CS224W - Colab 3 2020

May 25, 2022

1 CS224W - Colab 3

In Colab 2 we constructed GNN models by using PyTorch Geometric built in GCN layer, the `GCNConv`. In this Colab we will implement the **GraphSAGE** (Hamilton et al. (2017)) and **GAT** (Veličković et al. (2018)) layers directly. Then we will run our models on the CORA dataset, which is a standard citation network benchmark dataset.

We will then use DeepSNAP, a Python library assisting efficient deep learning on graphs, to split the graphs in different settings and apply dataset transformations.

At last, using DeepSNAP transductive link prediction split functionality, we will construct a simple GNN model on the edge property predition (link prediction) task.

Note: Make sure to **sequentially run all the cells in each section**, so that the intermediate variables / packages will carry over to the next cell

Have fun on Colab 3 :)

2 Device

You might need to use GPU for this Colab.

Please click `Runtime` and then `Change runtime type`. Then set the `hardware accelerator` to **GPU**.

2.1 Installation

[1]:
```
# !pip install -q torch-scatter -f https://pytorch-geometric.com/whl/torch-1.7.
↪0+cu101.html
# !pip install -q torch-sparse -f https://pytorch-geometric.com/whl/torch-1.7.
↪0+cu101.html
# !pip install -q torch-geometric
# !pip install -q git+https://github.com/snap-stanford/deepsnap.git
```

```
/bin/bash: /home/arch/anaconda3/envs/tf1.15_py3.8_gpu/lib/libtinfo.so.6: no
version information available (required by /bin/bash)
/bin/bash: /home/arch/anaconda3/envs/tf1.15_py3.8_gpu/lib/libtinfo.so.6: no
version information available (required by /bin/bash)
/bin/bash: /home/arch/anaconda3/envs/tf1.15_py3.8_gpu/lib/libtinfo.so.6: no
version information available (required by /bin/bash)
```

```
/bin/bash: /home/arch/anaconda3/envs/tf1.15_py3.8_gpu/lib/libtinfo.so.6: no
version information available (required by /bin/bash)
```

