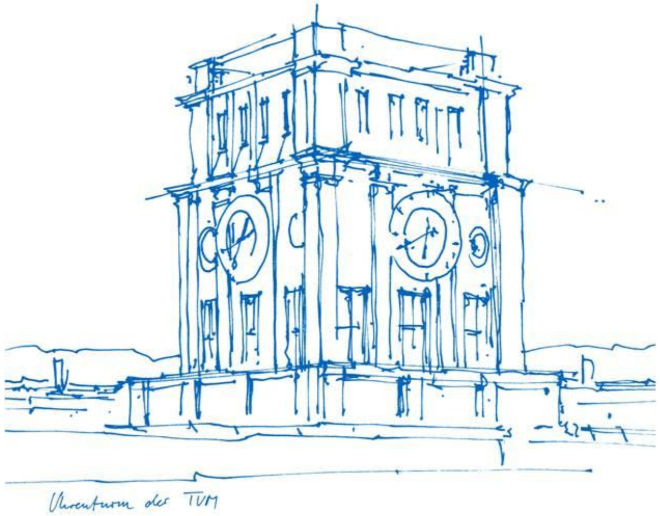
Learning For Self-Driving Cars and Intelligent Systems

26 April 2021

Qadeer Khan

https://vision.in.tum.de/teaching/ss2021/intellisys_ss2021



Announcements

- As requested, deadline for the task introduced this week(Task 3 on GNN’s) has now been extended to 16 May.
- Deadline for Task 2 (Carla) will remain the same i.e. 2 May.
- There will be an “Office hour” session this Thursday at 12:30pm to discuss any questions or confusions you may have. (Same online meeting room as the lecture)
- Please make sure to anonymize your coding task reports if feedback is desired.
- Also make sure to push your report as pdf files
- As requested, I will also put the anonymized reports in the shared folder after the task deadlines.
- If you have any objection to having your anonymized report placed on the shared folder, then please let me know latest by this Thursday(29 April, 2021)

Shared Folder : /storage/group/intellisys

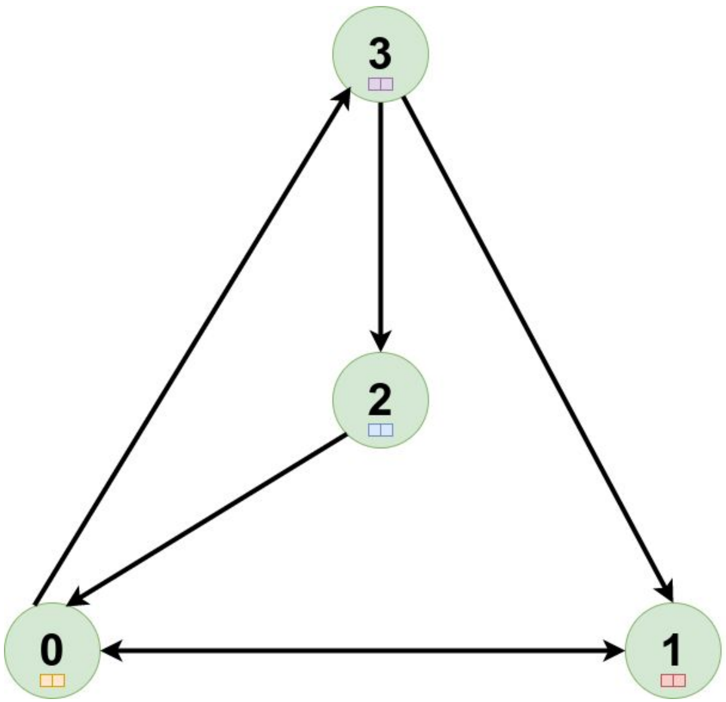
Graphical Networks

Comprise of Nodes/Vertices(V) connected by Edges(E)

Edges can be directed or undirected

Directed Graph:

(Each node is represented by a 2-dimensional feature vector)



