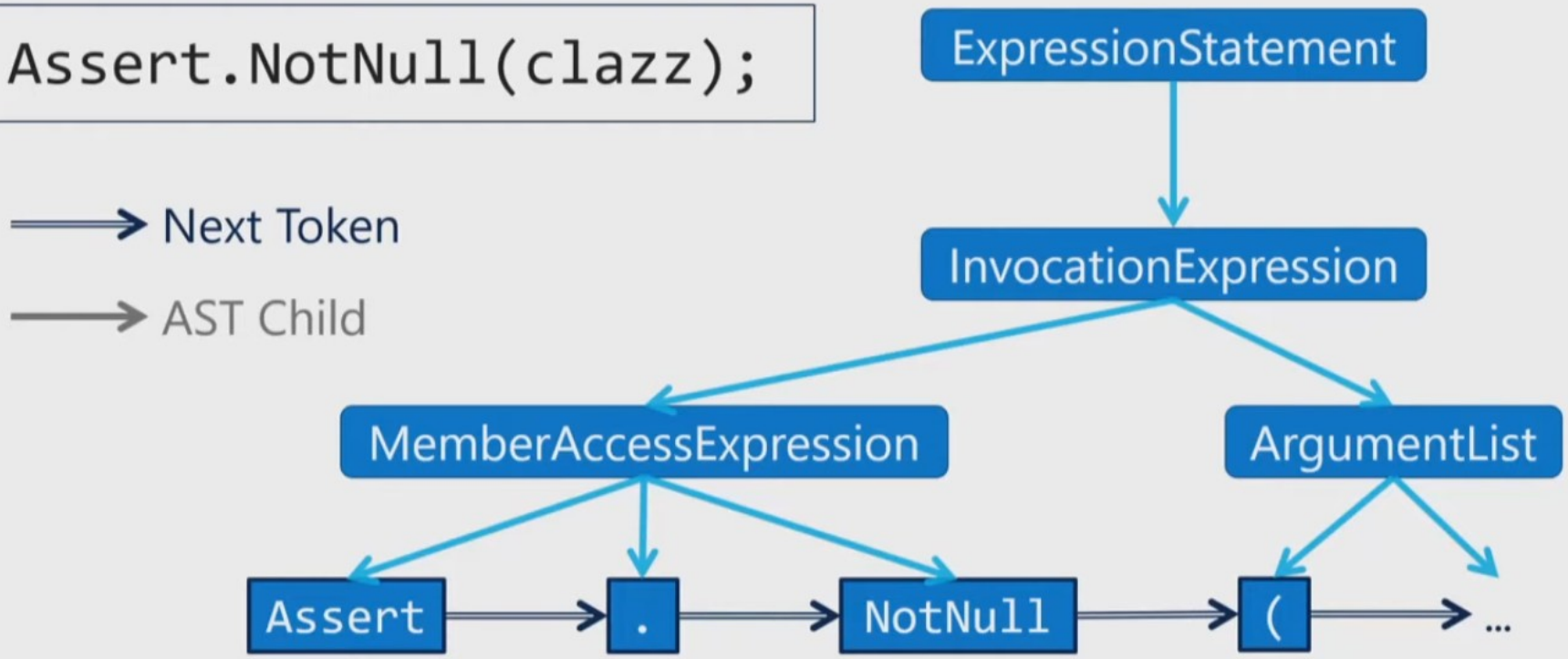
Possible type-correct options: `clazz, first`

Programs as Graphs



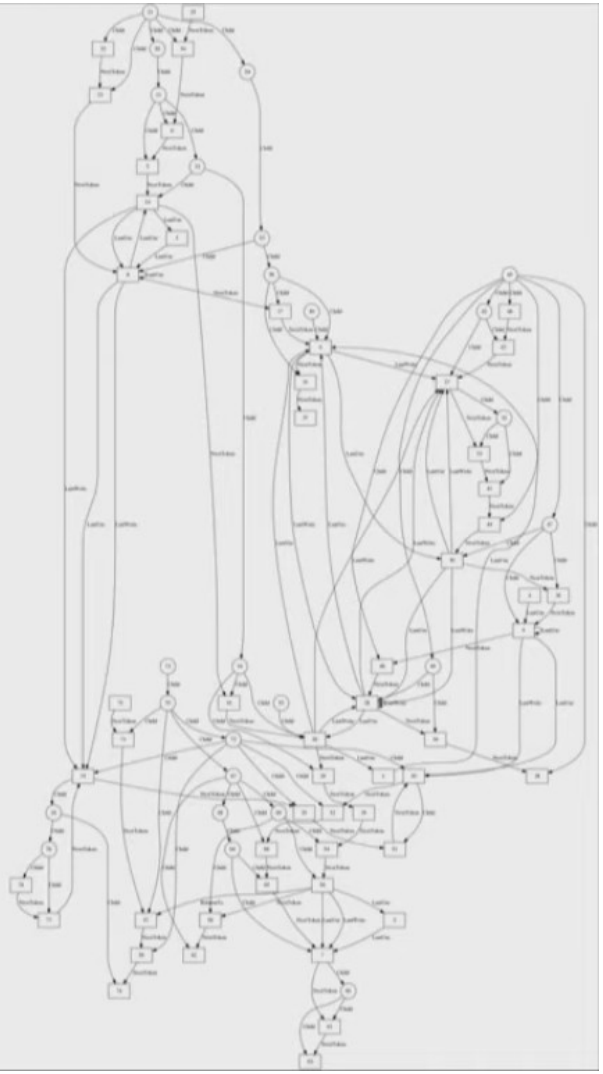
Programs as Graphs: Syntax

```
Assert.NotNull(clazz);
```



