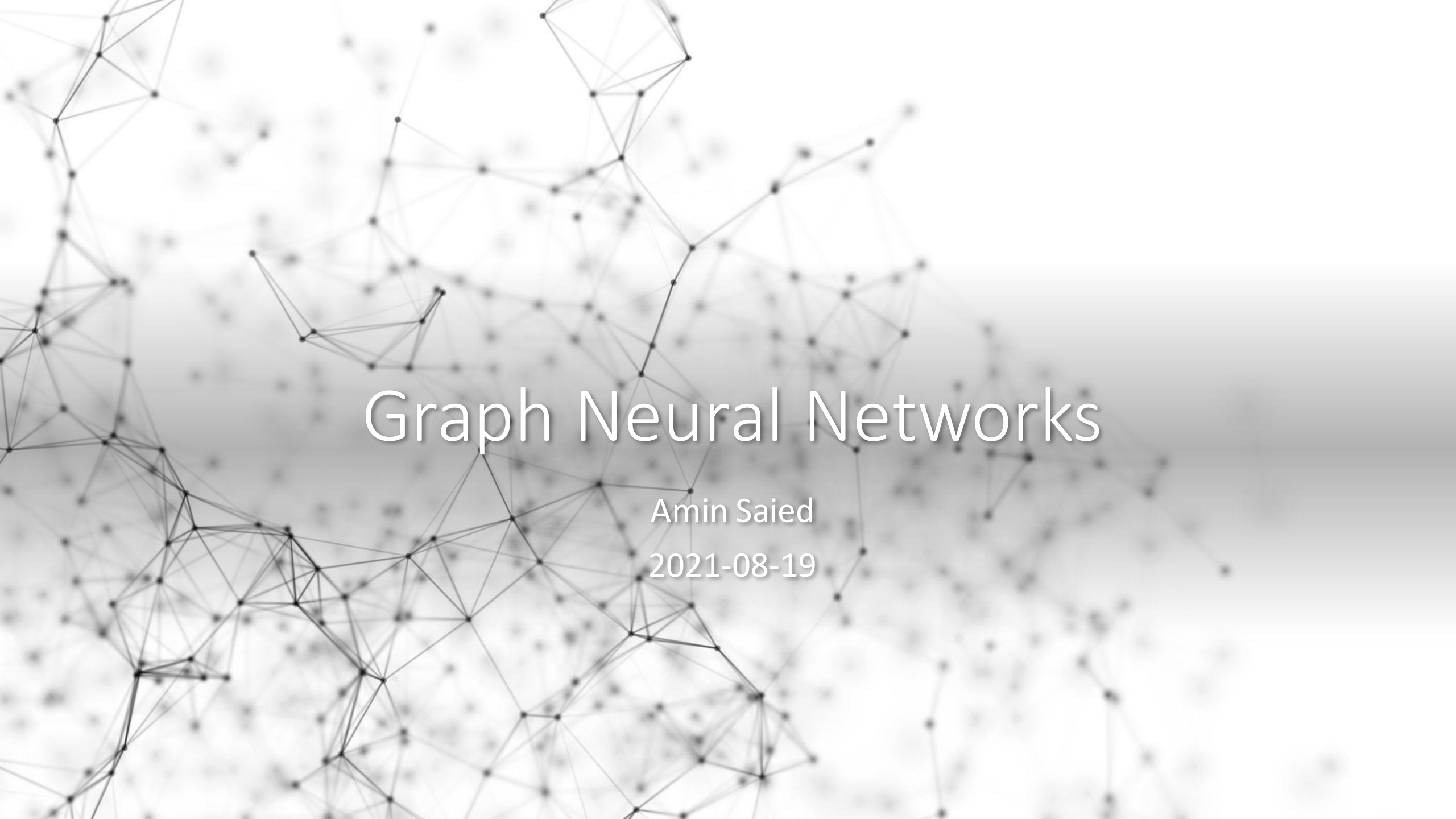
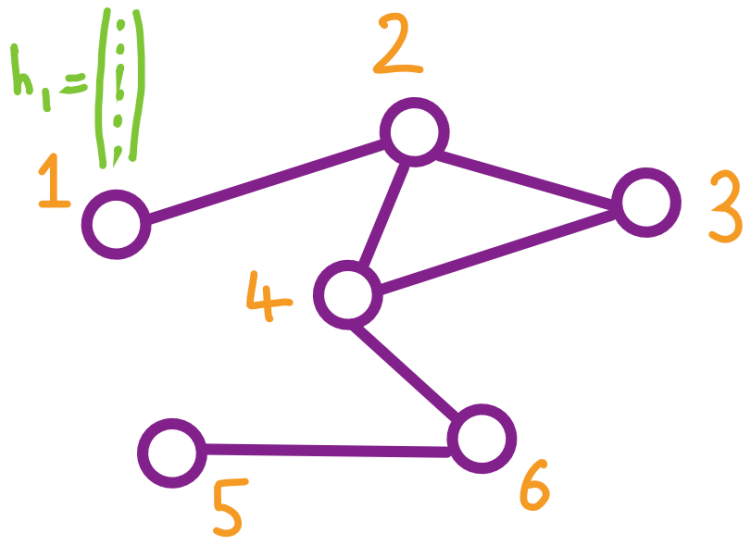