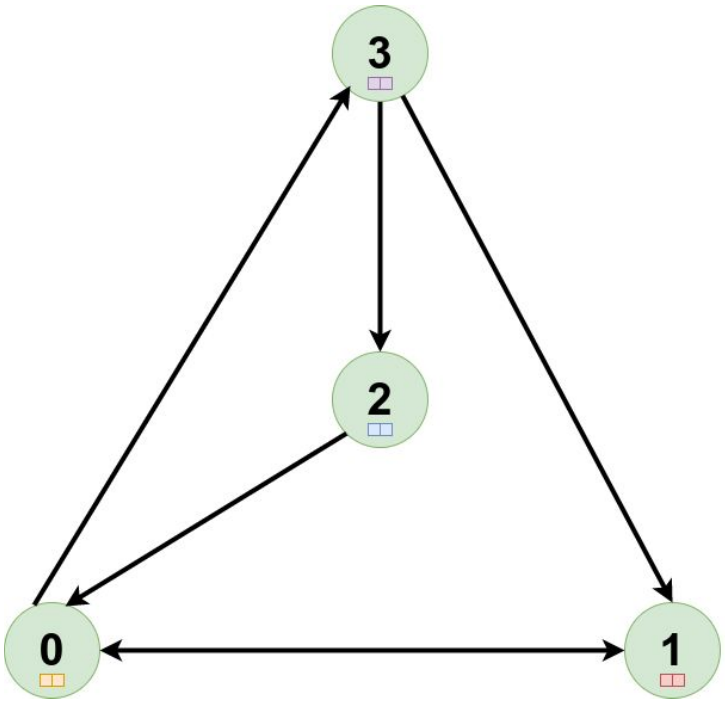
Examples

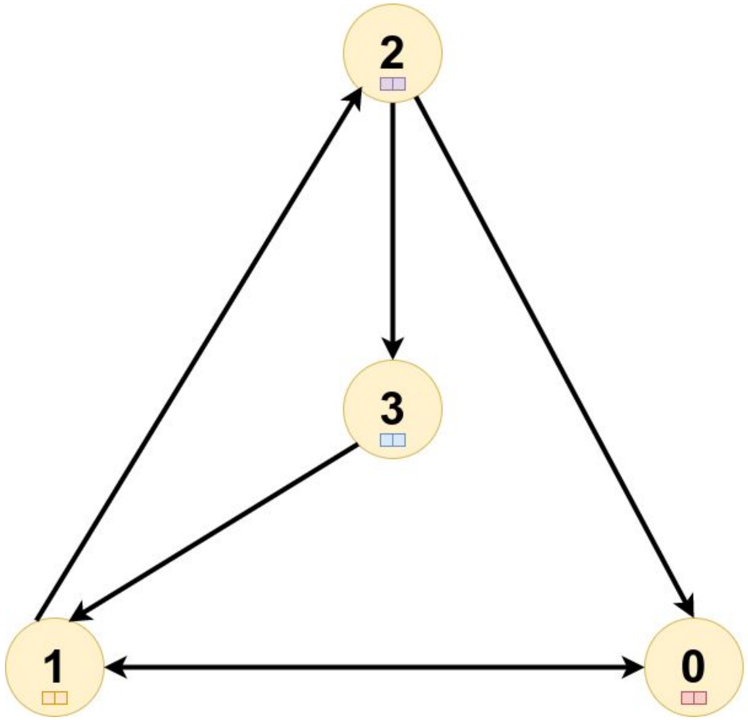
Some basic example

- *Social Networks*. Vertices(individuals), Edges(connections). Undirected (Xing), Directed(Twitter)
- *Molecules*. Vertex(Atoms) Edges(bonds)
- Power System. Vertex(Voltage at nodes), Edge(Resistance/current flow)
- *Recommender System* Vertex(individuals/products), Edges(ratings)

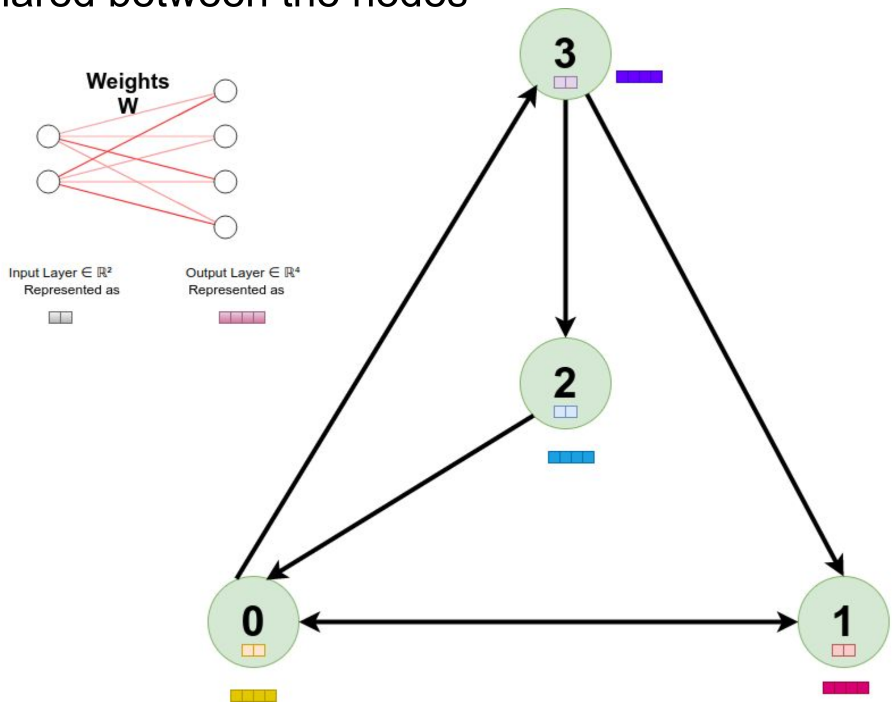
Permutation invariance

are they the same graph?