[3]:
```
!pip install torch-scatter -f https://pytorch-geometric.com/whl/torch-1.9.
↪0+cu111.html
!pip install torch-sparse -f https://pytorch-geometric.com/whl/torch-1.9.
↪0+cu111.html
!pip install torch-geometric
!pip install -q git+https://github.com/snap-stanford/deepsnap.git
```

```
/bin/bash: /home/arch/anaconda3/envs/tf1.15_py3.8_gpu/lib/libtinfo.so.6: no
version information available (required by /bin/bash)
Looking in indexes: https://pypi.org/simple, https://pypi.ngc.nvidia.com
Looking in links: https://pytorch-geometric.com/whl/torch-1.9.0+cu111.html
Requirement already satisfied: torch-scatter in
/home/arch/anaconda3/lib/python3.8/site-packages (2.0.7)
/bin/bash: /home/arch/anaconda3/envs/tf1.15_py3.8_gpu/lib/libtinfo.so.6: no
version information available (required by /bin/bash)
Looking in indexes: https://pypi.org/simple, https://pypi.ngc.nvidia.com
Looking in links: https://pytorch-geometric.com/whl/torch-1.9.0+cu111.html
Requirement already satisfied: torch-sparse in
/home/arch/anaconda3/lib/python3.8/site-packages (0.6.9)
Requirement already satisfied: scipy in /home/arch/anaconda3/lib/python3.8/site-
packages (from torch-sparse) (1.6.2)
Requirement already satisfied: numpy<1.23.0,>=1.16.5 in
/home/arch/anaconda3/lib/python3.8/site-packages (from scipy->torch-sparse)
(1.20.1)
/bin/bash: /home/arch/anaconda3/envs/tf1.15_py3.8_gpu/lib/libtinfo.so.6: no
version information available (required by /bin/bash)
Looking in indexes: https://pypi.org/simple, https://pypi.ngc.nvidia.com
Requirement already satisfied: torch-geometric in
/home/arch/anaconda3/lib/python3.8/site-packages (2.0.3)
Requirement already satisfied: networkx in
/home/arch/anaconda3/lib/python3.8/site-packages (from torch-geometric) (2.5)
Requirement already satisfied: rdflib in
/home/arch/anaconda3/lib/python3.8/site-packages (from torch-geometric) (6.1.1)
Requirement already satisfied: pyparsing in
/home/arch/anaconda3/lib/python3.8/site-packages (from torch-geometric) (2.4.7)
Requirement already satisfied: yacs in /home/arch/anaconda3/lib/python3.8/site-
packages (from torch-geometric) (0.1.8)
Requirement already satisfied: googledrivedownloader in
/home/arch/anaconda3/lib/python3.8/site-packages (from torch-geometric) (0.4)
Requirement already satisfied: requests in
/home/arch/anaconda3/lib/python3.8/site-packages (from torch-geometric) (2.25.1)
Requirement already satisfied: pandas in /home/arch/.local/lib/python3.8/site-
packages (from torch-geometric) (1.3.5)
Requirement already satisfied: numpy in /home/arch/anaconda3/lib/python3.8/site-
packages (from torch-geometric) (1.20.1)
```

```
Requirement already satisfied: scikit-learn in
/home/arch/anaconda3/lib/python3.8/site-packages (from torch-geometric) (0.24.1)
Requirement already satisfied: PyYAML in /home/arch/.local/lib/python3.8/site-
packages (from torch-geometric) (6.0)
Requirement already satisfied: jinja2 in /home/arch/.local/lib/python3.8/site-
packages (from torch-geometric) (3.0.3)
Requirement already satisfied: scipy in /home/arch/anaconda3/lib/python3.8/site-
packages (from torch-geometric) (1.6.2)
Requirement already satisfied: tqdm in /home/arch/anaconda3/lib/python3.8/site-
packages (from torch-geometric) (4.59.0)
Requirement already satisfied: MarkupSafe>=2.0 in
/home/arch/anaconda3/lib/python3.8/site-packages (from jinja2->torch-geometric)
(2.1.1)
Requirement already satisfied: decorator>=4.3.0 in
/home/arch/.local/lib/python3.8/site-packages (from networkx->torch-geometric)
(5.1.1)
Requirement already satisfied: python-dateutil>=2.7.3 in
/home/arch/.local/lib/python3.8/site-packages (from pandas->torch-geometric)
(2.8.2)
Requirement already satisfied: pytz>=2017.3 in
/home/arch/anaconda3/lib/python3.8/site-packages (from pandas->torch-geometric)
(2021.1)
Requirement already satisfied: six>=1.5 in
/home/arch/anaconda3/lib/python3.8/site-packages (from python-
dateutil>=2.7.3->pandas->torch-geometric) (1.15.0)
Requirement already satisfied: setuptools in
/home/arch/anaconda3/lib/python3.8/site-packages (from rdflib->torch-geometric)
(52.0.0.post20210125)
Requirement already satisfied: isodate in
/home/arch/anaconda3/lib/python3.8/site-packages (from rdflib->torch-geometric)
(0.6.1)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in
/home/arch/anaconda3/lib/python3.8/site-packages (from requests->torch-
geometric) (1.26.4)
Requirement already satisfied: chardet<5,>=3.0.2 in
/home/arch/anaconda3/lib/python3.8/site-packages (from requests->torch-
geometric) (4.0.0)
Requirement already satisfied: idna<3,>=2.5 in
/home/arch/anaconda3/lib/python3.8/site-packages (from requests->torch-
geometric) (2.10)
Requirement already satisfied: certifi>=2017.4.17 in
/home/arch/anaconda3/lib/python3.8/site-packages (from requests->torch-
geometric) (2021.10.8)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/home/arch/anaconda3/lib/python3.8/site-packages (from scikit-learn->torch-
geometric) (2.1.0)
Requirement already satisfied: joblib>=0.11 in
/home/arch/anaconda3/lib/python3.8/site-packages (from scikit-learn->torch-
```

```
geometric) (1.1.0)
/bin/bash: /home/arch/anaconda3/envs/tf1.15_py3.8_gpu/lib/libtinfo.so.6: no
version information available (required by /bin/bash)
```

