香港科技大学(广州)
THE HONG KONG
UNIVERSITY OF SCIENCE AND
TECHNOLOGY (GUANGZHOU)

Graph Neural Networks

Xing Daiyan

50015641

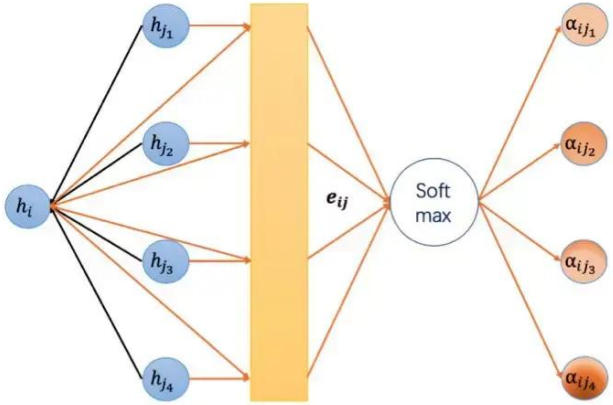
dxing004@connect.hkust-gz.edu.cn

Notebook| GCN&GAT

- Graph Convolutional Network (GCN)
  - GCN layer employs a uniform weight for convolutional operations, meaning that each neighboring node's contribution to the target node is weighted equally.
  - It performs a linear combination of neighboring node features with a fixed weight matrix.
  - GCN layer's weights are fixed and not influenced by the content of the input data.
- Graph Attention Network (GAT)
  - GAT layer utilizes the attention mechanism to compute weighted contributions from each neighboring node to a target node.
  - It allows each node to dynamically assign different weights to its neighbors' features based on learned attention scores.
  - GAT introduces the concept of multiple heads, enabling parallel processing of multiple attention mechanisms to capture different relationships.

$$h_i^{(l+1)} = \sigma\left(\sum_{j\in\mathcal{N}(i)} \frac{1}{c_{ij}} W^{(l)} h_j^{(l)}\right),$$

$$c_{ij} = \sqrt{|N(i)|}\sqrt{|N(j)|}$$



$$z_i^{(l)} = W^{(l)} h_i^{(l)}, \quad (1)$$

$$e_{ij}^{(l)} = \text{LeakyReLU}(\vec{a}^{(l)^T}(z_i^{(l)} \| z_j^{(l)})), \quad (2)$$

$$\alpha_{ij}^{(l)} = \frac{\exp(e_{ij}^{(l)})}{\sum_{k\in\mathcal{N}(i)} \exp(e_{ik}^{(l)})}, \quad (3)$$

$$h_i^{(l+1)} = \sigma\left(\sum_{j\in\mathcal{N}(i)} \alpha_{ij}^{(l)} z_j^{(l)}\right), \quad (4)$$

The core difference between GAT and GCN lies in how they gather and aggregate feature representations from neighboring nodes at a distance of 1. The key distinction is how GAT uses attention to weigh neighbor features differently, while GCN uses a fixed weight for all neighbor features at distance 1. This fundamental difference allows GAT to capture more nuanced and adaptive relationships in graph data.

https://openreview.net/pdf?id=SJU4ayYgl
https://tkipf.github.io/graph-convolutional-networks/

Notebook| GraphConv

- GraphConv（Graph Convolution）
  - GraphConv is a generic term used to describe various types of graph convolutional layers, rather than referring to a specific method.
  - These methods may include GCN, GAT, as well as other variants and improvements of graph convolutional layers.
  - The specific implementation and performance of GraphConv may vary depending on the researchers and the tasks at hand.

Additionally to GCN and GAT, we added the layer `geom_nn.GraphConv` (documentation). GraphConv is a GCN with a separate weight matrix for the self-connections. Mathematically, this would be:

$$\mathbf{x}_i^{(l+1)} = \mathbf{W}_1^{(l+1)}\mathbf{x}_i^{(l)} + \mathbf{W}_2^{(\ell+1)} \sum_{j \in \mathcal{N}_i} \mathbf{x}_j^{(l)}$$

In this formula, the neighbor's messages are added instead of averaged. However, PyTorch Geometric provides the argument `aggr` to switch between summing, averaging, and max pooling.