- Features of a each node can be converted using a Fully connected layers of variable size
- In this case it is 2 x 4
- Weights are shared between the nodes



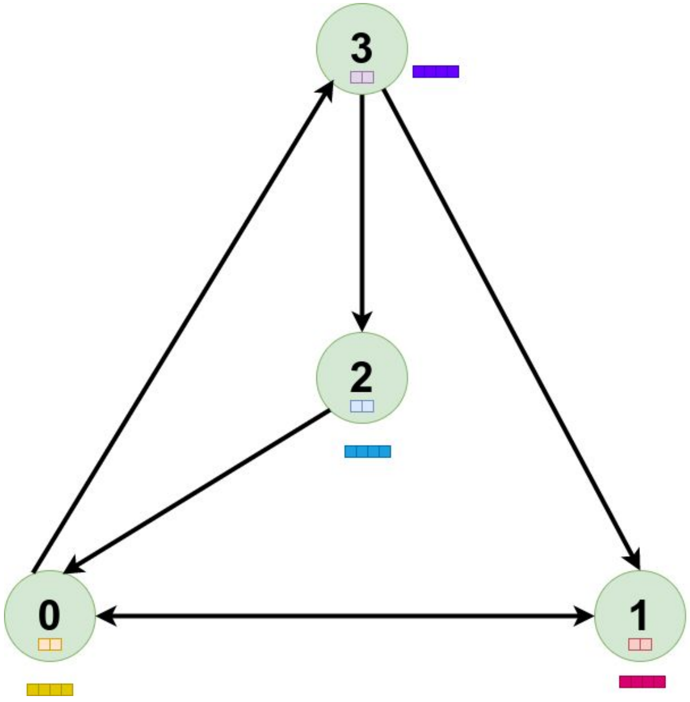
Representation

Adjacency matrix of n x n

