Graph Neural Network
Lecture 2

THE GEORGE
WASHINGTON
UNIVERSITY

WASHINGTON, DC

Table of Contents

1. Introduction to PyG
2. Basic Code Examples
3. Advanced Features
4. Conclusion

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

1. Introduction to PyG

2. Basic Code Examples

3. Advanced Features

4. Conclusion

Creating a simple graph

```python
import torch
from torch_geometric.data import Data

# Define node features and edges
x = torch.tensor([[1], [2], [3], [4]],
                 dtype=torch.float)
edge_index = torch.tensor([[0, 1, 2, 3],
                           [1, 0, 3, 2]],
                           dtype=torch.long)
# Create a graph data object
data = Data(x=x, edge_index=edge_index)
print(data)
```

Accessing Graph Properties

```
print(f’Number of nodes: {data.num_nodes}’)
print(f’Number of edges: {data.num_edges}’)
print(f’Has isolated nodes: {data.has_isolated_nodes()
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

```python
from torch_geometric.utils import to_undirected
edge_index = to_undirected(edge_index)
print(edge_index)
```

Using Built-in Datasets

```python
from torch_geometric.datasets import Planetoid

# Load the Cora dataset
dataset = Planetoid(root=’/tmp/Cora’, name=’Cora’)
data = dataset[0]  # Get the first graph

print(f’Number of nodes: {data.num_nodes}’)
print(f’Number of edges: {data.num_edges}’)
print(f’Number of classes: {dataset.num_classes}’)
```

Table of Contents

1. Introduction to PyG
2. Basic Code Examples
3. Advanced Features
4. Conclusion

Batching Graphs

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

```python
from torch_geometric.data import DataLoader

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

Graph Transformations

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

```
from torch_geometric.transforms import NormalizeFeature

dataset = Planetoid(root=’/tmp/Cora’, name=’Cora’,
                    transform=NormalizeFeatures())
data = dataset[0]
print(data.x[0])  # Normalized node features
```

Using a GCN Layer

```python
from torch_geometric.nn import GCNConv
import torch.nn.functional as F

class GCN(torch.nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = GCNConv(1, 16)
        self.conv2 = GCNConv(16, 2)

    def forward(self, x, edge_index):
        x = self.conv1(x, edge_index)
        x = F.relu(x)
        x = self.conv2(x, edge_index)
        return x
```

Defining a Custom GNN Layer

```python
from torch_geometric.nn import MessagePassing

class CustomGNN(MessagePassing):
    def __init__(self):
        super().__init__(aggr=’mean’)

    def forward(self, x, edge_index):
        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

1. Introduction to PyG
2. Basic Code Examples
3. Advanced Features
4. Conclusion

Conclusion

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