PART - 2

Graph Neural Networks

GNN

@avkashchauhan

Tutorial Agenda

PART 1

- Fundamentals of Graph
- Mathematics of Graph
- Introduction to NetworkX Python Package
- Graph Programming with NetworkX
- Introduction to GNN
- Relationship between GNN and CNN
- Introduction to PyG (pytorch_geometric)
- Graph Visualization Tools
  - Gephi
  - yEd
- Various Graph Data Manipulation

PART 2

- Graph representations
  - Adjacency Matrix
  - Feature Matrix
  - Incidence Matrix
  - Degree Matrix
  - Laplacian Matrix
  - Bag of Nodes
- Node Embedding and Node Embedding Space
- Applying Convolution to Graph similar to Image
- Message Passing
- Understanding Graph Datasets available in PyG
- Node Classification using MLP & GNN
- NetworkX and tSNE visualization of Graphs
- GNN Explainer

Recap

- Graph -
  - Data Type
    - Structured Data Types
  - Components
    - Nodes or Vertices
    - Edges
  - Definition:
    - G = {V,E)
  - Types:
    - Directed (i.e. Twitter) and Indirected (i.e. Facebook Friend Connection)
  - Labels
    - Both nodes and edges can have text and numeric labels,
    - Edges can have multiple keys and labels
  - Weight
    - If nodes have weight then the graph is Weighted graph
- **Node Features** are the property of the node
  - **Node** - City, **Label**: San Francisco, **Features**: population, male/female population, area, schools, bridges etc
  - **Node** - Person, **Label**: Avkash, **Features**: Age, sex, income, marital status, spouse info etc
  - Note: Do not mix the **node features** with **node** or **edge** label

Graph Neural Networks: Part 2

- Graph representations
  - Adjacency Matrix
  - Feature Matrix
  - Incidence Matrix
  - Degree Matrix
  - Laplacian Matrix
  - Bag of Nodes
- Node Embedding and Node Embedding Space
- Applying Convolution to Graph similar to Image
- Message Passing
- Understanding Graph Datasets available in PyG
- Node Classification using MLP & GNN
- NetworkX and tSNE visualization of Graphs
- GNN Explainer

Graph Representation

Adjacency Matrix (A)

- Square Matrix (A = NxN)

Incidence (Relation Matrix) Matrix (I)

- Nodes n with edges m will have (I = nxm)

Degree Matrix (D)

- Diagonal Matrix (D)

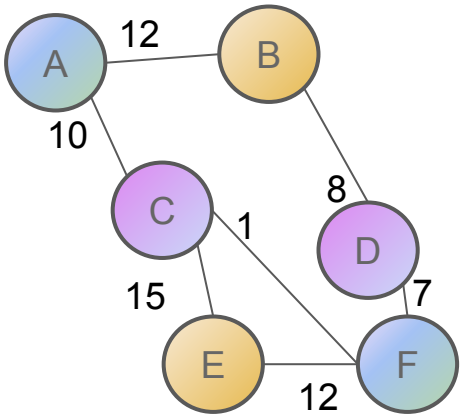
Laplacian Matrix:

- L = D - A

Bag of Nodes

- Aggregate node level features using some mathematical approach i.e. taking the mean of the node degrees or histogram of the edge connections

Adjacency Matrix

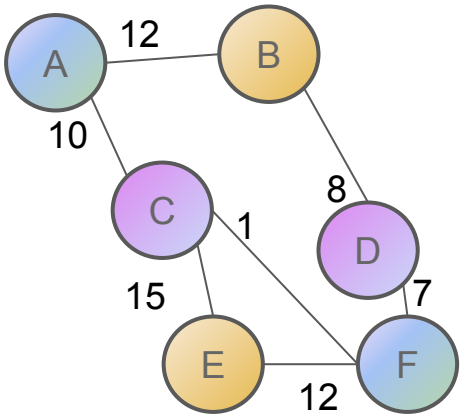


- Adjacency matrix a 2D square matrix
- Each node in the graph has an entry in both dimensions.
- Unweighted graph as T/F or 1/0 values (Binary Value Adjacency Matrix)
- Weighted graph as weights, no weights means -1 (Weighted Adjacency Matrix)

Representation:

- $A = N \, x \, N$

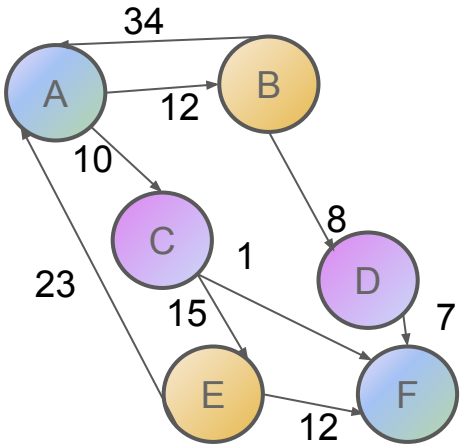
Adjacency Matrix - Weighted & Undirected



	A	B	C	D	E	F
A	-1	12	10	-1	-1	-1
B	-1	-1	-1	8	-1	-1
C	-1	-1	-1	-1	15	1
D	-1	-1	-1	-1	-1	7
E	-1	-1	-1	-1	-1	12
F	-1	-1	-1	-1	-1	-1

$A = 6 \; x \; 6$

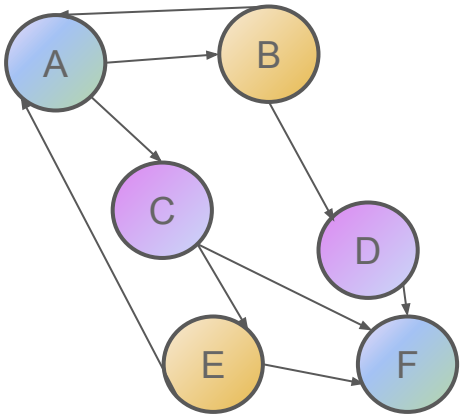
Adjacency Matrix - Weighted & Directed



	A	B	C	D	E	F
A	-1	12	10	-1	-1	-1
B	34	-1	-1	8	-1	-1
C	-1	-1	-1	-1	15	1
D	-1	-1	-1	-1	-1	7
E	23	-1	-1	-1	-1	12
F	-1	-1	-1	-1	-1	-1

$$A = 6 \; x \; 6$$

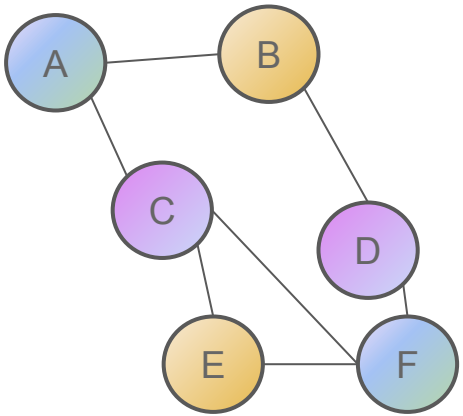
Adjacency Matrix - Un-Weighted & Directed