Graph Neural Networks

Amin Saied

2021-08-19

Introduction to GNNs



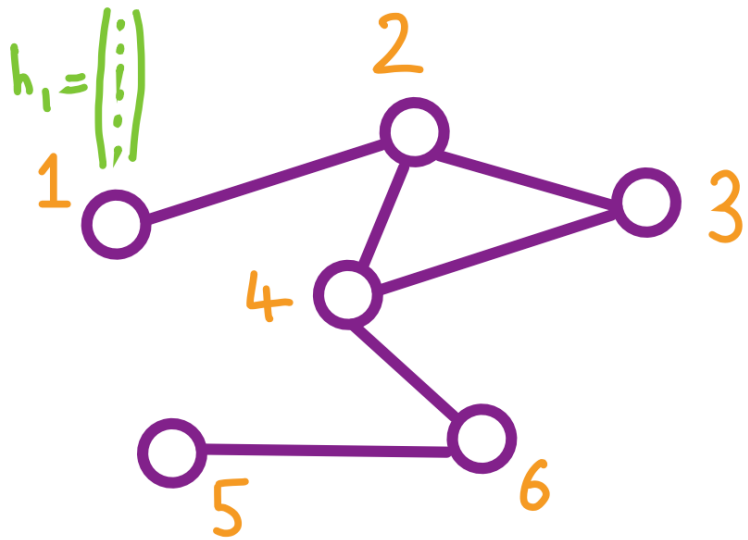
$$A = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{pmatrix} 1 & 1 & \cdot & \cdot & \cdot & \cdot \\ 1 & 1 & 1 & 1 & \cdot & \cdot \\ \cdot & 1 & 1 & 1 & \cdot & \cdot \\ \cdot & 1 & 1 & 1 & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & 1 & 1 \\ \cdot & \cdot & \cdot & 1 & 1 & 1 \end{pmatrix} \end{array}$$

n.b. self loops

node features

$$H = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{pmatrix} \leftarrow & h_1 & \rightarrow \\ & h_2 & \\ & h_3 & \\ & h_4 & \\ & h_5 & \\ \leftarrow & h_6 & \rightarrow \end{pmatrix}$$

$\underbrace{\qquad}_{\text{dim} = d}$

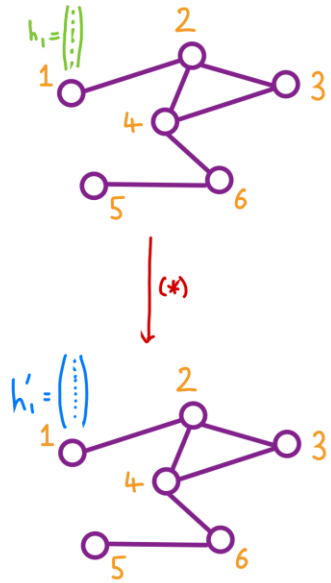


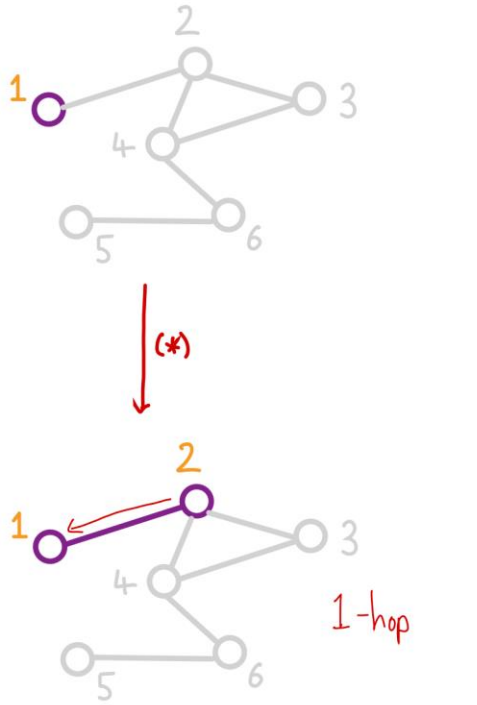
Vanilla GNN: $h_i' = \sigma\left(\sum_{j \in E(i)} W \cdot h_j \right)$ (*)

$H = \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{matrix} \begin{pmatrix} \leftarrow & h_1 & \rightarrow \\ & h_2 & \\ & h_3 & \\ & h_4 & \\ & h_5 & \\ \leftarrow & h_6 & \rightarrow \end{pmatrix}$ $\underset{\dim = d}{}$ $\overset{(*)}{\rightsquigarrow}$ $H' = \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{matrix} \begin{pmatrix} \leftarrow & h_1 & \rightarrow \\ & h_2 & \\ & h_3 & \\ & h_4 & \\ & h_5 & \\ \leftarrow & h_6 & \rightarrow \end{pmatrix}$ $\underset{\dim = d'}{}$

$$H' = \sigma\left(\underset{(|V|,|V|)}{A} \; \underset{(|V|,d)}{H} \; \underset{(d,d')}{W} \right)$$