[1]:
```python
import torch_geometric
torch_geometric.__version__
```

[1]: '2.0.3'

3 1 GNN Layers

3.1 Implementing Layer Modules

In colab 2, we implemented a network using GCN in node and graph classification tasks. However, the GCN module we used in colab 2 is from the official library. For this problem, we will provide you with a general Graph Neural Network Stack, where you'll be able to plugin your own modules of GraphSAGE and GATs. We will use our implementations to complete node classification on CORA, which is a standard citation network benchmark dataset. In this dataset, nodes correspond to documents and edges correspond to undirected citations. Each node has a class label. The node features are elements of a bag-or-words representation of a document. For the Cora dataset, there are 2708 nodes, 5429 edges, 7 prediction classes for nodes, and 1433 features per node.

3.2 GNN Stack Module

Below is the implementation for a general GNN Module that could plugin any layers, including **GraphSage**, **GAT**, etc. This module is provided for you, and you own **GraphSage** and **GAT** layers will function as components in the GNNStack Module.

[5]:
```python
import torch
import torch_scatter
import torch.nn as nn
import torch.nn.functional as F

import torch_geometric.nn as pyg_nn
import torch_geometric.utils as pyg_utils

from torch import Tensor
from typing import Union, Tuple, Optional
from torch_geometric.typing import (OptPairTensor, Adj, Size, NoneType,
                                    OptTensor)

from torch.nn import Parameter, Linear
from torch_sparse import SparseTensor, set_diag
from torch_geometric.nn.conv import MessagePassing
from torch_geometric.utils import remove_self_loops, add_self_loops, softmax

class GNNStack(torch.nn.Module):
    def __init__(self, input_dim, hidden_dim, output_dim, args, emb=False):
```

```
        super(GNNStack, self).__init__()
        conv_model = self.build_conv_model(args.model_type)
        self.convs = nn.ModuleList()
        self.convs.append(conv_model(input_dim, hidden_dim))
        assert (args.num_layers >= 1), 'Number of layers is not >=1'
        for l in range(args.num_layers-1):
            self.convs.append(conv_model(args.heads * hidden_dim, hidden_dim))

        # post-message-passing
        self.post_mp = nn.Sequential(
            nn.Linear(args.heads * hidden_dim, hidden_dim), nn.Dropout(args.
↪dropout),
            nn.Linear(hidden_dim, output_dim))

        self.dropout = args.dropout
        self.num_layers = args.num_layers

        self.emb = emb

    def build_conv_model(self, model_type):
        if model_type == 'GraphSage':
            return GraphSage
        elif model_type == 'GAT':
            # When applying GAT with num heads > 1, one needs to modify the
            # input and output dimension of the conv layers (self.convs),
            # to ensure that the input dim of the next layer is num heads
            # multiplied by the output dim of the previous layer.
            # HINT: In case you want to play with multiheads, you need to␣
↪change the for-loop when builds up self.convs to be
            # self.convs.append(conv_model(hidden_dim * num_heads,␣
↪hidden_dim)),
            # and also the first nn.Linear(hidden_dim * num_heads, hidden_dim)␣
↪in post-message-passing.
            return GAT

    def forward(self, data):
        x, edge_index, batch = data.x, data.edge_index, data.batch

        for i in range(self.num_layers):
            x = self.convs[i](x, edge_index)
            x = F.relu(x)
            x = F.dropout(x, p=self.dropout)

        x = self.post_mp(x)

        if self.emb == True:
            return x
```

```
        return F.log_softmax(x, dim=1)

    def loss(self, pred, label):
        return F.nll_loss(pred, label)
```

3.3 GraphSage Implementation

Now let's start working on our own implementation of layers! This part is to get you familiar with how to implement Pytorch layer based on Message Passing. You will be implementing the **forward**, **message** and **aggregate** functions.