Notebook| Tasks on graph-structured data

香港科技大学(广州)
THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY (GUANGZHOU)

- Node-level tasks: Semi-supervised node classification
  - GNNModel(nn.Module)
  - MLPModel(nn.Module)
  - NodeLevelGNN(pl.LightningModule)
- Edge-level tasks: Link prediction
- Graph-level tasks: Graph classification
  - GraphGNNModel(nn.Module)
  - GraphLevelGNN(pl.LightningModule)

Notebook| Tasks on graph-structured data

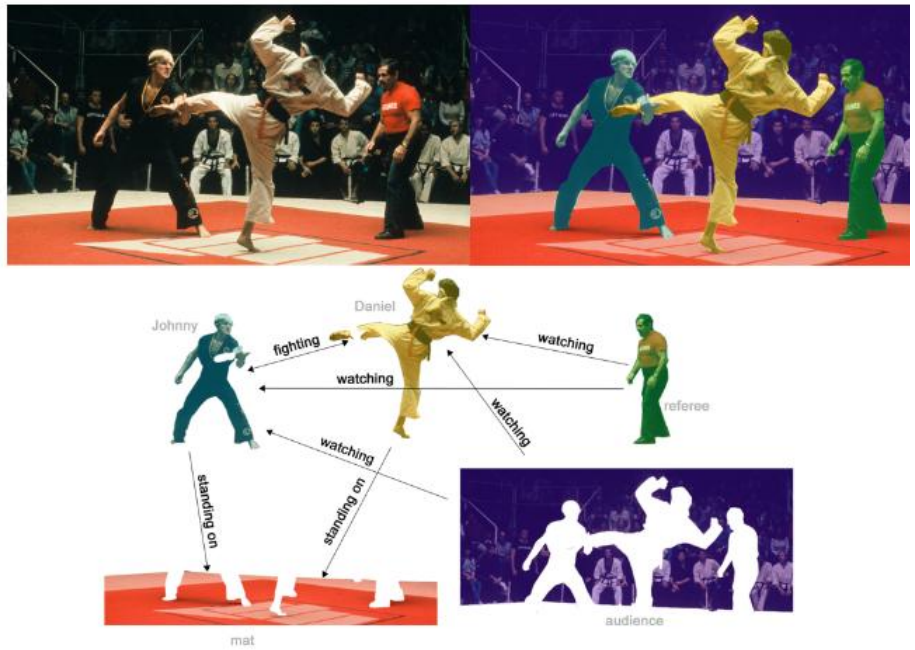
香港科技大学(广州)
THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY (GUANGZHOU)

- Node-level tasks: Semi-supervised node classification
  - GNNModel(nn.Module)
    - A class representing a Graph Neural Network (GNN) model designed for node classification tasks on graph data
    - Constructs a model with multiple GNN layers based on the given number of layers num_layers and the specified GNN layer type layer_name.
    - Includes an output layer with a GNN layer but without ReLU and dropout layers.
    - The forward method takes input feature tensor x and graph edge indices edge_index and passes the input data through multiple GNN layers.
  - MLPModel(nn.Module)
    - A class representing a Multi-Layer Perceptron (MLP) model designed for node classification tasks
    - Constructs the model with multiple hidden layers, where each hidden layer includes a linear layer, ReLU activation function, and dropout layer.
    - Includes an output layer with a linear layer but without ReLU and dropout layers.
    - The forward method takes input feature tensor x and passes it through multiple linear layers.
  - NodeLevelGNN(pl.LightningModule)
    - A PyTorch Lightning model class used for training and evaluating node classifiers
    - Chooses to use either GNNModel or MLPModel as the underlying model based on the specified model name.
    - The configure_optimizers method defines the optimizer, where SGD is used in this case.

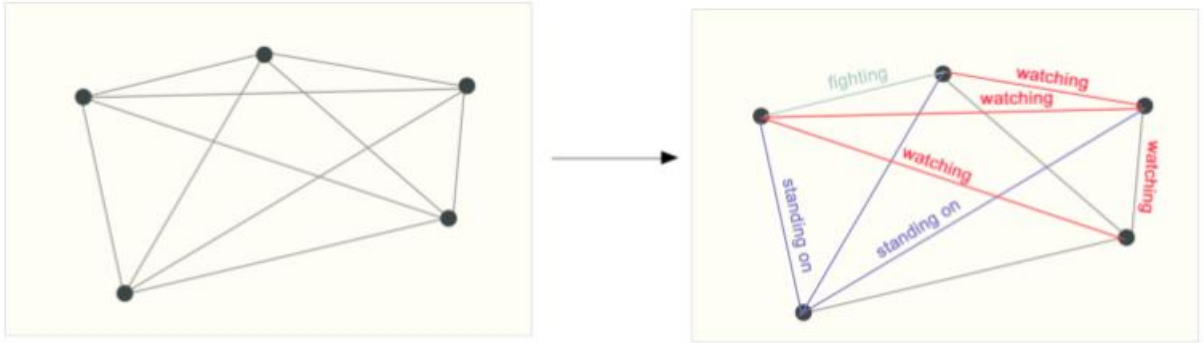
Notebook| Tasks on graph-structured data