$(|V|, d')$


h_1 =
1
2
3
4
5
6
(*)
h'_1 =
1
2
3
4
5
6


1
2
3
4
5
6
(*)
1
2
3
4
5
6
1-hop
(*)
1
2
3
4
5
6
2-hops

Attention GNNs

GRAPH ATTENTION NETWORKS

Petar Veličković*
Department of Computer Science and Technology
University of Cambridge
petar.velickovic@cst.cam.ac.uk

Guillem Cucurull*
Centre de Visió per Computador, UAB
gcucurull@gmail.com

Arantxa Casanova*
Centre de Visió per Computador, UAB
ar.casanova.8@gmail.com

Adriana Romero
Montréal Institute for Learning Algorithms
adriana.romero.soriano@umontreal.ca

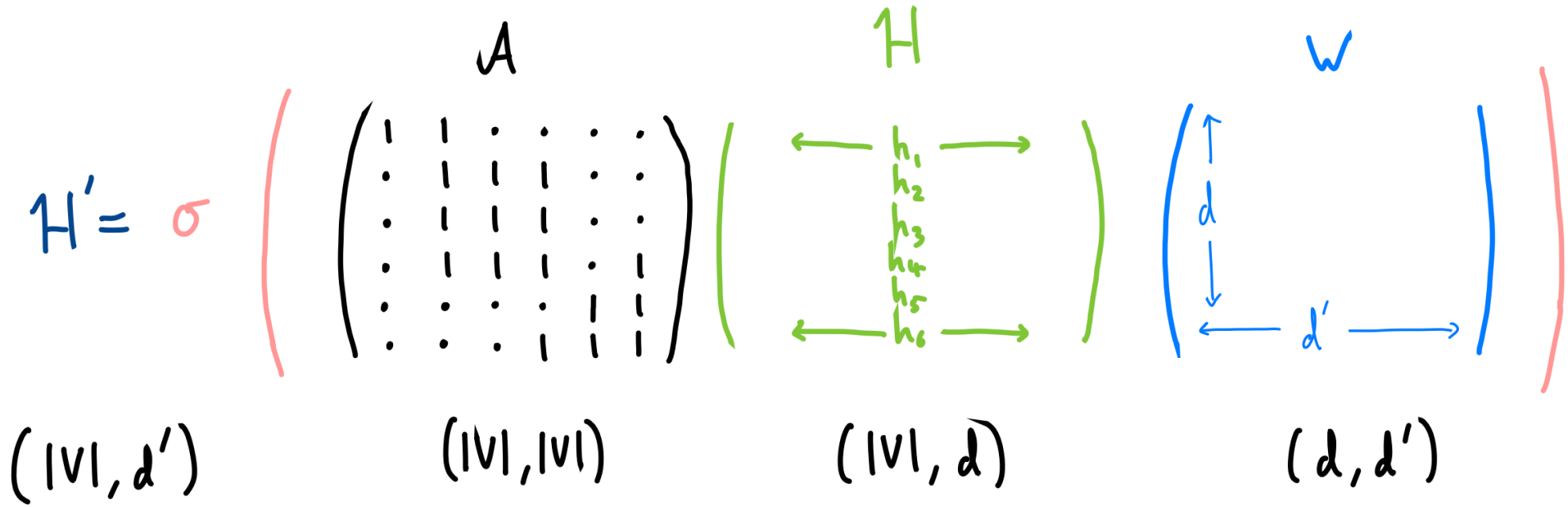
Pietro Liò
Department of Computer Science and Technology
University of Cambridge
pietro.lio@cst.cam.ac.uk