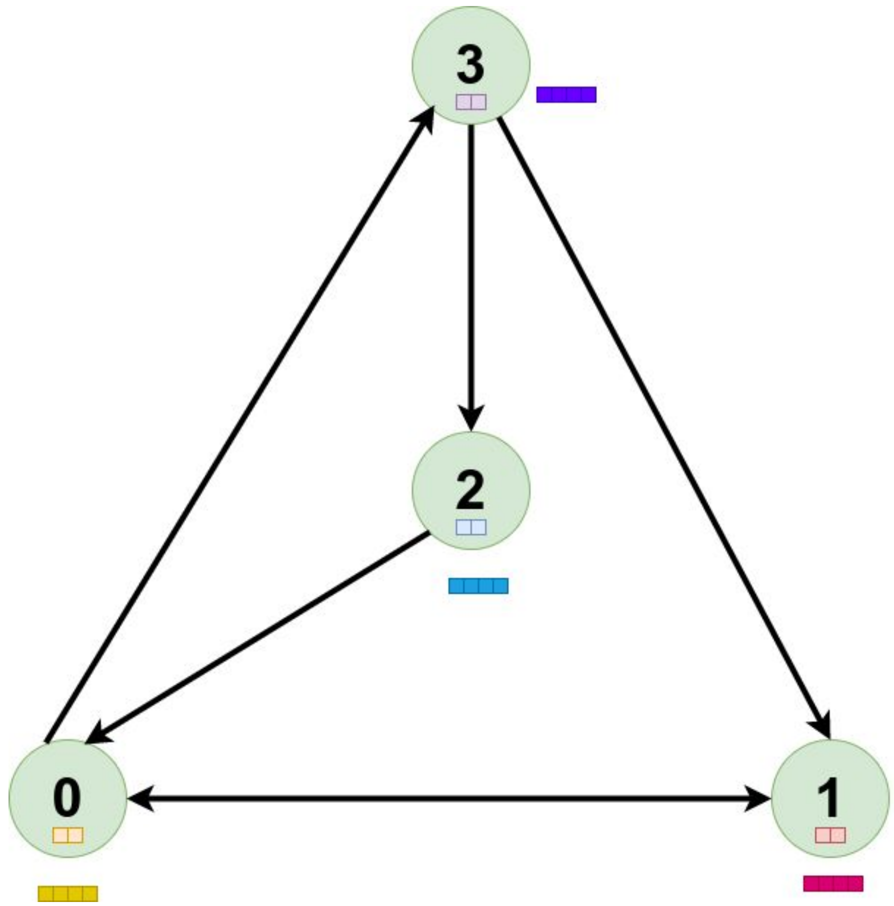
n = # of nodes

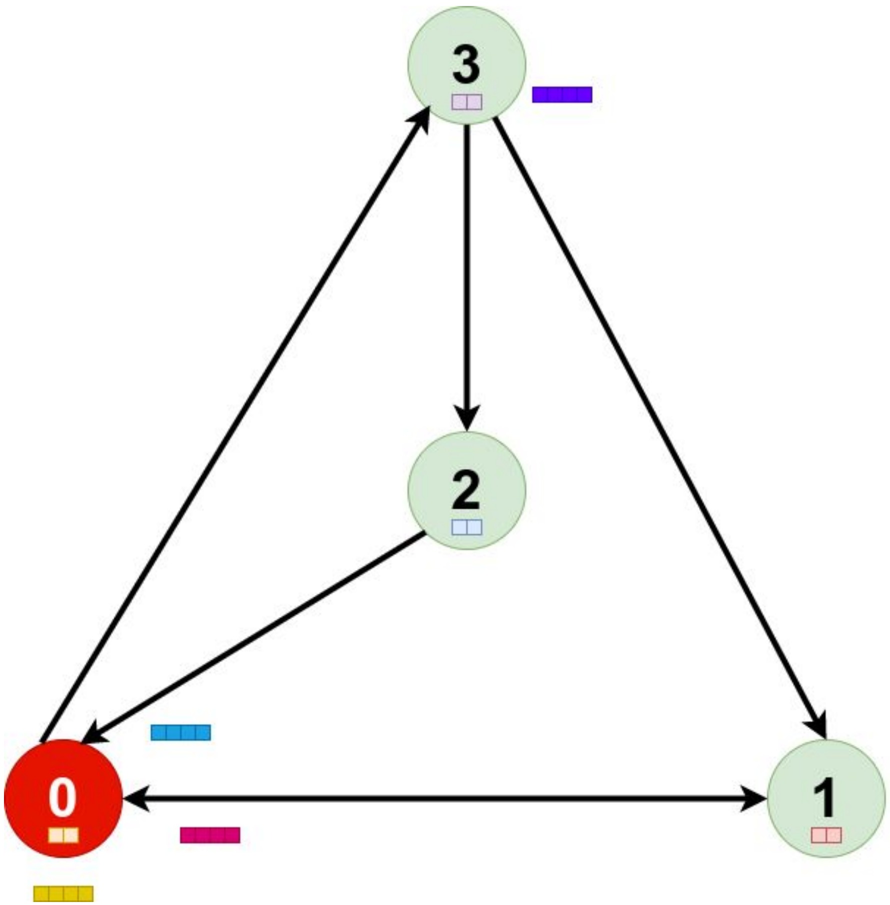
Entries in the adjacency matrix correspond to edges

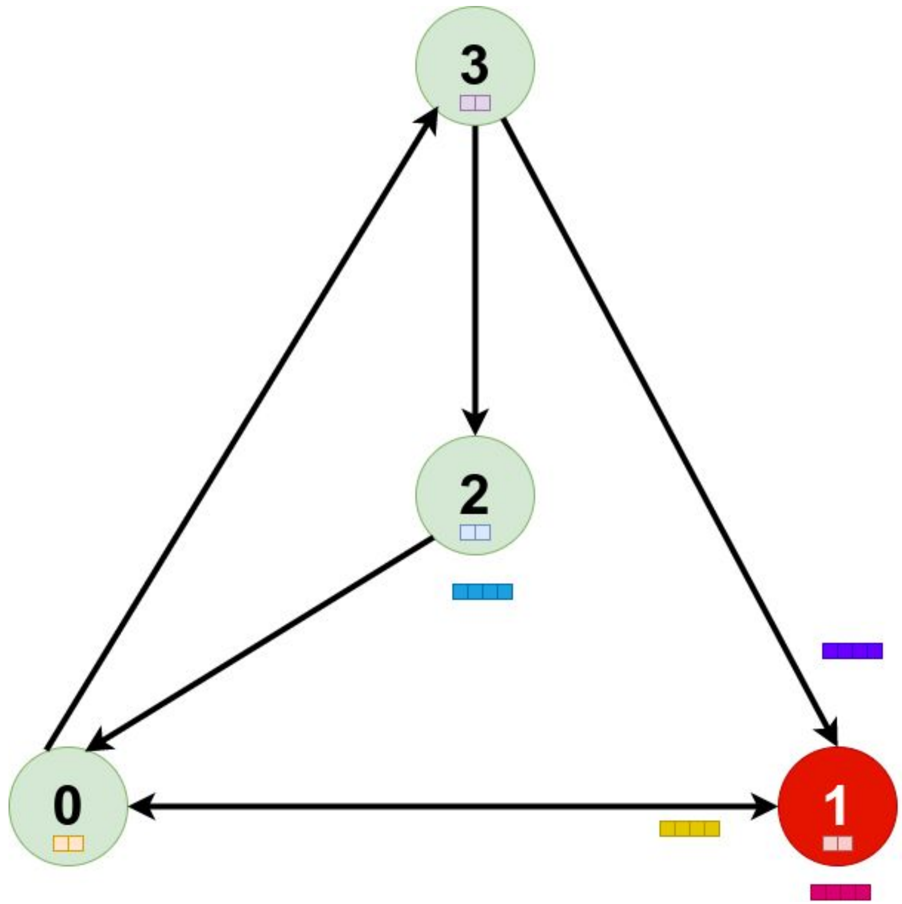
$$Adjacency = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