Generally, the **forward** function is where the actual message passing is conducted. All logic in each iteration happens in **forward**, where we'll call **propagate** function to propagate information from neighbor nodes to central nodes. So the general paradigm will be pre-processing -> propagate -> post-processing.

Recall the process of message passing we introduced in homework 1. **propagate** further calls **message** which transforms information of neighbor nodes into messages, **aggregate** which aggregates all messages from neighbor nodes into one, and **update** which further generates the embedding for nodes in the next iteration.

Our implementation is slightly variant from this, where we'll not explicitly implement **update**, but put the logic for updating nodes in **forward** function. To be more specific, after information is propagated, we can further conduct some operations on the output of **propagate**. The output of **forward** is exactly the embeddings after the current iteration.

In addition, tensors passed to **propagate()** can be mapped to the respective nodes i and j by appending _i or _j to the variable name, .e.g. x_i and x_j. Note that we generally refer to i as the central nodes that aggregates information, and refer to j as the neighboring nodes, since this is the most common notation.

Please find more details in the comments. One thing to note is that we're adding **skip connections** to our GraphSage. Formally, the update rule for our model is described as below:

$$h_v^{(l)} = W_l \cdot h_v^{(l-1)} + W_r \cdot AGG(\{h_u^{(l-1)}, \forall u \in N(v)\}) \tag{1}$$

For simplicity, we use mean aggregations where:

$$AGG(\{h_u^{(l-1)}, \forall u \in N(v)\}) = \frac{1}{|N(v)|} \sum_{u \in N(v)} h_u^{(l-1)} \tag{2}$$

Additionally, ℓ-2 normalization is applied after each iteration.

In order to complete the work correctly, we have to understand how the different functions interact with each other. In **propagate** we can pass in any parameters we want. For example, we pass in x as an parameter:

... = propagate(..., x=($x_{central}$, $x_{neighbor}$), ...)

Here $x_{central}$ and $x_{neighbor}$ represent the features from **central** nodes and from **neighbor** nodes. If we're using the same representations from central and neighbor, then $x_{central}$ and $x_{neighbor}$ could be identical.

Suppose $x_{central}$ and $x_{neighbor}$ are both of shape N * d, where N is number of nodes, and d is dimension of features.

Then in message function, we can take parameters called x_i and x_j. Usually x_i represents "central nodes", and x_j represents "neighbor nodes". Pay attention to the shape here: x_i and x_j are both of shape E * d (**not N!**). x_i is obtained by concatenating the embeddings of central nodes of all edges through lookups from $x_{central}$ we passed in propagate. Similarly, x_j is obtained by concatenating the embeddings of neighbor nodes of all edges through lookups from $x_{neighbor}$ we passed in propagate.

Let's look at an example. Suppose we have 4 nodes, so $x_{central}$ and $x_{neighbor}$ are of shape 4 * d. We have two edges (1, 2) and (3, 0). Thus, x_i is obtained by $[x_{central}[1]^T; x_{central}[3]^T]^T$, and x_j is obtained by $[x_{neighbor}[2]^T; x_{neighbor}[0]^T]^T$

For the following questions, DON'T refer to any existing implementations online.