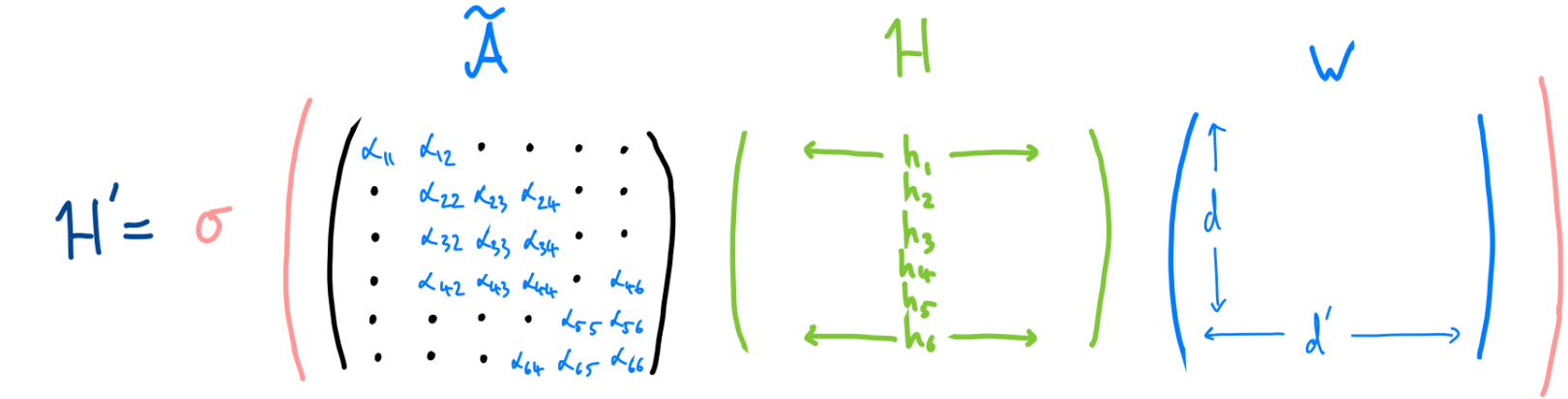
Yoshua Bengio
Montréal Institute for Learning Algorithms
yoshua.umontreal@gmail.com

Graph Attention Networks

Attention GNNs

α_{ij} = "attention between node i and node j"

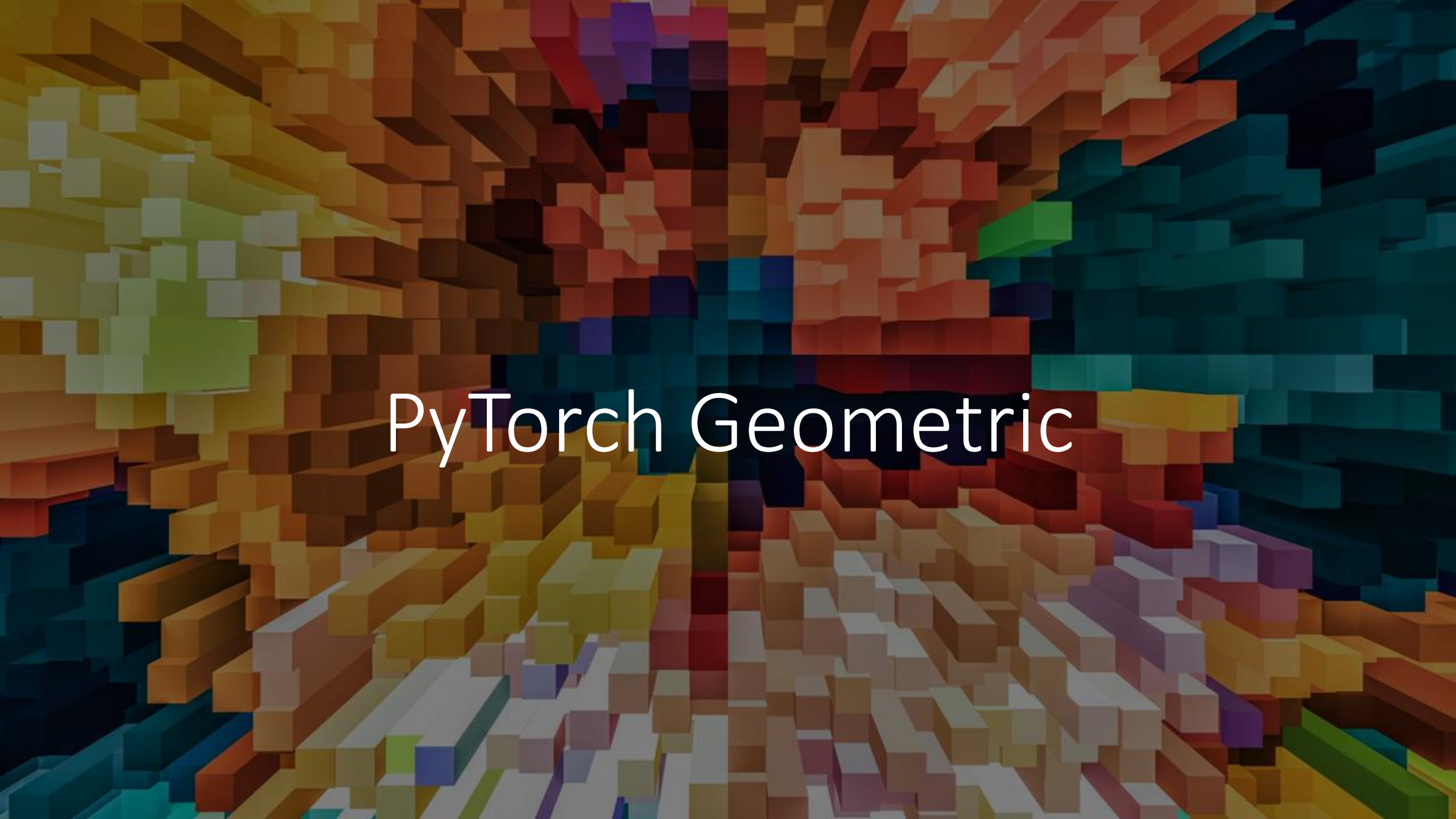




$$h_i' = \sigma\left(\sum_{j \in E(i)} \alpha_{ij}\, W \cdot h_j\right)$$

Similarity to LMs

the quick brown fox jumped


PyTorch Geometric

Karate Club

An Information Flow Model for Conflict and Fission in Small Groups[1]