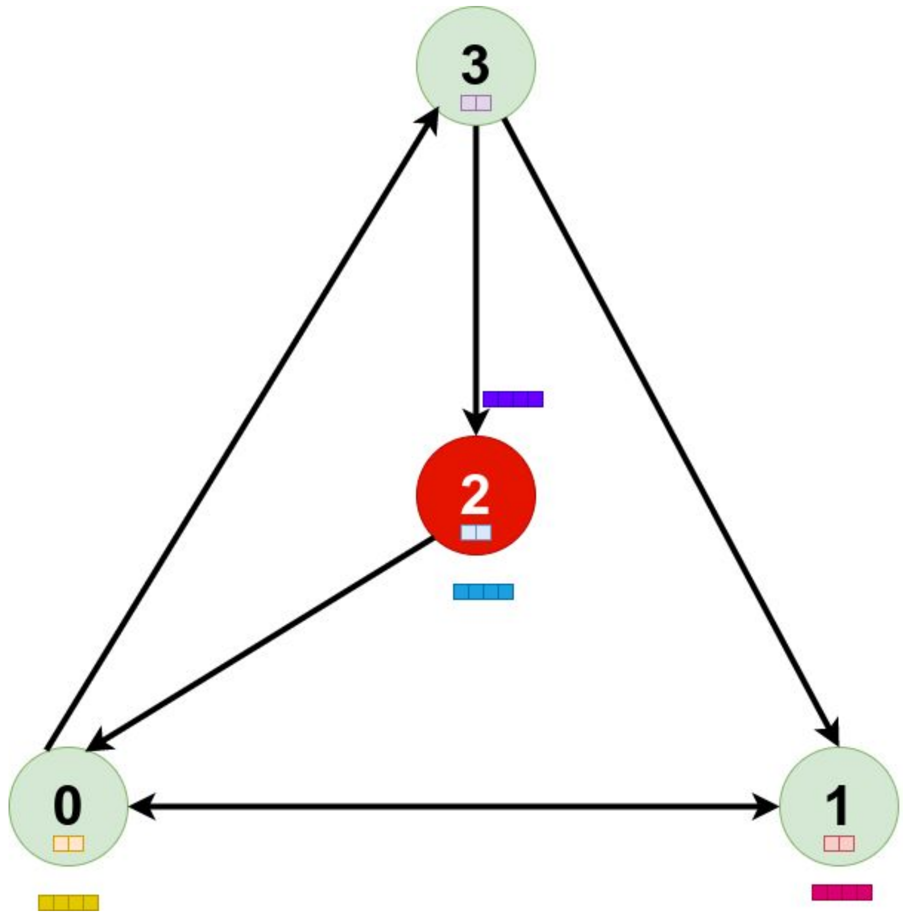


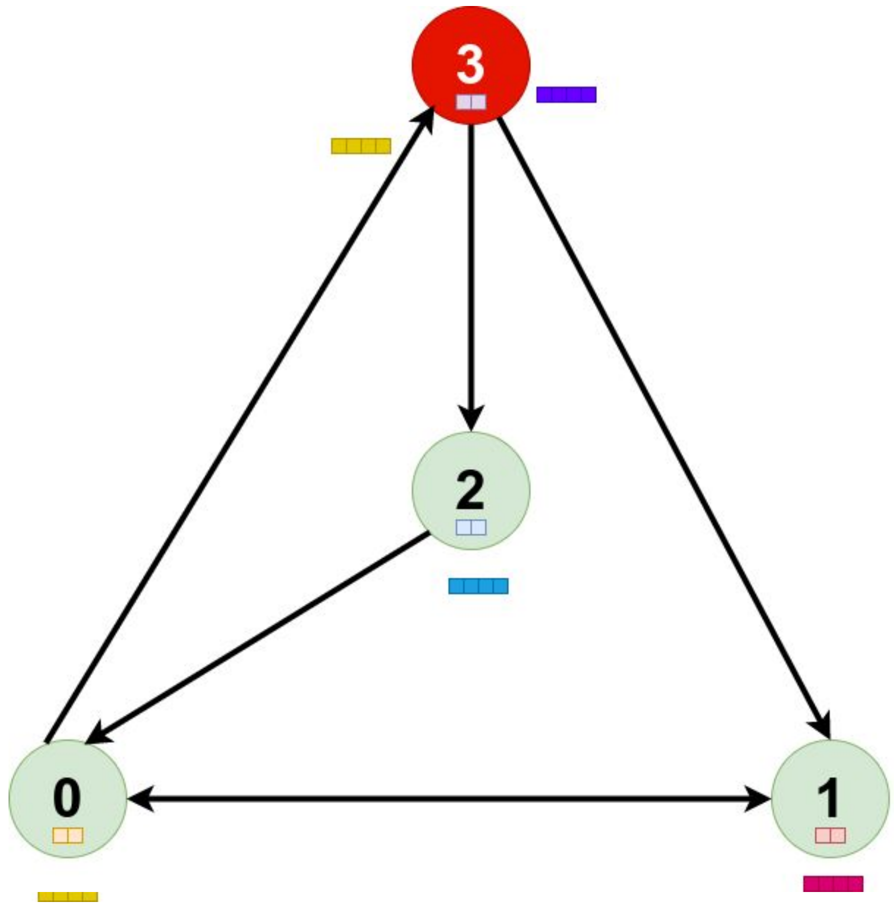
- Each node receives information from its neighbours
- Edges indicate which are the neighbouring nodes
- Perform order invariant operations on the received information:
    - Sum
    - mean
    - max
- We take sum in the following example


3
2
0
1


3
2
0
1


3
2
0
1

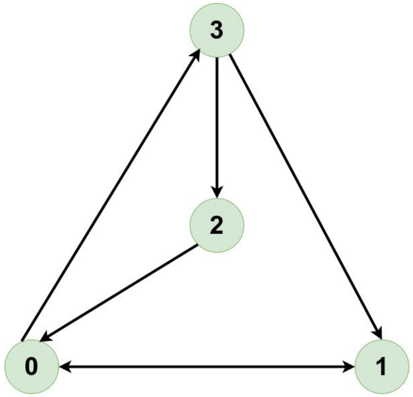

3
2
0
1


3
2
0
1

Features = [[1. 1.]
[2. 4.]
[3. 9.]
[4. 16.]]

Adjacency = [[0., 1., 1., 0.],
[1., 0., 0., 1.],
[0., 0., 0., 1.],
[1., 0., 0., 0.]]

New Features = Adjacency * Features = [[5., 13.],
[5., 17.],
[4., 16.],
[1., 1.]]