[6]:
```
class GraphSage(MessagePassing):

    def __init__(self, in_channels, out_channels, normalize = True,
                 bias = False, **kwargs):
        super(GraphSage, self).__init__(**kwargs)

        self.in_channels = in_channels
        self.out_channels = out_channels
        self.normalize = normalize

        self.lin_l = None
        self.lin_r = None


 ↪############################################################################
        # TODO: Your code here!
        # Define the layers needed for the message and update functions below.
        # self.lin_l is the linear transformation that you apply to embedding
        #            for central node.
        # self.lin_r is the linear transformation that you apply to aggregated
        #            message from neighbors.
        # Our implementation is ~2 lines, but don't worry if you deviate from
 ↪this.

        self.lin_l = nn.Linear(self.in_channels, self.out_channels, bias)
        self.lin_r = nn.Linear(self.in_channels, self.out_channels, bias)


 ↪############################################################################
```

```python
        self.reset_parameters()

    def reset_parameters(self):
        self.lin_l.reset_parameters()
        self.lin_r.reset_parameters()

    def forward(self, x, edge_index, size = None):
        """"""

        out = None

        ############################################################################
        # TODO: Your code here!
        # Implement message passing, as well as any post-processing (our update rule).
        # 1. First call propagate function to conduct the message passing.
        #    1.1 See there for more information:
        #        https://pytorch-geometric.readthedocs.io/en/latest/notes/create_gnn.html
        #    1.2 We use the same representations for central (x_central) and
        #        neighbor (x_neighbor) nodes, which means you'll pass x=(x, x)
        #        to propagate.
        # 2. Update our node embedding with skip connection.
        # 3. If normalize is set, do L-2 normalization (defined in
        #    torch.nn.functional)
        # Our implementation is ~5 lines, but don't worry if you deviate from this.
        pass_msg = self.propagate(edge_index, x=(x,x), size=size)
        out = self.lin_l(x) + self.lin_r(pass_msg)
        if self.normalize:
            out = F.normalize(out, p=2)



        ############################################################################

        return out

    def message(self, x_j):

        out = None

        ############################################################################
```

```
        # TODO: Your code here!
        # Implement your message function here.
        # Our implementation is ~1 lines, but don't worry if you deviate from␣
↪this.
        out = x_j


       ␣
↪############################################################################

        return out

    def aggregate(self, inputs, index, dim_size = None):

        out = None

        # The axis along which to index number of nodes.
        node_dim = self.node_dim

       ␣
↪############################################################################
        # TODO: Your code here!
        # Implement your aggregate function here.
        # See here as how to use torch_scatter.scatter:
        # https://pytorch-scatter.readthedocs.io/en/latest/functions/scatter.
↪html#torch_scatter.scatter
        # Our implementation is ~1 lines, but don't worry if you deviate from␣
↪this.
        out = torch_scatter.scatter(inputs, index, node_dim, dim_size=dim_size,␣
↪reduce='mean')


       ␣
↪############################################################################

        return out
```

3.4 GAT Implementation

Attention mechanisms have become the state-of-the-art in many sequence-based tasks such as machine translation and learning sentence representations. One of the major benefits of attention-based mechanisms is their ability to focus on the most relevant parts of the input to make decisions. In this problem, we will see how attention mechanisms can be used to perform node classification of graph-structured data through the usage of Graph Attention Networks (GATs).

The building block of the Graph Attention Network is the graph attention layer, which is a variant of the aggregation function . Let N be the number of nodes and F be the dimension of the feature vector for each node. The input to each graph attentional layer is a set of node features: $\mathbf{h} = \{\overrightarrow{h_1}, \overrightarrow{h_2}, \dots, \overrightarrow{h_N}\}$, $\overrightarrow{h_i} \in R^F$. The output of each graph attentional layer is a new set of node features, which may have a new dimension F': $\mathbf{h}' = \{\overrightarrow{h'_1}, \overrightarrow{h'_2}, \dots, \overrightarrow{h'_N}\}$, with $\overrightarrow{h'_i} \in \mathbb{R}^{F'}$.

We will now describe this transformation of the input features into higher-level features performed by each graph attention layer. First, a shared linear transformation parametrized by the weight matrix $\mathbf{W} \in \mathbb{R}^{F' \times F}$ is applied to every node. Next, we perform self-attention on the nodes. We use a shared attentional mechanism:

$$a : \mathbb{R}^{F'} \times \mathbb{R}^{F'} \to \mathbb{R}. \tag{3}$$

This mechanism computes the attention coefficients that capture the importance of node j's features to node i:

$$e_{ij} = a(\mathbf{W_l}\overrightarrow{h_i}, \mathbf{W_r}\overrightarrow{h_j}) \tag{4}$$

The most general formulation of self-attention allows every node to attend to all other nodes which drops all structural information. To utilize graph structure in the attention mechanisms, we can use masked attention. In masked attention, we only compute e_{ij} for nodes $j \in \mathcal{N}_i$ where $\mathcal{N}_i$ is some neighborhood of node i in the graph.