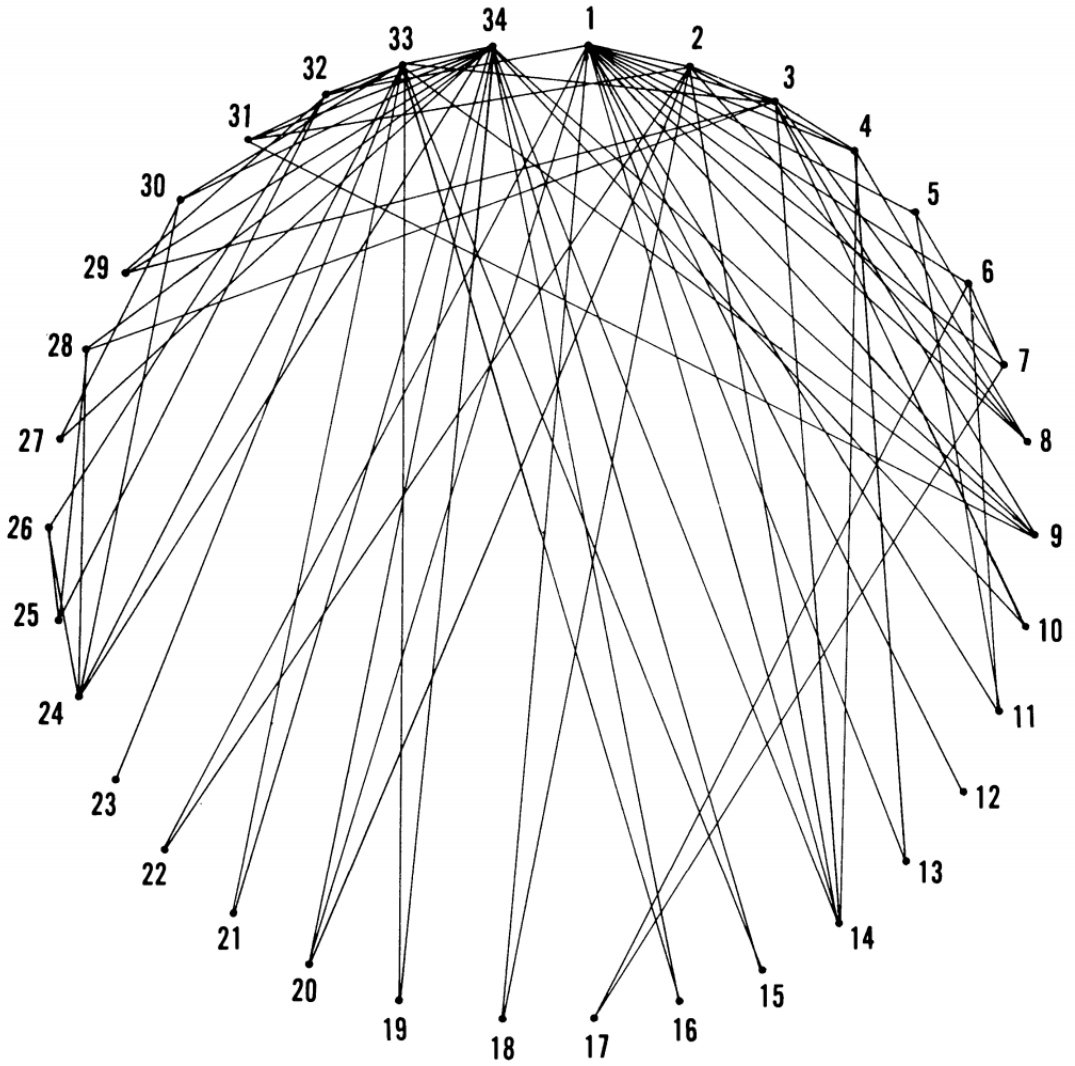
WAYNE W. ZACHARY

Data from a voluntary association are used to construct a new formal model for a traditional anthropological problem, fission in small groups. The process leading to fission is viewed as an unequal flow of sentiments and information across the ties in a social network. This flow is unequal because it is uniquely constrained by the contextual range and sensitivity of each relationship in the network. The subsequent differential sharing of sentiments leads to the formation of subgroups with more internal stability than the group as a whole, and results in fission. The Ford-Fulkerson labeling algorithm allows an accurate prediction of membership in the subgroups and of the locus of the fission to be made from measurements of the potential for information flow across each edge in the network. Methods for measurement of potential information flow are discussed, and it is shown that all appropriate techniques will generate the same predictions.