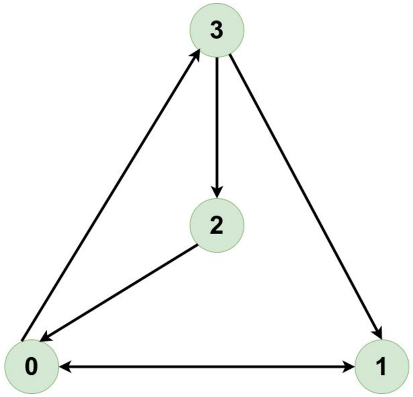


But we want to add information from the transformed neighbouring features

Let Weights of the fully connected layer be the following 2 x 4 matrix:

W = [[0.4, -2,0.6,1.2],
[0.3, -2,0.7,-1.3]]

Features = [[1. 1.]
[2. 4.]
[3. 9.]
[4. 16.]]



Transformed Features = Features * W = [[0.7, -4. , 1.3, -0.1],
[2. , -12. , 4. , -2.8],
[3.9, -24. , 8.1, -8.1],
[6.4, -40. , 13.6, -16.]]

New features = Adjacency * Transformed Features = [[5.9, -36. , 12.1, -10.9],
[7.1, -44. , 14.9, -16.1],
[6.4, -40. , 13.6, -16.],
[0.7, -4. , 1.3, -0.1]]

For every node the new features do not contain information from itself.