To easily compare coefficients across different nodes, we normalize the coefficients across j using a softmax function:

$$\alpha_{ij} = \mathrm{softmax}_j(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{k \in \mathcal{N}_i} \exp(e_{ik})} \tag{5}$$

For this problem, our attention mechanism a will be a single-layer feedforward neural network parametrized by a weight vector $\overrightarrow{a} \in \mathbb{R}^{F'}$, followed by a LeakyReLU nonlinearity (with negative input slope 0.2). Let $\cdot^T$ represent transposition and $||$ represent concatenation. The coefficients computed by our attention mechanism may be expressed as:

$$\alpha_{ij} = \frac{\exp\left(\mathrm{LeakyReLU}\left(\overrightarrow{a_l}^T \mathbf{W_l}\overrightarrow{h_i} + \overrightarrow{a_r}^T \mathbf{W_r}\overrightarrow{h_j}\right)\right)}{\sum_{k \in \mathcal{N}_i} \exp\left(\mathrm{LeakyReLU}\left(\overrightarrow{a_l}^T \mathbf{W_l}\overrightarrow{h_i} + \overrightarrow{a_r}^T \mathbf{W_r}\overrightarrow{h_k}\right)\right)} \tag{6}$$

For the following questions, we denote $\alpha_l = [..., \overrightarrow{a_l}^T \mathbf{W_l}\overrightarrow{h_i}, ...]$ and $\alpha_r = [..., \overrightarrow{a_r}^T \mathbf{W_r}\overrightarrow{h_j}, ...]$.

At every layer of GAT, after the attention coefficients are computed for that layer, the aggregation function can be computed by a weighted sum of neighborhood messages, where weights are specified by α_{ij}.

Now, we use the normalized attention coefficients to compute a linear combination of the features corresponding to them. These aggregated features will serve as the final output features for every node.

$$h'_i = \sum_{j \in \mathcal{N}_i} \alpha_{ij} \mathbf{W_r} \overrightarrow{h_j}. \tag{7}$$

To stabilize the learning process of self-attention, we use multi-head attention. To do this we use K independent attention mechanisms, or "heads' ' compute output features as in the above equations. Then, we concatenate these output feature representations:

$$\overrightarrow{h_i}' = ||_{k=1}^{K}\Big(\sum_{j\in\mathcal{N}_i}\alpha_{ij}^{(k)}\mathbf{W_r}^{(k)}\overrightarrow{h_j}\Big) \tag{8}$$

where $||$ is concentation, $\alpha_{ij}^{(k)}$ are the normalized attention coefficients computed by the k-th attention mechanism (a^k), and $\mathbf{W}^{(k)}$ is the corresponding input linear transformation's weight matrix. Note that for this setting, $\mathbf{h'} \in \mathbb{R}^{KF'}$.