THE PROBLEM OF HOW and why fission takes place in small bounded groups has long been a central issue in social anthropology, even though the small groups studied have often been described under some other rubric, such as kinship. Fission in kinship groups has been studied from a variety of perspectives, especially those of descent theory (e.g. Evans-Pritchard 1940; Middleton and Tait 1958; Peters 1960; Forde 1964), and ecological adaptation (e.g. Reay 1967; Kelly 1968; Rappaport 1969; Nelson 1971). Another type of small bounded group frequently studied by anthropologists is the voluntary association (e.g.

[An Information Flow Model for Conflict and Fission in Small Groups - Zachary]

FIGURE 1
Social Network Model of Relationships in the Karate Club



This is the graphic representation of the social relationships among the 34 individuals in the karate club. A line is drawn between two points when the two individuals being represented consistently interacted in contexts outside those of karate classes, workouts, and club meetings. Each such line drawn is referred to as an edge.

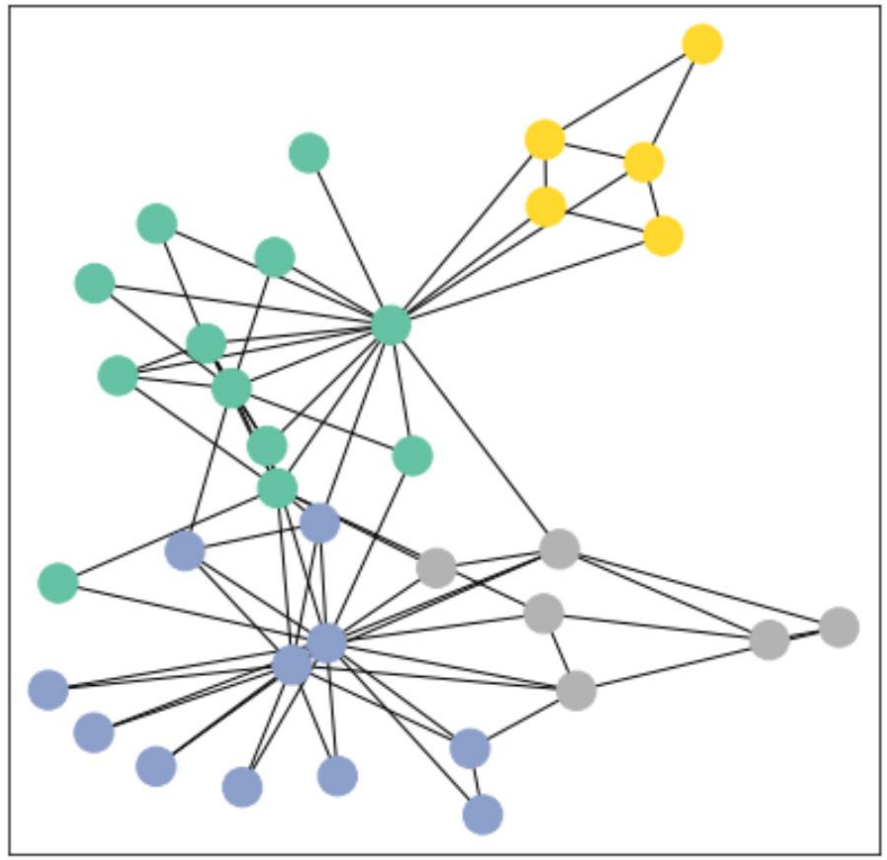
```
[ ]  from torch_geometric.datasets import KarateClub

     dataset = KarateClub()
     print(f'Dataset: {dataset}:')
     print('======================')
     print(f'Number of graphs: {len(dataset)}')
     print(f'Number of features: {dataset.num_features}')
     print(f'Number of classes: {dataset.num_classes}')
```

```
Dataset: KarateClub():
======================
Number of graphs: 1
Number of features: 34
Number of classes: 4
```

```
data = dataset[0]  # Get the first graph object.

print(data)
print('==============================================================')

# Gather some statistics about the graph.
print(f'Number of nodes: {data.num_nodes}')
print(f'Number of edges: {data.num_edges}')
print(f'Average node degree: {data.num_edges / data.num_nodes:.2f}')
print(f'Number of training nodes: {data.train_mask.sum()}')
print(f'Training node label rate: {int(data.train_mask.sum()) / data.num_nodes:.2f}')
print(f'Contains isolated nodes: {data.contains_isolated_nodes()}')
print(f'Contains self-loops: {data.contains_self_loops()}')
print(f'Is undirected: {data.is_undirected()}')
```

```
Data(edge_index=[2, 156], train_mask=[34], x=[34, 34], y=[34])
==============================================================
Number of nodes: 34
Number of edges: 156
Average node degree: 4.59
Number of training nodes: 4
Training node label rate: 0.12
Contains isolated nodes: False
Contains self-loops: False
Is undirected: True
```

```
from torch_geometric.utils import to_networkx

G = to_networkx(data, to_undirected=True)
visualize(G, color=data.y)
```



[Pytorch geometric docs](#) / [example notebook](#)

For this, we will use on of the most simple GNN operators, the **GCN layer** (Kipf et al. (2017)), which is defined as

$$\mathbf{x}_v^{(\ell+1)} = \mathbf{W}^{(\ell+1)} \sum_{w \in \mathcal{N}(v) \cup \{v\}} \frac{1}{c_{w,v}} \cdot \mathbf{x}_w^{(\ell)}$$

where $\mathbf{W}^{(\ell+1)}$ denotes a trainable weight matrix of shape `[num_output_features, num_input_features]` and $c_{w,v}$ refers to a fixed normalization coefficient for each edge.