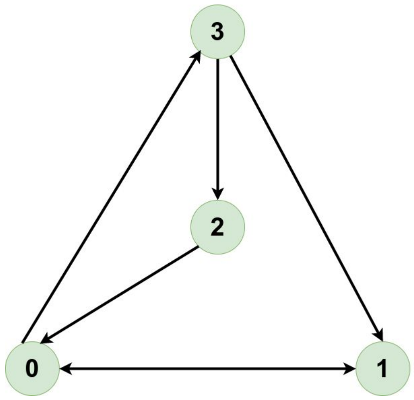
Information is **only** passed from the neighbours

Add self-loops in the adjacency matrix

A = A + Identity(n)

```
New Adjacency =         [[1. 1. 1. 0.]
                         [1. 1. 0. 1.]
                         [0. 0. 1. 1.]
                         [1. 0. 0. 1.]]
```

```
New Features = [[  6.6 -40.   13.4 -11. ]
                [  9.1 -56.   18.9 -18.9]
                [ 10.3 -64.   21.7 -24.1]
                [  7.1 -44.   14.9 -16.1]]
```



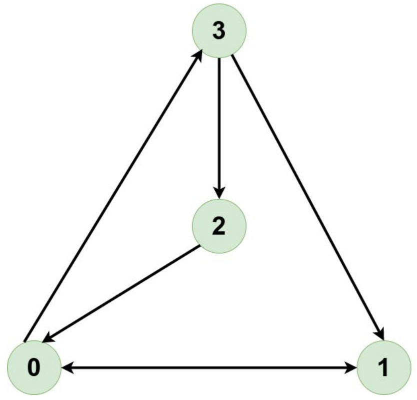
Degree normalization

Degree = # of neighbours of a node where information is sent

Some nodes have more neighbours than others

Therefore, nodes with higher number of nodes will have a higher sum.

We should therefore normalize such that the weighted average of the information sent to all neighbours adds up to 1



D = [[3. 0. 0. 0.]
[0. 2. 0. 0.]
[0. 0. 2. 0.]
[0. 0. 0. 3.]]

New Adjacency = [[1. 1. 1. 0.]
[1. 1. 0. 1.]
[0. 0. 1. 1.]
[1. 0. 0. 1.]]

inv(D) * A = [[0.33 0.33 0.33 0.]
[0.5 0.5 0. 0.5]
[0. 0. 0.5 0.5]
[0.33 0. 0. 0.33]]

We are effectively multiplying each edge by the entries given in the above matrix

Final solution for new feature =

inv(D) * new adjacency * Features * Weights

```
[[  2.2000, -13.3333,   4.4667,  -3.6667],
[  4.5500, -28.0000,   9.4500,  -9.4500],
[  5.1500, -32.0000,  10.8500, -12.0500],
[  2.3667, -14.6667,   4.9667,  -5.3667]]
```

Pytorch Geometric

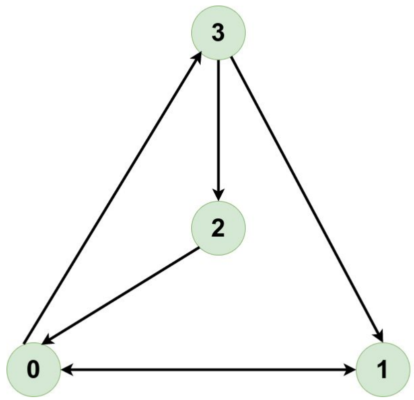
Writing out an adjacency matrix infeasible for large graphs