	A	B	C	D	E	F
A	F	T	T	F	F	F
B	T	F	F	T	F	F
C	F	F	F	F	T	T
D	F	F	F	F	F	T
E	T	F	F	F	F	T
F	F	F	F	F	F	F

$$A = 6 \; x \; 6$$

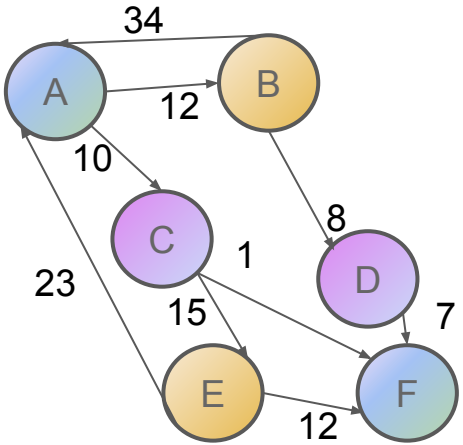
Adjacency Matrix - Unweighted & Undirected



	A	B	C	D	E	F
A	F	T	T	F	F	F
B	F	F	F	T	F	F
C	F	F	F	F	T	T
D	F	F	F	F	F	T
E	F	F	F	F	F	T
F	F	F	F	F	F	F

$$A = 6 \; x \; 6$$

Adjacency Matrix - Weighted & Directed

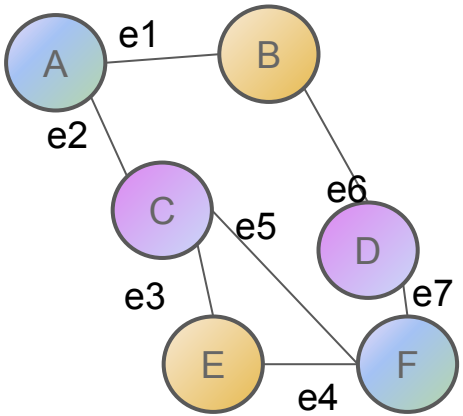


	A	B	C	D	E	F
A	-1	12	10	-1	-1	-1
B	34	-1	-1	8	-1	-1
C	-1	-1	-1	-1	15	1
D	-1	-1	-1	-1	-1	7
E	23	-1	-1	-1	-1	12
F	-1	-1	-1	-1	-1	-1

A_{AB}	A-B
A_{AC}	A-C
A_{AF}	A-C-E-F
A_{CD}	C-E-A-B-D
A_{BC}	X

$$A = 6 \; x \; 6$$

Incidence Matrix - Nodes N/Edge M - **Undirected** Graph

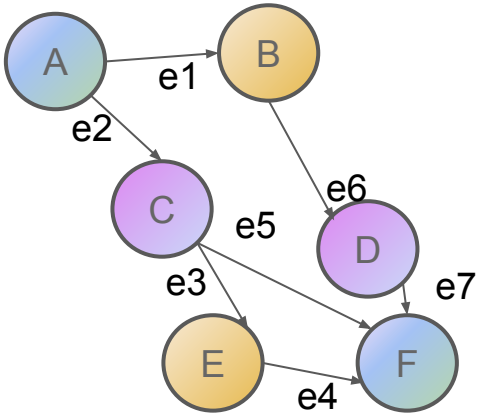


	e1	e2	e3	e4	e5	e6	e7
A	1	1	0	0	0	0	0
B	1	0	0	0	0	1	0
C	0	1	1	0	1	0	0
D	0	0	0	0	0	1	1
E	0	0	1	1	0	0	0
F	0	0	0	1	1	0	1

If Node and Edge are connected then value is 1 otherwise the value is 0.
Nodes are represented as **ROWS** and edges are as **COLUMNS**

$$I = 6 \; x \; 7$$

Incidence Matrix - Nodes N/Edge M - **Directed** Graph



	e1	e2	e3	e4	e5	e6	e7
A	-1	-1	0	0	0	0	0
B	1	0	0	0	0	-1	0
C	0	1	-1	0	-1	0	0
D	0	0	0	0	0	1	-1
E	0	0	1	-1	0	0	0
F	0	0	0	1	1	0	1

$$I = 6 \; x \; 7$$

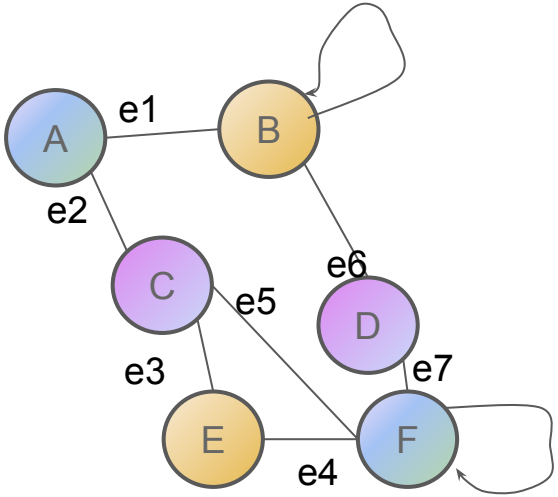
If Node and Edge are connected then value -1/1, otherwise the value is 0.

- If Node to **outward** direction connection then the value is -1
- If node has **inwards** direction connection then the value is 1

Nodes are represented as **ROWS** and edges are as **COLUMNS**

Degree Matrix - **Directed/Undirected** Graph (Same)

Diagonal Matrix



	A	B	C	D	E	F
A	2	0	0	0	0	0
B	0	4	0	0	0	0
C	0	0	3	0	0	0
D	0	0	0	2	0	0
E	0	0	0	0	2	0
F	0	0	0	0	0	5

Diagonal

Degree = Number of edges connected to each node

If Node has connection from edge then value is 1 otherwise the value is 0.
Nodes are represented as **ROWS** and **COLUMNS**

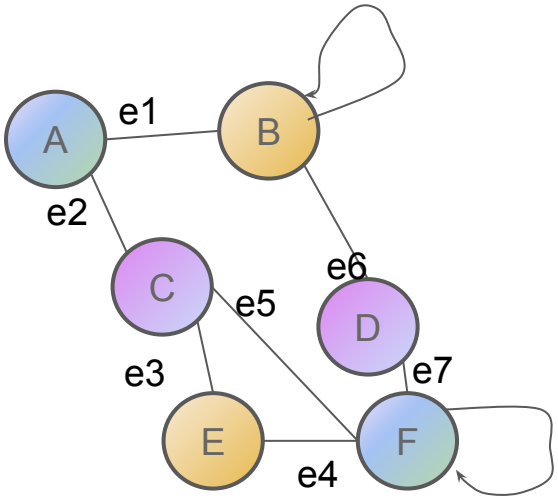
$$D = 6 \; x \; 6$$

Laplacian Matrix - Graph Laplacian Matrix

- Measure of the smoothness of the matrix
- How quickly it changes between the Adjacent Vertices
- L = Diagonal Matrix - Adjacency Matrix
- L = Number of Edges connected to Node - Adjacency Matrix
- L = {D - A}