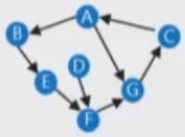
Programs as Graphs

```
int SumPositive(int[] arr, int lim) {
  int sum = 0;
  for (int i=0; i<; ++)
    if ([]>0)
      +=[];
  return sum;
}
```



~900 nodes/graph ~8k edges/graph

Graph Representation for Variable Misuse



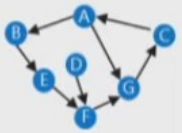

```
var clazz=classTypes["Root"].Single() as JsonCodeGenerator.ClassType;
Assert.NotNull(clazz);

var first=classTypes["RecClass"].Single() as JsonCodeGenerator.ClassType;
Assert.NotNull( SLOT );
Assert.Equal("string", first.Properties["Name"].Name);
Assert.False(clazz.Properties["Name"].IsArray);
```

first

clazz

Graph Representation for Variable Misuse




```
var clazz=classTypes["Root"].Single() as JsonCodeGenerator.ClassType;
Assert.NotNull(clazz);

var first=classTypes["RecClass"].Single() as JsonCodeGenerator.ClassType;
Assert.NotNull( SLOT );

Assert.Equal("string", first.Properties["Name"].Name);
Assert.False(clazz.Properties["Name"].IsArray);
```

first

clazz

Goal: make the representation of SLOT as close as possible to the representation of the correct candidate node

$$f\left(\boldsymbol{h}_T^{SLOT}, \boldsymbol{h}_T^{first}\right) \gg f\left(\boldsymbol{h}_T^{SLOT}, \boldsymbol{h}_T^{clazz}\right)$$

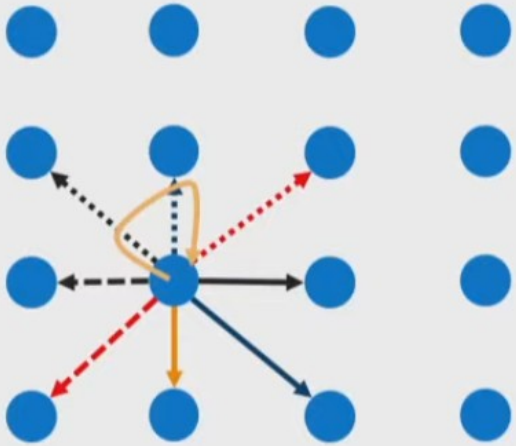
```
bool TryFindGlobalDirectivesFile(string baseDirectory, string fullPath, out string path) {
   baseDirectory = baseDirectory.TrimEnd(Path.DirectorySeparatorChar);
   var directivesDirectory = Path.GetDirectoryName(fullPath)
                                 .TrimEnd(Path.DirectorySeparatorChar);
   while (directivesDirectory != null && directivesDirectory.Length >= baseDirectory.Length) {
      path = Path.Combine(directivesDirectory, GlobalDirectivesFileName);
      if (File.Exists(path)) return true;

      directivesDirectory = Path.GetDirectoryName(directivesDirectory)
                                .TrimEnd(Path.DirectorySeparatorChar);
   }
   path = null;
   return false;
}
```

What the model sees...

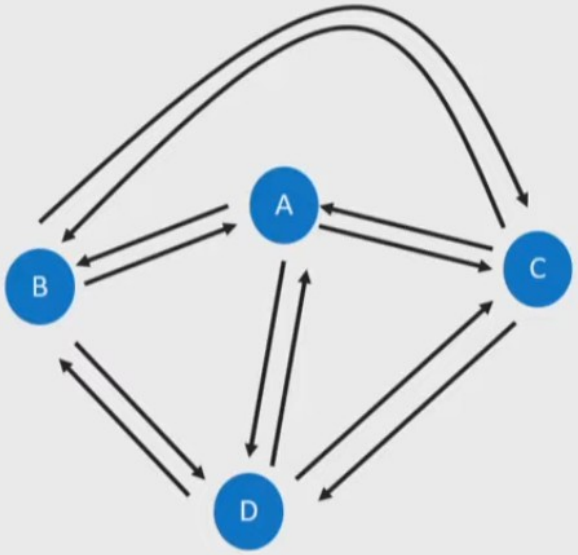
Other Models as Special Cases of GNNs

Special Case 1: Convolutions (CNN)



Special Case 2: “Deep Sets”

Special Case 2: “Deep Sets”



Stanford material

- http://snap.stanford.edu/class/cs224w-2020/