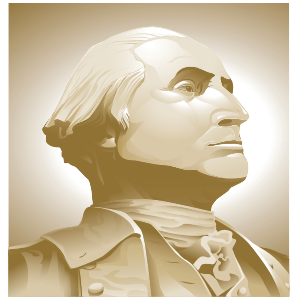
Graph Neural Network
Lecture 2



THE GEORGE WASHINGTON UNIVERSITY

WASHINGTON, DC

Table of Contents

What is PyTorch Geometric?

- **PyG** (PyTorch Geometric) is a library for deep learning on graphs built on PyTorch.
- Provides tools for graph creation, transformations, and GNN layers.
- Efficient handling of large-scale graphs.
- Installation:
  - `pip install torch_geometric`
  - `conda install pyg c pyg`

Key Features

- **Data Handling**: Easy-to-use data structures for graphs.
- **Predefined Layers**: GCN, GAT, GraphSAGE, etc.
- **Datasets**: Built-in datasets like Cora, Citeseer, and more.
- **Extensibility**: Custom layers and datasets.

Table of Contents

Creating a simple graph

```
1  import torch
2  from torch_geometric.data import Data
3
4  # Define node features and edges
5  x = torch.tensor([[1], [2], [3], [4]],
6                   dtype=torch.float)
7  edge_index = torch.tensor([[0, 1, 2, 3],
8                             [1, 0, 3, 2]],
9                             dtype=torch.long)
10 # Create a graph data object
11 data = Data(x=x, edge_index=edge_index)
12 print(data)
```

Accessing Graph Properties

```
1 print(f'Number of nodes: {data.num_nodes}')
2 print(f'Number of edges: {data.num_edges}')
3 print(f'Has isolated nodes: {data.has_isolated_nodes()
4 print(f'Is undirected: {data.is_undirected()}')
```

Adding Edge Weights

```
edge_attr = torch.tensor([[0.5], [0.8], [0.3], [1.0]],
                         dtype=torch.float)
data.edge_attr = edge_attr
print(data)
```

Convert to Undirected Graph

```
1 from torch_geometric.utils import to_undirected
2 edge_index = to_undirected(edge_index)
3 print(edge_index)
```

Using Built-in Datasets

```
1 from torch_geometric.datasets import Planetoid
2
3 # Load the Cora dataset
4 dataset = Planetoid(root='/tmp/Cora', name='Cora')
5 data = dataset[0]  # Get the first graph
6
7 print(f'Number of nodes: {data.num_nodes}')
8 print(f'Number of edges: {data.num_edges}')
9 print(f'Number of classes: {dataset.num_classes}')
```

Table of Contents

Batching Graphs

- PyG supports batching multiple graphs into a single graph object.
- Useful for training on datasets with multiple small graphs.

```
from torch_geometric.data import DataLoader

loader = DataLoader(dataset,
                    batch_size=32,
                    shuffle=True)
for batch in loader:
    print(batch)
```

- PyG provides utilities for graph transformations (e.g., normalization, augmentation).

```
1 from torch_geometric.transforms import NormalizeFeature
2
3 dataset = Planetoid(root='/tmp/Cora', name='Cora',
4                      transform=NormalizeFeatures())
5 data = dataset[0]
6 print(data.x[0])  # Normalized node features
```

Using a GCN Layer

```
1  from torch_geometric.nn import GCNConv
2  import torch.nn.functional as F
3
4  class GCN(torch.nn.Module):
5      def __init__(self):
6          super().__init__()
7          self.conv1 = GCNConv(1, 16)
8          self.conv2 = GCNConv(16, 2)
9
10     def forward(self, x, edge_index):
11         x = self.conv1(x, edge_index)
12         x = F.relu(x)
13         x = self.conv2(x, edge_index)
14         return x
```

Defining a Custom GNN Layer

```
1 from torch_geometric.nn import MessagePassing
2
3 class CustomGNN(MessagePassing):
4     def __init__(self):
5         super().__init__(aggr='mean')
6
7     def forward(self, x, edge_index):
8         return self.propagate(edge_index, x=x)
```

Training a GNN Model

```
model = GCN()
optimizer = torch.optim.Adam(model.parameters(),
                             lr=0.01)

for epoch in range(100):
    optimizer.zero_grad()
    out = model(data.x, data.edge_index)
    loss = F.mse_loss(out, torch.rand((4, 2)))
    loss.backward()
    optimizer.step()
```

Table of Contents

Conclusion

- PyTorch Geometric simplifies working with GNNs.
- Supports various graph neural network architectures.
- Efficient for large-scale graph data processing.