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

Laplacian Matrix - Graph Laplacian Matrix



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

	A	B	C	D	E	F
A	2	0	0	0	0	0
B	0	4	0	0	0	0
C	0	0	3	0	0	0
D	0	0	0	2	0	0
E	0	0	0	0	2	0
F	0	0	0	0	0	5

D

	A	B	C	D	E	F
A	0	1	1	0	0	0
B	1	2	0	1	0	0
C	1	0	0	0	1	1
D	0	1	0	0	0	1
E	0	0	1	0	0	1
F	0	0	1	1	1	2

A

Self Loop is considered as **2** nodes

Laplacian Matrix - Graph Laplacian Matrix

	A	B	C	D	E	F
A	2	0	0	0	0	0
B	0	4	0	0	0	0
C	0	0	3	0	0	0
D	0	0	0	2	0	0
E	0	0	0	0	2	0
F	0	0	0	0	0	5

D

$-$

	A	B	C	D	E	F
A	0	1	1	0	0	0
B	1	2	0	1	0	0
C	1	0	0	0	1	1
D	0	1	0	0	0	1
E	0	0	1	0	0	1
F	0	0	1	1	1	2

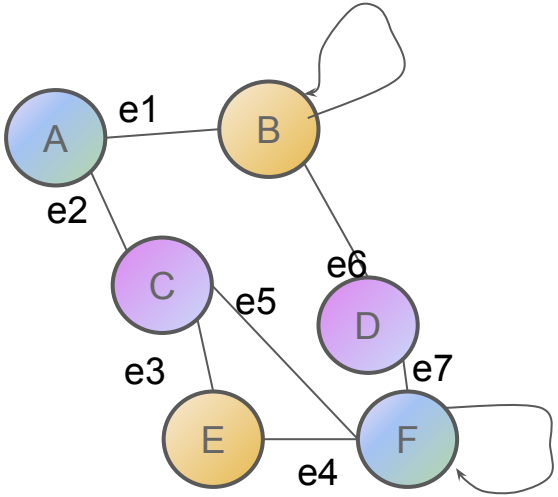
A

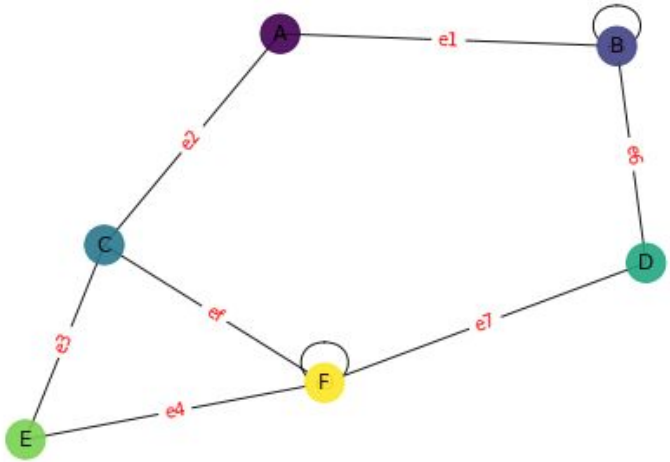
$=$

	A	B	C	D	E	F
A	2	-1	-1	0	0	0
B	-1	2	0	-1	0	0
C	-1	0	3	0	-1	-1
D	0	-1	0	2	0	-1
E	0	0	-1	0	2	-1
F	0	0	-1	-1	-1	3

$L = \{D - A\}$

Displaying the graph





Jupyter Notebook Exercise with NetworkX in Python

Graph Representation

Incidence (Relation Matrix) Matrix (I)

- Nodes n with edges m will have (I = nxm)

Adjacency Matrix (A)

- Square Matrix (A = NxN)

Degree Matrix (D)

- Diagonal Matrix (D)

Laplacian Matrix:

- L = D - A

Bag of Nodes

- Aggregate node level features using some mathematical approach i.e. taking the mean of the node degrees or histogram of the edge connections

Bag of nodes using Weisfeiler-Lehman Kernel

- Iteratively aggregate information from the node's neighbourhoods.
- The algorithm:
  - Set an initial label to each node of the graph, for example a node's degree
  - Iteratively assign a new label to each node using hashed labels from the neighbourhood
  - After K iterations, we now have gathered information from K-hop neighbourhood. We can then use any type of Bag of Nodes metrics to summarize these new labels

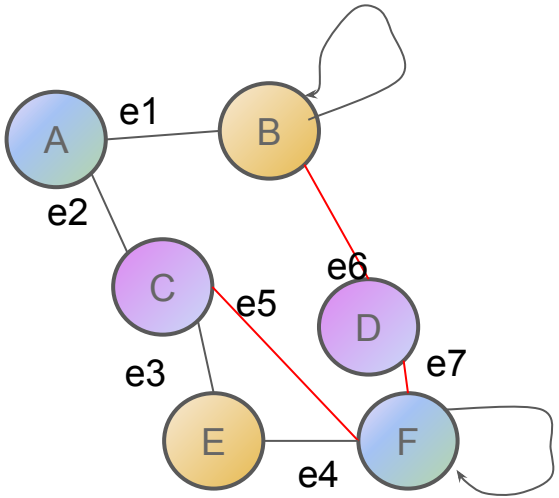
[Example usage in cheminformatics]

Ref: https://towardsdatascience.com/feature-extraction-for-graphs-625f4c5fb8cd

Graph Walking - Random Walk

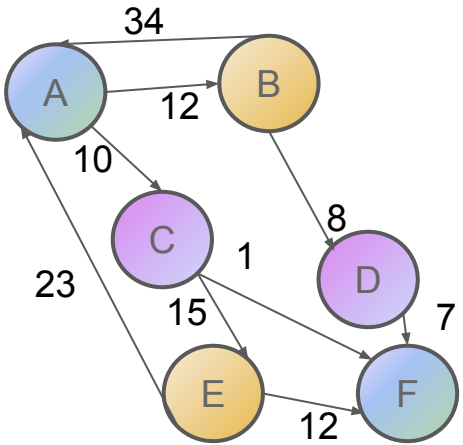
Traversing the graph through the connected nodes