- Edge-level tasks: Link prediction

One example of edge-level inference is in image scene understanding. Beyond identifying objects in an image, deep learning models can be used to predict the relationship between them. We can phrase this as an edge-level classification: given nodes that represent the objects in the image, we wish to predict which of these nodes share an edge or what the val of that edge is. If we wish to discover connections between entities, we could consider the graph fully connected and based on their predicted value prune edges to arrive at a sparse graph.



In (b), above, the original image (a) has been segmented into five entities: each of the fighters, the referee, the audience and the mat. (C) shows the relationships between these entities.



Input: fully connected graph, unlabeled edges

Output: labels for edges

On the left we have an initial graph built from the previous visual scene. On the right is a possible edge-labeling of this graph when some connections were pruned based on the model's output.

Notebook| Tasks on graph-structured data

香港科技大学(广州)
THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY (GUANGZHOU)

- Graph-level tasks: Graph classification
  - GraphGNNModel(nn.Module)
    - A class representing a Graph Neural Network (GNN) model designed for node classification tasks
    - compared to GNNModel:
      - dp_rate_linear: Dropout rate before the linear layer, typically higher than the dropout rate inside the GNN.
      - GNNModel(nn.Module) class focuses solely on constructing and forward propagation of the GNN model, while GraphGNNModel(nn.Module) class extends it by adding a head for additional processing, global pooling operation, and extra constructor parameters.
  - GraphLevelGNN(pl.LightningModule)
    - A PyTorch Lightning model class used for training and evaluating node classifiers
    - Chooses to use GraphGNNModel as the underlying model
    - The configure_optimizers method defines the optimizer, where AdamW is used in this case.

Improve the Performance of the Provided GNNs

香港科技大学(广州)
THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY (GUANGZHOU)

- Adjust Network Architecture:
  - Increase or decrease the number of graph convolutional layers.
  - Tune the number of hidden units in each layer.
- Optimize the Training Process:
  - Adjust the learning rate and optimizer parameters.
  - Implement early stopping to prevent overfitting.
- Data Preprocessing:
  - Normalize or standardize the input features.
  - Apply data augmentation techniques, such as perturbing the graph data or adding synthetic nodes.
- Regularization and Dropout:
  - Add dropout layers between the GNN layers.
  - Apply L1 or L2 regularization techniques.
- Advanced Techniques:
  - Utilize advanced GNN layers like Graph Attention Networks (GAT).
  - Incorporate graph pooling techniques.

Improve the Performance of the Provided GNNs

香港科技大学(广州)
THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY (GUANGZHOU)

```python
class GNNModel(nn.Module):
    def __init__(self, c_in, c_hidden, c_out, num_layers=2, layer_name="GCN", dp_rate=0.1, **kwargs):
        super().__init__()
        gnn_layer = gnn_layer_by_name[layer_name]

        layers = []
        in_channels, out_channels = c_in, c_hidden
        for l_idx in range(num_layers-1):
            layers += [
                gnn_layer(in_channels=in_channels,out_channels=out_channels, **kwargs),
                nn.ReLU(inplace=True),
                nn.Dropout(dp_rate)
            ]
            in_channels = c_hidden
        layers += [gnn_layer(in_channels=in_channels, out_channels=c_out, **kwargs)]
        self.layers = nn.ModuleList(layers)

    def forward(self, x, edge_index):
        for l in self.layers:
            if isinstance(l, geom_nn.MessagePassing):
                x = l(x, edge_index)
            else:
                x = l(x)
        return x
```

```python
class ImprovedGNNModel(nn.Module):
    def __init__(self, c_in, c_hidden, c_out, num_layers=2, layer_name="GCN", dp_rate=0.01, **kwargs):
        super().__init__()
        gnn_layer = gnn_layer_by_name[layer_name]

        self.layers = nn.ModuleList()
        in_channels = c_in
        for l_idx in range(num_layers - 1):
            self.layers.append(gnn_layer(in_channels=in_channels, out_channels=c_hidden, **kwargs))
            self.layers.append(nn.BatchNorm1d(c_hidden))
            self.layers.append(nn.LeakyReLU(0.2))
            self.layers.append(nn.Dropout(dp_rate))
            in_channels = c_hidden

        self.layers.append(gnn_layer(in_channels=in_channels, out_channels=c_out, **kwargs))

        if c_in != c_out:
            self.residual = nn.Linear(c_in, c_out)
        else:
            self.residual = None
```