Sparse matrix, few non-zero entries

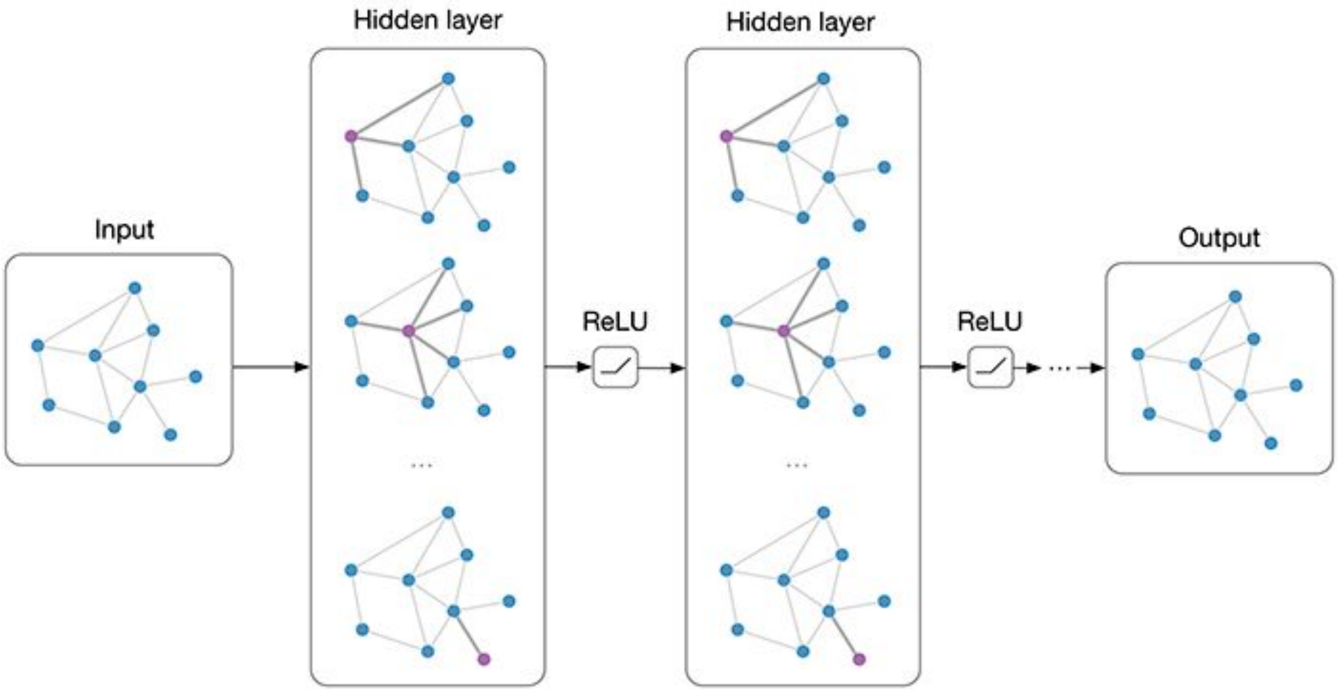
Rather than matrix, represent as edges

[[0, 0, 1, 2,3,3],

[1, 3, 0, 0,2,1]]



Multi-Layer Graph Convolutions



Ref: https://tkipf.github.io/graph-convolutional-networks/

Task 5

- Classify the point cloud contained obtained from images contained in
  /storage/group/intellisys/datasets/carla/episode_000/CameraDepthXX
- See tutorial_week3.ipynb on how to create a point cloud from a depth image.
  - Similar to week1 tutorial
- Semantic labels for each pixel is given in:
  /storage/group/intellisys/datasets/carla/episode_000/CameraSegXX
- The R channel of images contained in
  /storage/group/intellisys/datasets/carla/episode_000/CameraSegXX contain the semantic labels
- Labels are from 0-13.
- Some labels may be missing (for car, pedestrian etc.)

Some Hints

- Many nodes - downsample image to a smaller size
- Or downsample the point cloud (using for e.g. Farthest point sampling)
- Features could be x,y,z values, r,g,b values, nx,ny,nz normals for each point or their combination
    - If using normals, see this for estimation of point normals: http://www.open3d.org/docs/release/tutorial/geometry/pointcloud.html#Vertex-normal-estimation

- Try different conv. layers for e.g. EdgeConv, GraphConv, PPFConv etc. For more see: https://pytorch-geometric.readthedocs.io/en/latest/modules/nn.html#convolutional-layers
- Also look into GraphUnet (https://pytorch-geometric.readthedocs.io/en/latest/modules/nn.html#torch_geometric.nn.models.GraphUNet)

Guidelines

Steps:

1. 2D depth Image is first projected into 3D
2. A graph is formed using K nearest neighbours
   https://pytorch-geometric.readthedocs.io/en/1.3.0/_modules/torch_cluster/knn.html
3. A GNN model is trained that inputs this graph and outputs the semantic label of each point in the graph
4. Each point with the predicted semantic label can then be back-projected into a 2D image
5. Report the accuracy of correct predictions

Submission

- Submit an inference.py script which takes in any image path given by /storage/group/intellisys/datasets/carla/episode_000/CameraDepthXX/image_00yyy.png and outputs the semantic class map and also prints the accuracy
- If you could train for only one image specify it in the report

Bonus when training for a single image:

  - We know that even without training the embeddings for similar and closely related features are similar to each other
  - Therefore, not all semantic labels may be necessary
  - So make a plot to check the accuracy of the model when using for e.g. 100 labels, 200 labels, 300 labels etc.
  - As more labels are added the accuracy improves
  - Accuracy may already saturate without exhausting all labels

References

- Method used in the slides: https://towardsdatascience.com/how-to-do-deep-learning-on-graphs-with-graph-convolutional-networks-7d2250723780
- Introductory Tutorial: https://pytorch-geometric.readthedocs.io/en/1.3.0/notes/introduction.html
- Notebooks: https://pytorch-geometric.readthedocs.io/en/latest/notes/colabs.html
- Implementing your own dataloader: https://pytorch-geometric.readthedocs.io/en/latest/notes/create_dataset.html

QUESTIONS