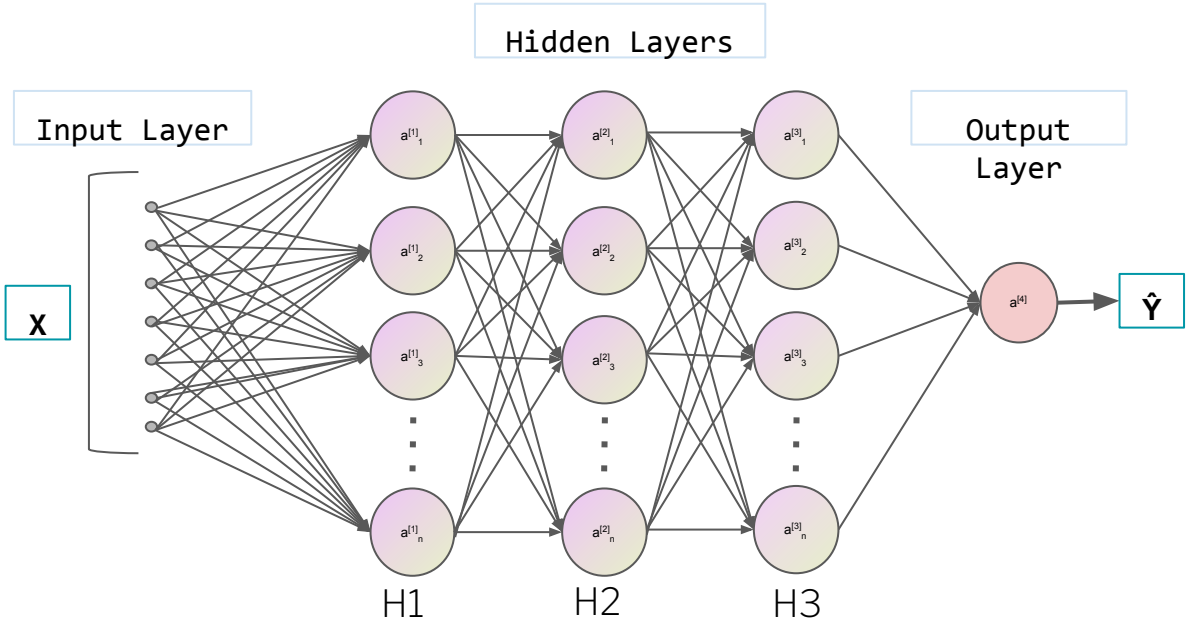
Random Walk is a technique to extract sequences from a graph from any random node perspective. It is used to train the skip-gram model to learn node embeddings.



C - F - **D** - B
C - E - F - **D** - R
C - A - B - **D**

Graph -> Graph Neural Networks

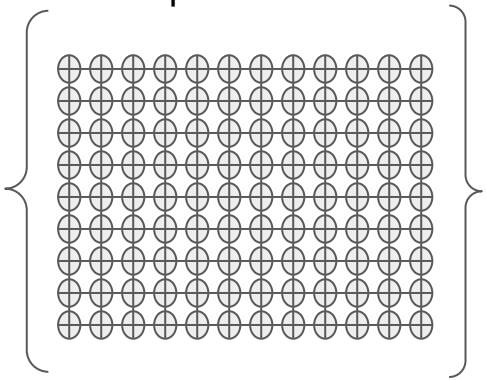




Role of Laplacian Matrix

First Order Difference of the image
f(x) = f(x+1, y) - f(x,y)

Smoothness of the **image** in the 2D Space



Smoothness of the **node/Vertices** in the Graph

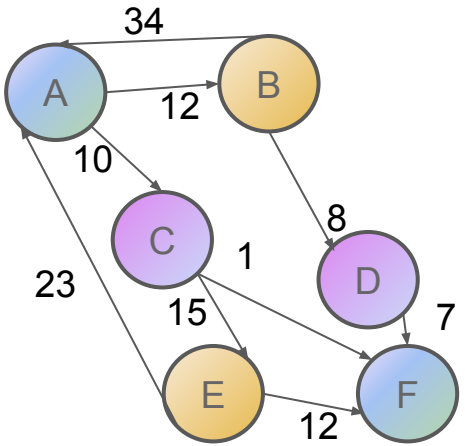
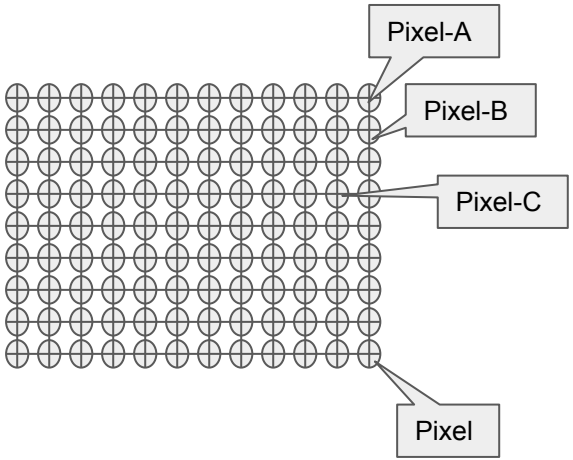
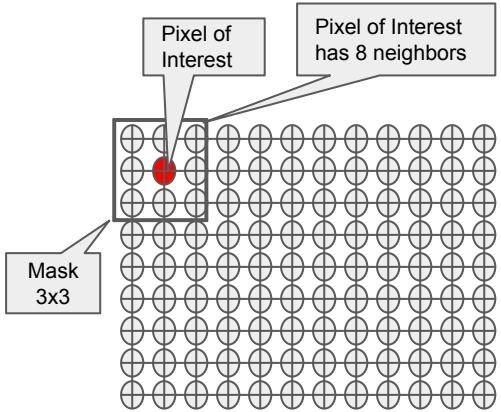


Image Representation as Fixed Size 2D Grid



- An image is defined as a regular Grid
  - Technically speaking 2D Grid
- Each Pixel is Fixed
- All the neighbouring pixel locations are also fixed
- Pixel A has 3 neighbors with fixed location
- Pixel B has 5 neighbors with fixed location
- Pixel C has 8 neighbors with fixed location

Convolution on Images - Convolution Neural Networks



Locality

- Contribution of N neighbors with reference to MxM mask will define the value of “Pixel of Interest”.
- Bringing mask to image sets up the **Locality**.

Aggregation

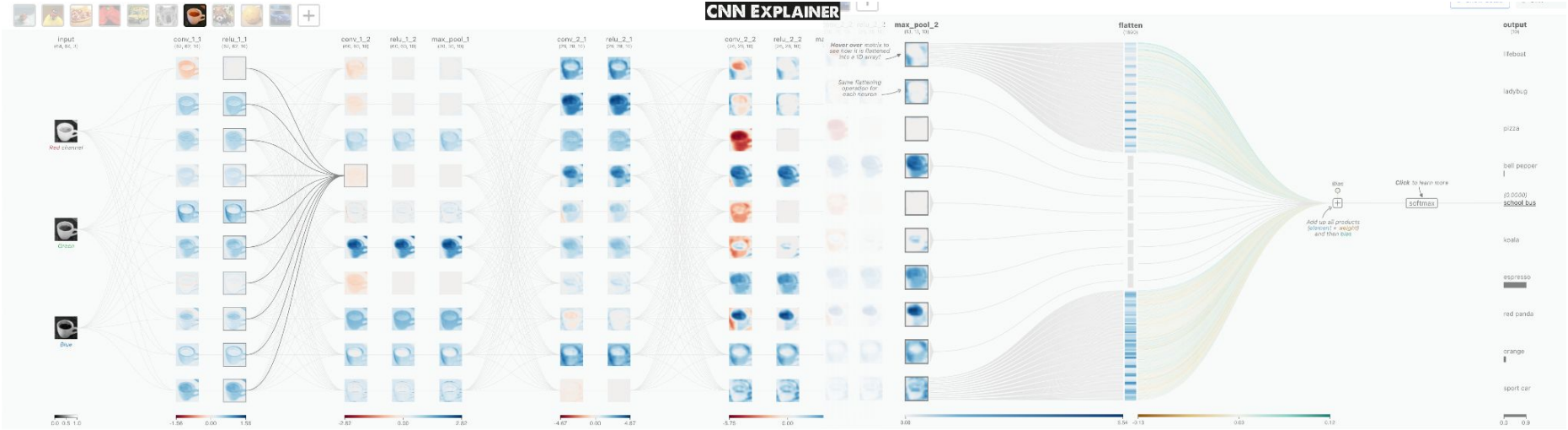
- Multiply the mask coefficients with all the pixels inside the mask and aggregate their sum will return the value of the center pixel
- The aggregates value replaces the value of the central Pixel