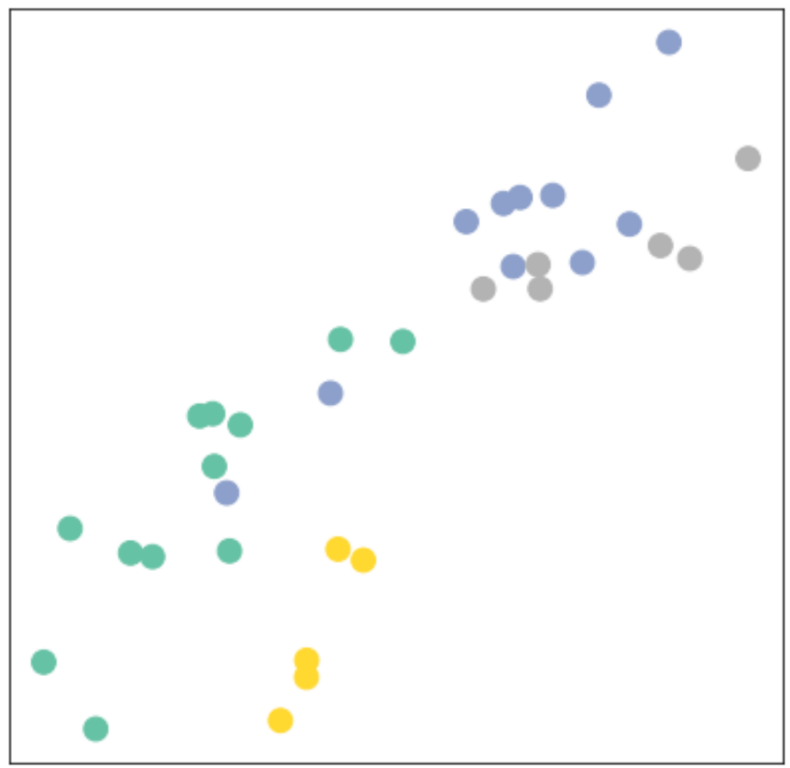
```python
import torch
from torch.nn import Linear
from torch_geometric.nn import GCNConv


class GCN(torch.nn.Module):
    def __init__(self):
        super(GCN, self).__init__()
        torch.manual_seed(12345)
        self.conv1 = GCNConv(dataset.num_features, 4)
        self.conv2 = GCNConv(4, 4)
        self.conv3 = GCNConv(4, 2)
        self.classifier = Linear(2, dataset.num_classes)

    def forward(self, x, edge_index):
        h = self.conv1(x, edge_index)
        h = h.tanh()
        h = self.conv2(h, edge_index)
        h = h.tanh()
        h = self.conv3(h, edge_index)
        h = h.tanh()  # Final GNN embedding space.

        # Apply a final (linear) classifier.
        out = self.classifier(h)

        return out, h
```

Embedding the Karate Club Network

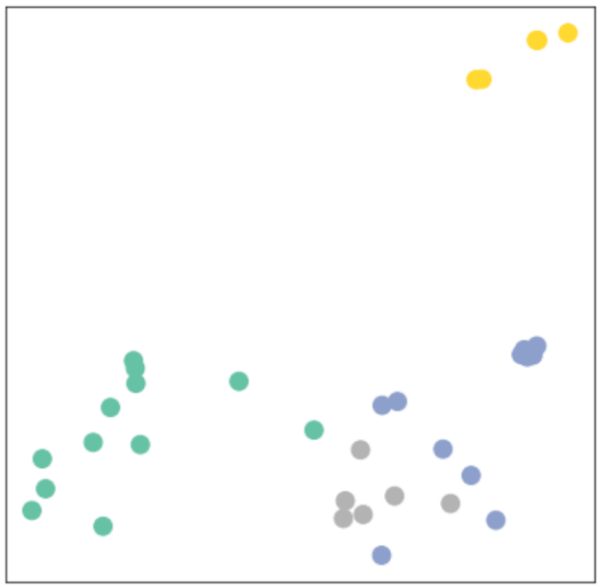
Let's take a look at the node embeddings produced by our GNN. Here, we pass in the initial node features `x` and the graph connectivity information `edge_index` to the model, and visualize its 2-dimensional embedding.

```
[ ]  model = GCN()

     _, h = model(data.x, data.edge_index)
     print(f'Embedding shape: {list(h.shape)}')

     visualize(h, color=data.y)
```