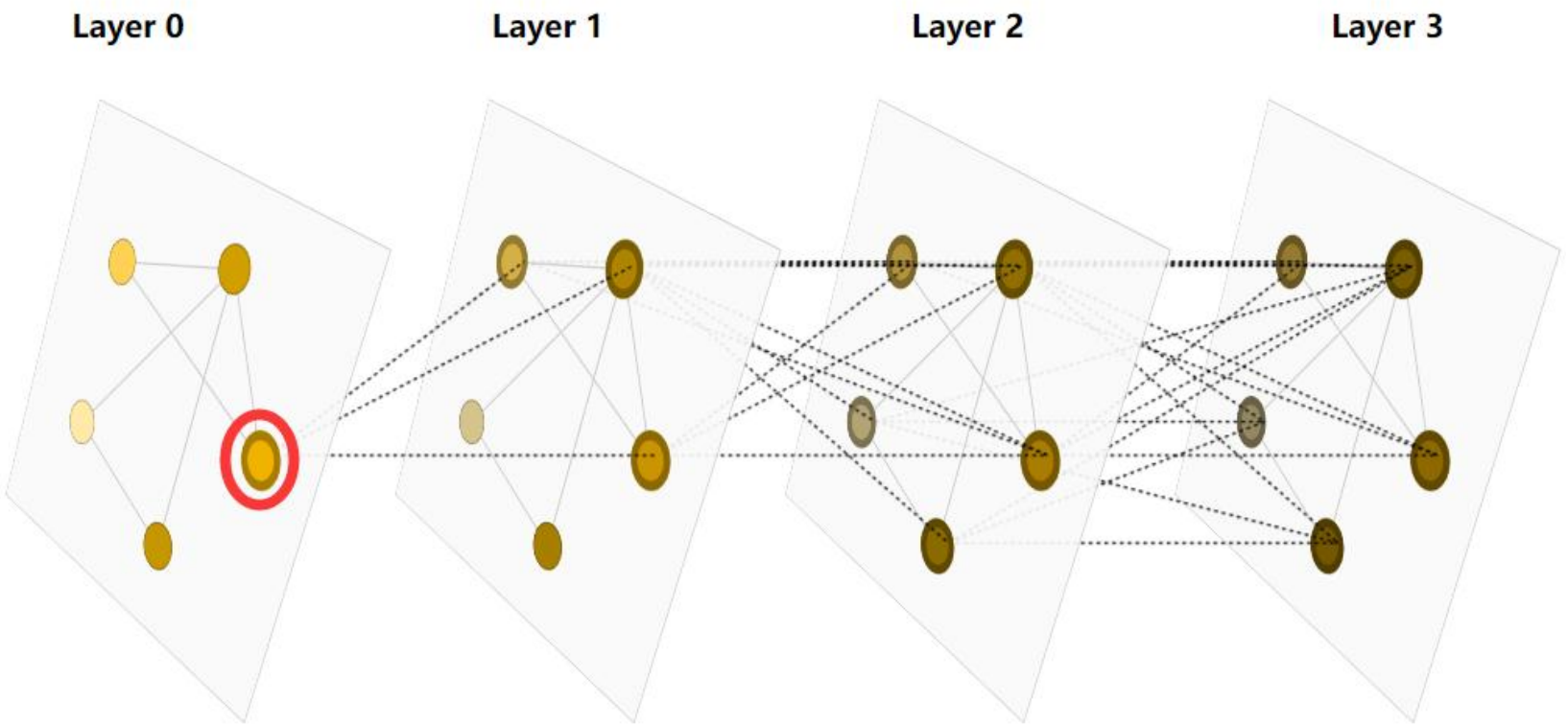
```python
def configure_optimizers(self):
    '''# We use SGD here, but Adam works as well
    optimizer = optim.SGD(self.parameters(), lr=0.1, momentum=0.9, weight_decay=2e-3)
    #optimizer = optim.SGD(self.parameters(), lr=0.01, momentum=0.9, weight_decay=2e-3)
    return optimizer'''
    optimizer = optim.Adam(self.parameters(), lr=0.1, weight_decay=2e-3)
    scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=50, gamma=0.15)
    return [optimizer], [scheduler]
```