Convolution on Every Pixel as Mask moves from start to finish in the whole image

Composition

- Function of the Functions
- As conv happens from input to output into the Hidden layers the features happen to grow from basic to complex with fully recognized attributes
- Composition makes features enhanced from basic to complex

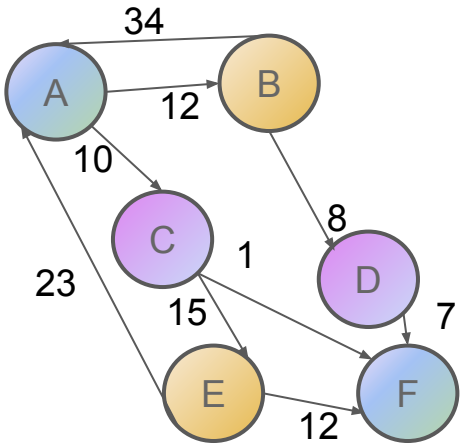
CNN: Composition - Function of the Functions



Layer complexity is growing

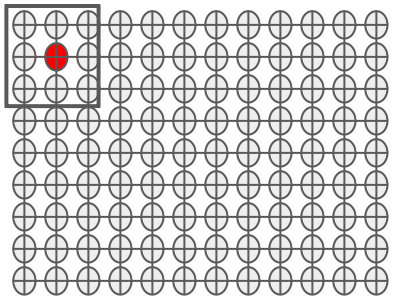
```
filters = 10
tiny_vgg = Sequential([
    Conv2D(filters, (3, 3), input_shape=(64, 64, 3), name='conv_1_1'),
    Activation('relu', name='relu_1_1'),
    ## 62,62, 10

    Conv2D(filters, (3, 3), name='conv_1_2'), ## 60,60, 10
    Activation('relu', name='relu_1_2'),
    ## 60, 60 >> 30,30
    MaxPool2D((2, 2), name='max_pool_1'),

    Conv2D(filters, (3, 3), name='conv_2_1'), ## 28,28, 10
    Activation('relu', name='relu_2_1'),

    Conv2D(filters, (3, 3), name='conv_2_2'), ## 26,26, 10
    Activation('relu', name='relu_2_2'), ##
    MaxPool2D((2, 2), name='max_pool_2'), ## 13, 13, 10

    # Input - 13x13,10 = 13*13*10 = 1690
    Flatten(name='flatten'),
    Dense(NUM_CLASS, activation='softmax', name='output')
])
```

Graph vs Fixed Grid Structured Data Objects



- Arbitrary Size Grid
- Non-euclidean Space
- Position and Sequence is not fixed, changes every time
- No fixed node order

Image



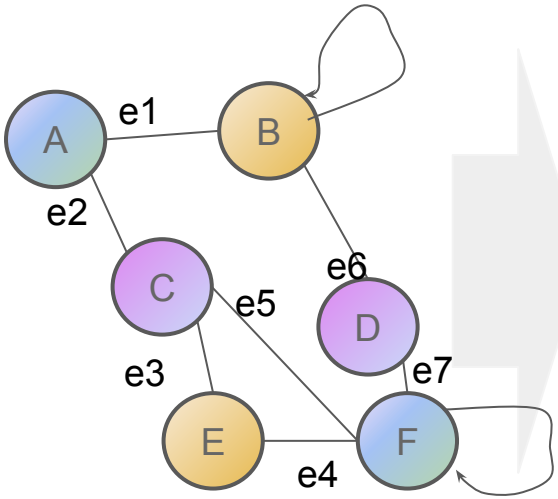
- Fixed Grid
- 2D Space
- Position & Sequence is fixed
- Pixel order is defined and stays exactly the same

Text

- Fixed Grid
- 1D Space
- Position and sequence is fixed
- Token/Vector orders is defined

Convolution requires grid and sequence to be fixed while processing structured data

Node Order and Adjacency Matrix



	A	B	C	D	E	F
A	0	1	1	0	0	0
B	1	2	0	1	0	0
C	1	0	0	0	1	1
D	0	1	0	0	0	1
E	0	0	1	0	0	1
F	0	0	1	1	1	2

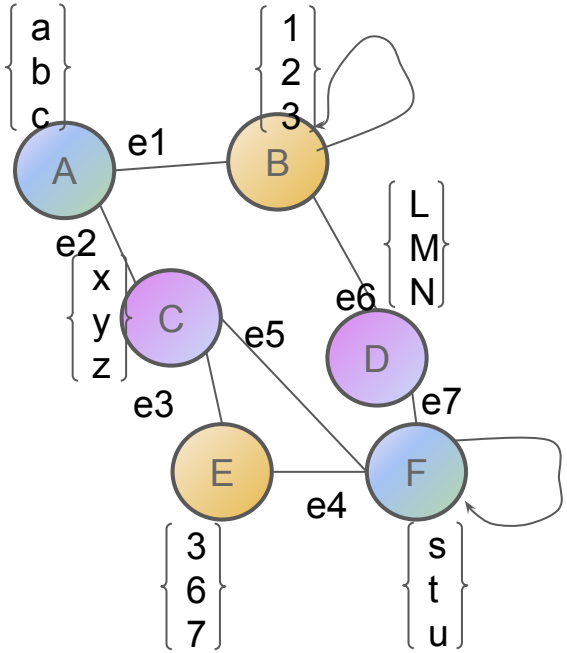
A

≠

	F	B	C	D	E	**A**
F	2	0	1	1	1	0
B	1	2	0	1	0	1
C	1	0	0	0	1	1
D	1	1	0	0	0	0
E	1	0	1	0	0	0
A	0	1	1	0	0	0

If we change the node order to something different the Adjacency matrix will change, this will change everything dependent on A.