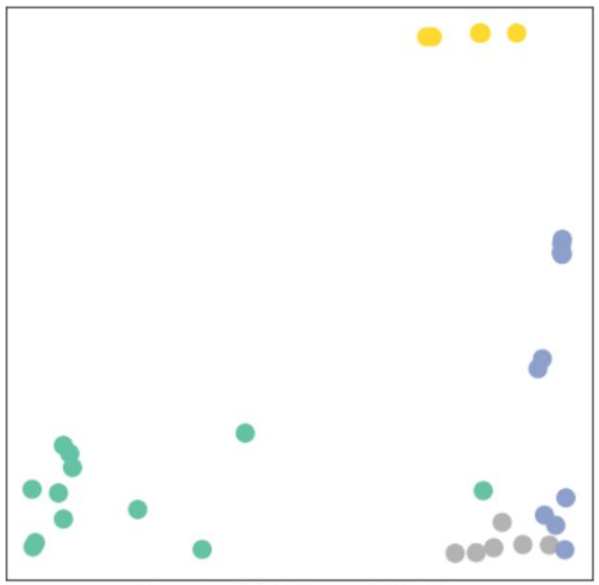
```
Embedding shape: [34, 2]
```



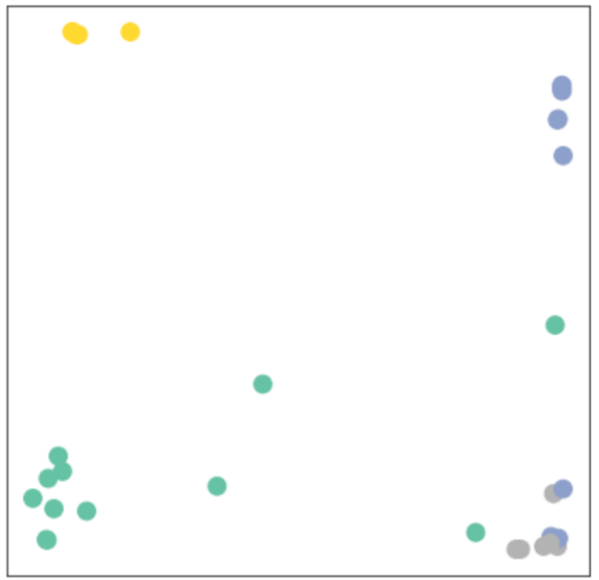
```python
model = GCN()
criterion = torch.nn.CrossEntropyLoss()  # Define loss criterion.
optimizer = torch.optim.Adam(model.parameters(), lr=0.01)  # Define optimizer.

def train(data):
    optimizer.zero_grad()  # Clear gradients.
    out, h = model(data.x, data.edge_index)  # Perform a single forward pass.
    loss = criterion(out[data.train_mask], data.y[data.train_mask])  # Compute the loss solely based on the training nodes.
    loss.backward()  # Derive gradients.
    optimizer.step()  # Update parameters based on gradients.
    return loss, h

for epoch in range(401):
    loss, h = train(data)
    if epoch % 10 == 0:
        visualize(h, color=data.y, epoch=epoch, loss=loss)
        time.sleep(0.3)
```



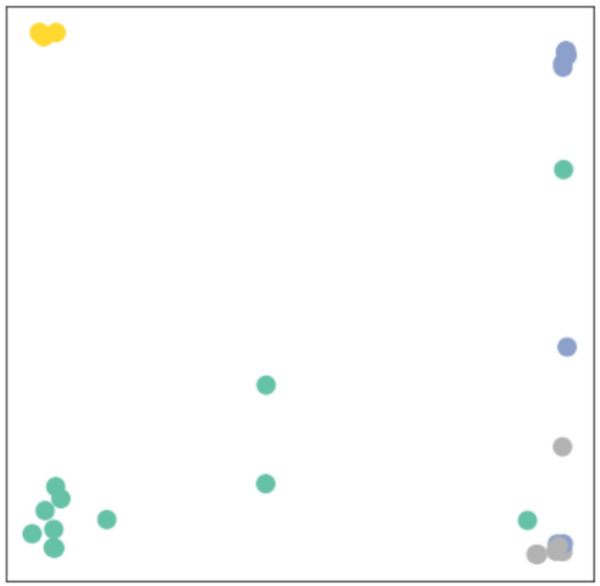
Epoch: 50, Loss: 0.8134



Epoch: 100, Loss: 0.4249



Epoch: 200, Loss: 0.0916



Epoch: 300, Loss: 0.0340

Word of caution

- https://github.com/rusty1s/pytorch_geometric/issues/1080

Undefined symbol: _ZN5torch3jit17parseSchemaOrNameERKSs #1080

References / Resources

2021.08.19 Graph Attention Networked.pdf

- Graph Attention Networks
- Google Colab Notebook
- Pytorch geometric docs