[8]:
```
class GAT(MessagePassing):

    def __init__(self, in_channels, out_channels, heads = 2,
                 negative_slope = 0.2, dropout = 0., **kwargs):
        super(GAT, self).__init__(node_dim=0, **kwargs)

        self.in_channels = in_channels
        self.out_channels = out_channels
        self.heads = heads
        self.negative_slope = negative_slope
        self.dropout = dropout

        self.lin_l = None
        self.lin_r = None
        self.att_l = None
        self.att_r = None

        ␣
↪############################################################################
        # TODO: Your code here!
        # Define the layers needed for the message functions below.
        # self.lin_l is the linear transformation that you apply to embeddings
        # BEFORE message passing.
        # Pay attention to dimensions of the linear layers, since we're using
        # multi-head attention.
        # Our implementation is ~1 lines, but don't worry if you deviate from␣
↪this.
        self.lin_l = nn.Linear(self.in_channels, self.out_channels * self.heads)

        ␣
↪############################################################################

        self.lin_r = self.lin_l

        ␣
↪############################################################################
        # TODO: Your code here!
        # Define the attention parameters \overrightarrow{a_l/r}^T in the above␣
↪intro.
```

```
        # You have to deal with multi-head scenarios.
        # Use nn.Parameter instead of nn.Linear
        # Our implementation is ~2 lines, but don't worry if you deviate from␣
↪this.
        self.att_l = nn.Parameter(torch.zeros(self.heads, self.out_channels))
        self.att_r = nn.Parameter(torch.zeros(self.heads, self.out_channels))

        ␣
↪############################################################################

        self.reset_parameters()

    def reset_parameters(self):
        nn.init.xavier_uniform_(self.lin_l.weight)
        nn.init.xavier_uniform_(self.lin_r.weight)
        nn.init.xavier_uniform_(self.att_l)
        nn.init.xavier_uniform_(self.att_r)

    def forward(self, x, edge_index, size = None):

        H, C = self.heads, self.out_channels

        ␣
↪############################################################################
        # TODO: Your code here!
        # Implement message passing, as well as any pre- and post-processing␣
↪(our update rule).
        # 1. First apply linear transformation to node embeddings, and split␣
↪that
        #    into multiple heads. We use the same representations for source and
        #    target nodes, but apply different linear weights (W_l and W_r)
        # 2. Calculate alpha vectors for central nodes (alpha_l) and neighbor␣
↪nodes (alpha_r).
        # 3. Call propagate function to conduct the message passing.
        #    3.1 Remember to pass alpha = (alpha_l, alpha_r) as a parameter.
        #    3.2 See there for more information: https://pytorch-geometric.
↪readthedocs.io/en/latest/notes/create_gnn.html
        # 4. Transform the output back to the shape of N * d.
        # Our implementation is ~5 lines, but don't worry if you deviate from␣
↪this.

        ## I lost my previous code!! :(
        ## This code (and the rest of the code required) is taken from https://
↪github.com/luciusssss/CS224W-Colab/blob/main/CS224W-Colab%203.ipynb
        x_l = self.lin_l(x).reshape(-1, H, C)
        x_r = self.lin_r(x).reshape(-1, H, C)
```

```
        alpha_l = self.att_l * x_l
        alpha_r = self.att_r * x_r
        out = self.propagate(edge_index, x=(x_l, x_r), alpha=(alpha_l,␣
↪alpha_r), size=size)
        out = out.reshape(-1, H*C)



       ␣
↪############################################################################

        return out


    def message(self, x_j, alpha_j, alpha_i, index, ptr, size_i):

       ␣
↪############################################################################
        # TODO: Your code here!
        # Implement your message function. Putting the attention in message
        # instead of in update is a little tricky.
        # 1. Calculate the final attention weights using alpha_i and alpha_j,
        #    and apply leaky Relu.
        # 2. Calculate softmax over the neighbor nodes for all the nodes. Use
        #    torch_geometric.utils.softmax instead of the one in Pytorch.
        # 3. Apply dropout to attention weights (alpha).
        # 4. Multiply embeddings and attention weights. As a sanity check, the␣
↪output
        #    should be of shape E * H * d.
        # 5. ptr (LongTensor, optional): If given, computes the softmax based on
        #    sorted inputs in CSR representation. You can simply pass it to␣
↪softmax.
        # Our implementation is ~5 lines, but don't worry if you deviate from␣
↪this.
        alpha = F.leaky_relu(alpha_i + alpha_j, negative_slope=self.
↪negative_slope)
        if ptr:
            att_weight = F.softmax(alpha_i + alpha_j, ptr)
        else:
            att_weight = torch_geometric.utils.softmax(alpha_i + alpha_j, index)
        att_weight = F.dropout(att_weight, p=self.dropout)
        out = att_weight * x_j



       ␣
↪############################################################################
```

```
        return out


    def aggregate(self, inputs, index, dim_size = None):

        ␣
↪##############################################################################
        # TODO: Your code here!
        # Implement your aggregate function here.
        # See here as how to use torch_scatter.scatter: https://pytorch-scatter.
↪readthedocs.io/en/latest/_modules/torch_scatter/scatter.html
        # Pay attention to "reduce" parameter is different from that in␣
↪GraphSage.
        # Our implementation is ~1 lines, but don't worry if you deviate from␣
↪this.

        out = torch_scatter.scatter(inputs, index, self.node_dim,␣
↪dim_size=dim_size, reduce='sum')

        ␣
↪##############################################################################

        return out
```

3.5 Building Optimizers

This function has been implemented for you. **For grading purposes please use the default Adam optimizer**, but feel free to play with other types of optimizers on your own.