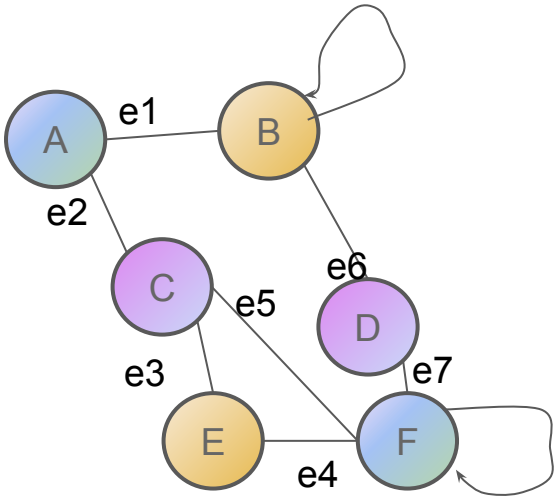
Graph Feature Matrix



- Graph G = {V, E}
- Adjacency Matrix - A = {N x N}
- Feature Matrix X = {N (nodes) x M (features) }
  - X = 6 x 3 = 18

	3		
A	a	b	c
B	1	2	3
C	x	y	z
D	L	M	N
E	3	6	7
F	s	t	u

Convolution on Graphs - Graph Neural Networks



Locality

- Contribution of All neighbors of the given nodes

+

Aggregation

- Perform Locality at all the nodes one by one

Convolution on Every Node by selected one node to another and aggregate all the neighbors contribution for the given node

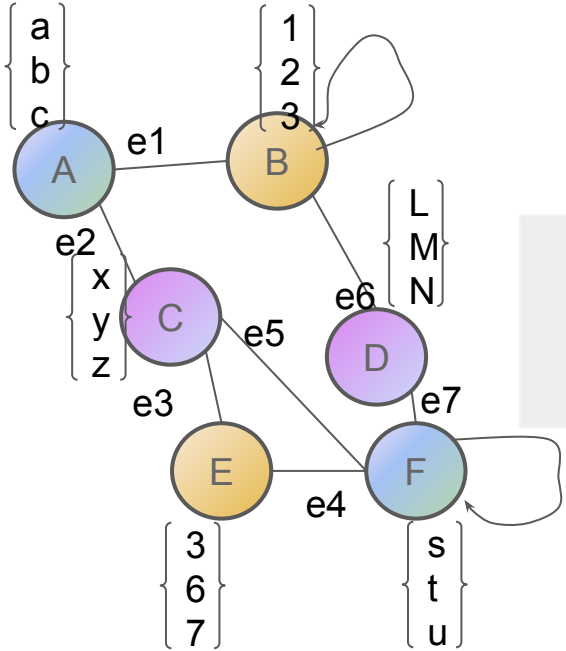
+

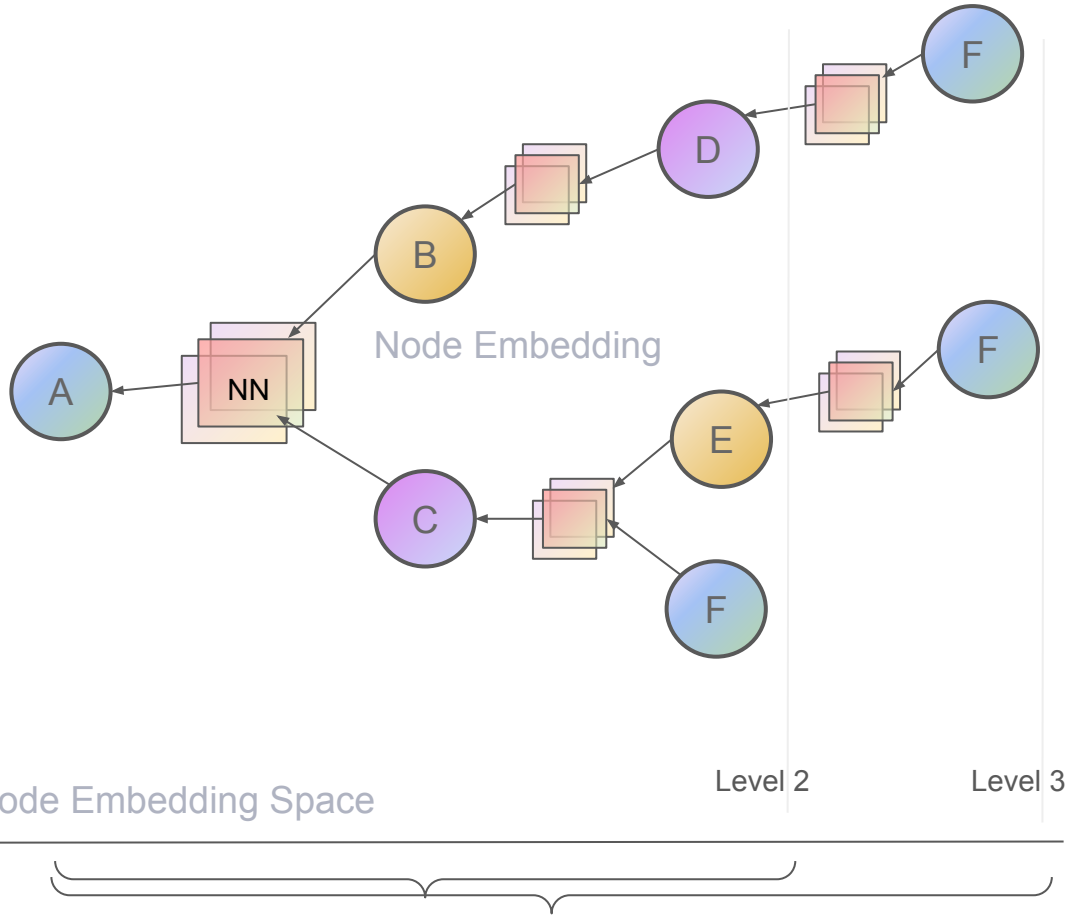
Composition

- Layer Stacking

Node Embedding is the answer - Its a way to represent the Graph nodes as **vectors**

Node Embeddings





If each node has path over 100s of levels deep, Node embedded can be selected arbitrary upto 2,3,4 levels

Repeat it for every node in the Graph
(**Embedding Depth** << **Original Node Depth**)
2/3/4 << 100/1000

Node Embedding - Applying convolution

Locality

- Node Embedding based on local neighborhood (Up to Level 2,3,4 of neighbors)

Aggregation:

- Aggregate Using Neural Networks (NN) (weights of each Input)

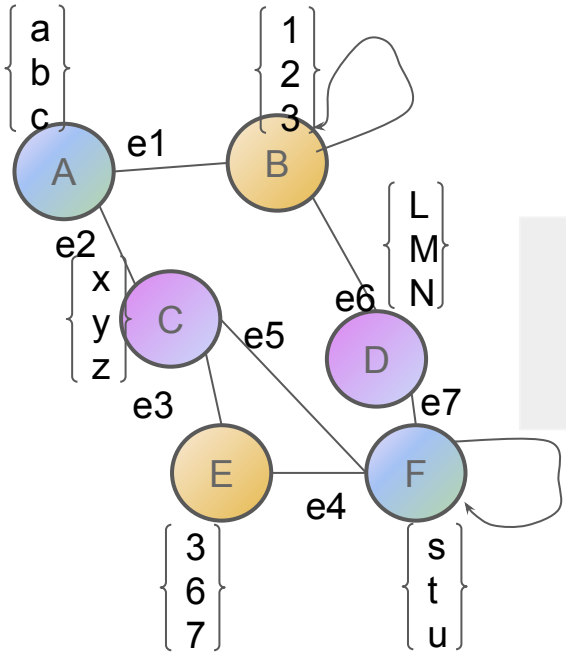
Composition:

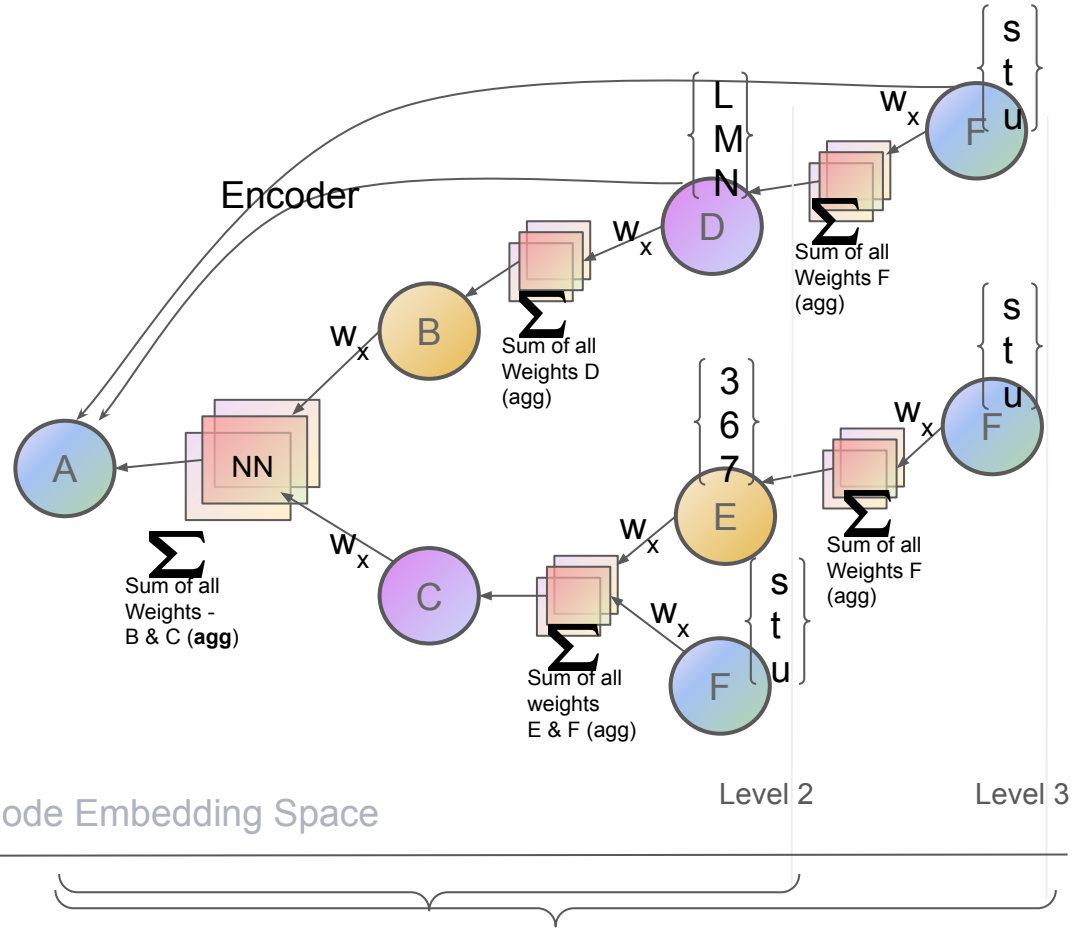
- Every node has its own computational Graph

Encoders

- Encoder function should be able to perform all 3 (Locality, Aggregation and Composition) functions all together

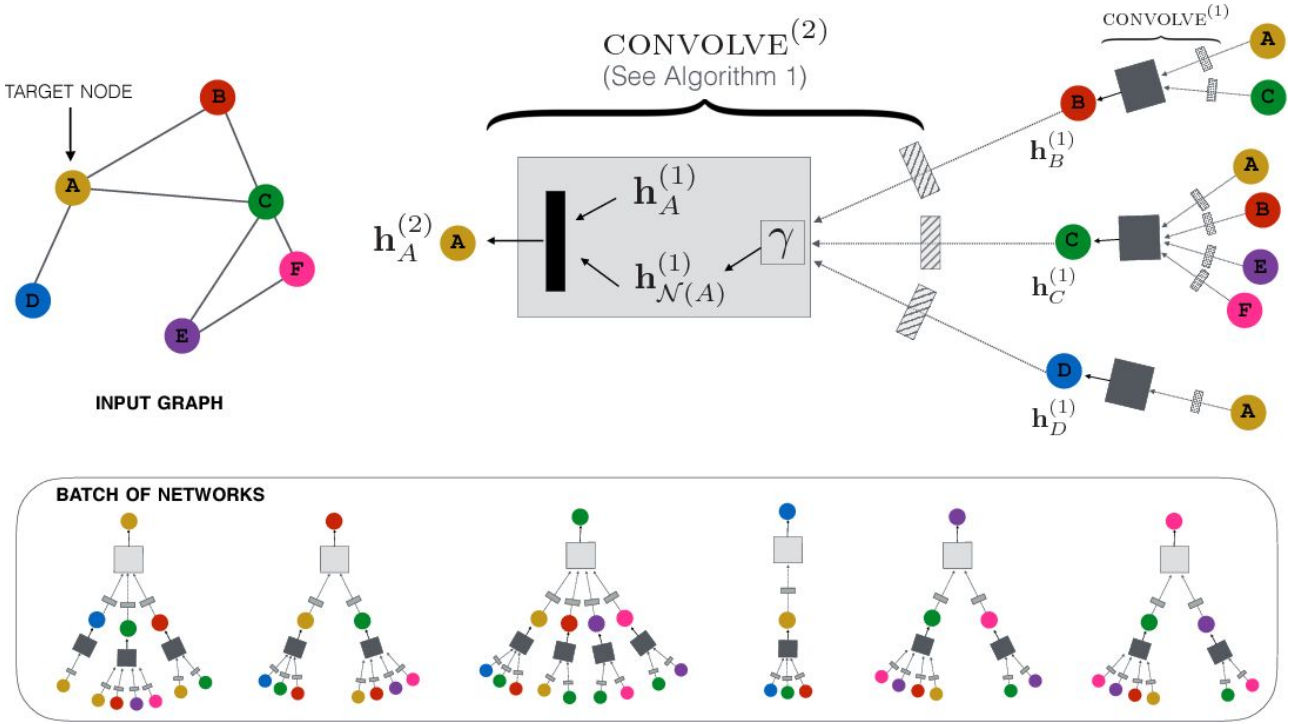
Node Embeddings





If each node has path over 100s of deep, still use embedded upto 2,3,4 levels or Hops away