- **Hyperparameter tuning:**
  - Learning Rate: Using smaller learning rates such as 0.01 or 0.001
  - Hidden Units: Increasing or decreasing the number of hidden units in the layers may help improve performance.
  - Number of Layers: Adding more layers can increase the model's capacity
- **Optimizer selection**:
  - Optimizer: SGD, Adam
- **Regularization strategies**:
  - Weight Decay: Adjusting the weight decay parameter can help prevent overfitting.
  - Dropout Rate: Tuning the dropout rate can help the model learn more robust features.
- **Residual connections**:
- **Data preprocessing**:
  - Feature Normalization: Ensure that the input features are properly normalized or standardized
  - Data Augmentation: For graph data, such as adding or removing edges, perturbing nodes, etc.

```
Train accuracy: 100.00%
Val accuracy:    76.20%
Test accuracy:   80.20%
```

```
Train accuracy: 100.00%
Val accuracy:    77.60%
Test accuracy:   78.60%
```

Paper Reading | A Gentle Introduction to Graph Neural Networks

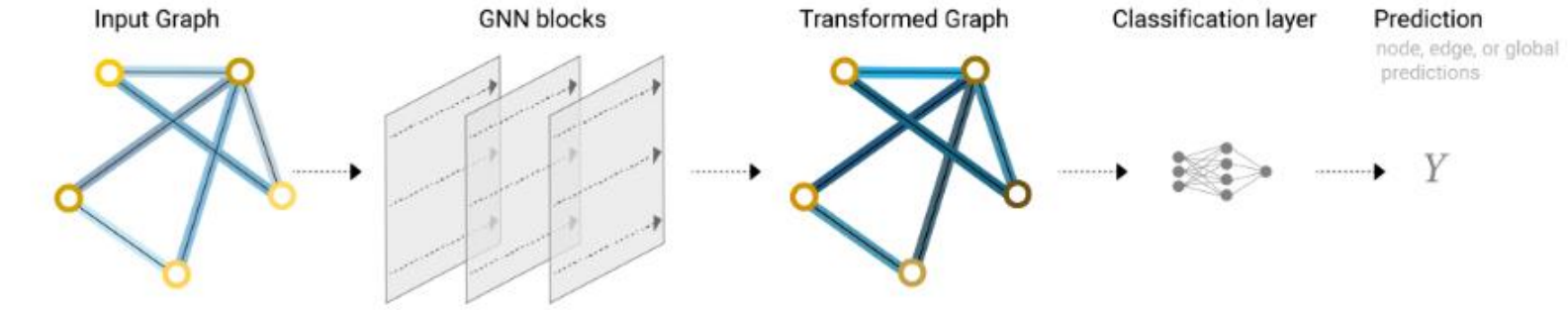


Hover over a node in the diagram below to see how it accumulates information from nodes around it through the layers of the network.

https://distill.pub/2021/understanding-gnns/
https://distill.pub/2021/gnn-intro/
Sánchez-Lengeling, Benjamín et al. “A Gentle Introduction to Graph Neural Networks.” Distill (2021): n. pag.

Paper reading | A Gentle Introduction to Graph Neural Networks

- GNN is an optimization-based transformation applied to all attributes of a graph, including vertices, edges, and global context. It preserves the symmetric information of the graph, meaning that the overall result remains unchanged when the vertices are reordered.
- GNN uses message passing, a framework for information exchange in neural networks.
- The input to GNN is a graph, and the output is also a graph. It transforms the vectors associated with vertices, edges, and global context, while preserving the connectivity of the graph. The connectivity information between vertices and edges remains unchanged within the graph neural network.



An end-to-end prediction task with a GNN model.

https://www.bilibili.com/video/BV1iT4y1d7zP/ another angel(´･ᴗ･`)
https://www.bilibili.com/read/cv14432698/?jump_opus=1

香港科技大学(广州)
THE HONG KONG
UNIVERSITY OF SCIENCE AND
TECHNOLOGY (GUANGZHOU)

Thank you!

Xing Daiyan

50015641

dxing004@connect.hkust-gz.edu.cn