Composition - Every node has own computational graph

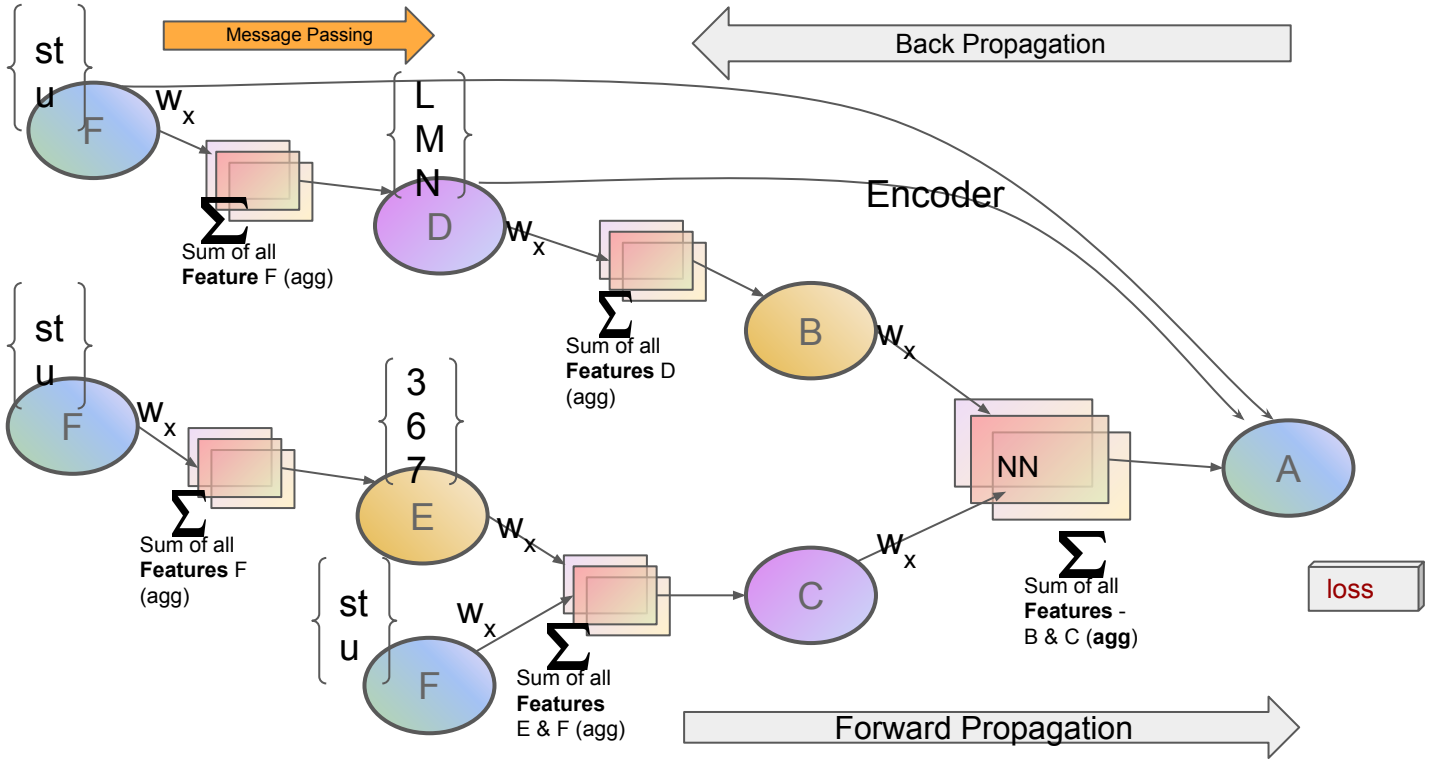


Img Reference: https://www.jiangekj.xyz/ProductDetail.aspx?iid=52504006&pr=41.88

Message Passing in Graph Neural Networks

- Every graph node has hidden states i.e. feature vectors
- For each node, we aggregate a function of hidden states and possibly edges of *all* neighbouring nodes with the node itself.
- Update the hidden state of the node using the the obtained message and the previous hidden state of the given node.
- Final message is the sum of all messages obtained from the neighbours for the given Node.

Message Passing with Propagation



Repetition are done several times to improve node A details

Advantages of Node Embeddings

- Node Embedding is a way to represent nodes as vectors
- Node embedding shows the network topology of the Graph
- Nodes that are similar in the network will have similar embeddings.
- Node embeddings is used for prediction in graph
-

Graph Neural Network Usage

- Node Similarity
  - Finding similar nodes with similar details
- Link Prediction
  - Recommendation of new potential edges for the given node
- Node classification
  - Classifying node type based on available types in the graph
- Clustering
  - Breaking embedding space grouping nodes with similarities together
  - Grouping based on common objective
- Graph classification
  - Aggregating node features and comparing it with other graph
  - Classifying different graphs with the same design objective
  - Breaking smaller graph for X from the main graph and matching it with Y from the same graph

Resources

Libraries

- https://networkit.github.io/
- https://github.com/danielegrattarola/spektral
- https://networkx.org/
- https://pytorch-geometric.readthedocs.io/en/latest/

Resources: Starters:

- https://www.kdnuggets.com/2018/05/wtf-tensor.html
- https://medium.com/tebs-lab/types-of-graphs-7f3891303ea8
- https://hadrienj.github.io/posts/Deep-Learning-Book-Series-2.1-Scalars-Vectors-Matrices-and-Tensors/
- https://towardsdatascience.com/an-introduction-to-graph-neural-network-gnn-for-analysing-structured-data-afce79f4cfdc
- https://towardsdatascience.com/understanding-graph-convolutional-networks-for-node-classification-a2bfdb7aba7b
- https://neptune.ai/blog/graph-neural-network-and-some-of-gnn-applications
- https://pub.towardsai.net/understanding-social-networks-409dffc785ea
- https://towardsdatascience.com/graph-coloring-with-networkx-88c45f09b8f4

Videos

- https://www.youtube.com/watch?v=GXhBEj1ZtE8
- https://www.youtube.com/watch?v=FdZ-EQkcHBo
- https://www.youtube.com/watch?v=3IS7UhNMQ3U&list=PLoROMvodv4rPLKxIpqhjhPgdQy7imNkDn&index=5

Resources

Advance:

- https://towardsdatascience.com/hands-on-graph-neural-networks-with-pytorch-pytorch-geometric-359487e221a8
- https://medium.com/analytics-vidhya/getting-the-intuition-of-graph-neural-networks-a30a2c34280d
- https://medium.com/@BorisAKnyazev/tutorial-on-graph-neural-networks-for-computer-vision-and-beyond-part-1-3d9fada3b80d

Examples

- https://graphsandnetworks.com/the-cora-dataset/

Documentations

- https://networkx.org/documentation/networkx-1.10/tutorial/tutorial.html
- https://pytorch-geometric.readthedocs.io/en/latest/notes/introduction.html