[9]:
```
import torch.optim as optim

def build_optimizer(args, params):
    weight_decay = args.weight_decay
    filter_fn = filter(lambda p : p.requires_grad, params)
    if args.opt == 'adam':
        optimizer = optim.Adam(filter_fn, lr=args.lr, weight_decay=weight_decay)
    elif args.opt == 'sgd':
        optimizer = optim.SGD(filter_fn, lr=args.lr, momentum=0.95,␣
↪weight_decay=weight_decay)
    elif args.opt == 'rmsprop':
        optimizer = optim.RMSprop(filter_fn, lr=args.lr,␣
↪weight_decay=weight_decay)
    elif args.opt == 'adagrad':
        optimizer = optim.Adagrad(filter_fn, lr=args.lr,␣
↪weight_decay=weight_decay)
```

```
    if args.opt_scheduler == 'none':
        return None, optimizer
    elif args.opt_scheduler == 'step':
        scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=args.
↪opt_decay_step, gamma=args.opt_decay_rate)
    elif args.opt_scheduler == 'cos':
        scheduler = optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=args.
↪opt_restart)
    return scheduler, optimizer
```

3.6 Training and Testing

Here we provide you with the functions to train and test. **Please do not modify this part for grading purposes.**

[10]:
```
import time

import networkx as nx
import numpy as np
import torch
import torch.optim as optim

from torch_geometric.datasets import TUDataset
from torch_geometric.datasets import Planetoid
from torch_geometric.data import DataLoader

import torch_geometric.nn as pyg_nn

import matplotlib.pyplot as plt


def train(dataset, args):

    print("Node task. test set size:", np.sum(dataset[0]['train_mask'].numpy()))
    test_loader = loader = DataLoader(dataset, batch_size=args.batch_size,␣
↪shuffle=True)

    # build model
    model = GNNStack(dataset.num_node_features, args.hidden_dim, dataset.
↪num_classes,
                            args)
    scheduler, opt = build_optimizer(args, model.parameters())

    # train
    losses = []
    test_accs = []
    for epoch in range(args.epochs):
```

```
        total_loss = 0
        model.train()
        for batch in loader:
            opt.zero_grad()
            pred = model(batch)
            label = batch.y
            pred = pred[batch.train_mask]
            label = label[batch.train_mask]
            loss = model.loss(pred, label)
            loss.backward()
            opt.step()
            total_loss += loss.item() * batch.num_graphs
        total_loss /= len(loader.dataset)
        losses.append(total_loss)

        if epoch % 10 == 0:
          test_acc = test(test_loader, model)
          test_accs.append(test_acc)
        else:
          test_accs.append(test_accs[-1])
    return test_accs, losses

def test(loader, model, is_validation=True):
    model.eval()

    correct = 0
    for data in loader:
        with torch.no_grad():
            # max(dim=1) returns values, indices tuple; only need indices
            pred = model(data).max(dim=1)[1]
            label = data.y

        mask = data.val_mask if is_validation else data.test_mask
        # node classification: only evaluate on nodes in test set
        pred = pred[mask]
        label = data.y[mask]

        correct += pred.eq(label).sum().item()

    total = 0
    for data in loader.dataset:
        total += torch.sum(data.val_mask if is_validation else data.test_mask).
 ↪item()
    return correct / total

class objectview(object):
    def __init__(self, d):
```

```
        self.__dict__ = d
```

3.7 Let's Start the Training!

We will be working on the CORA dataset on node-level classification.

This part is implemented for you. **For grading purposes, please do not modify the default parameters.** However, feel free to play with different configurations just for fun!

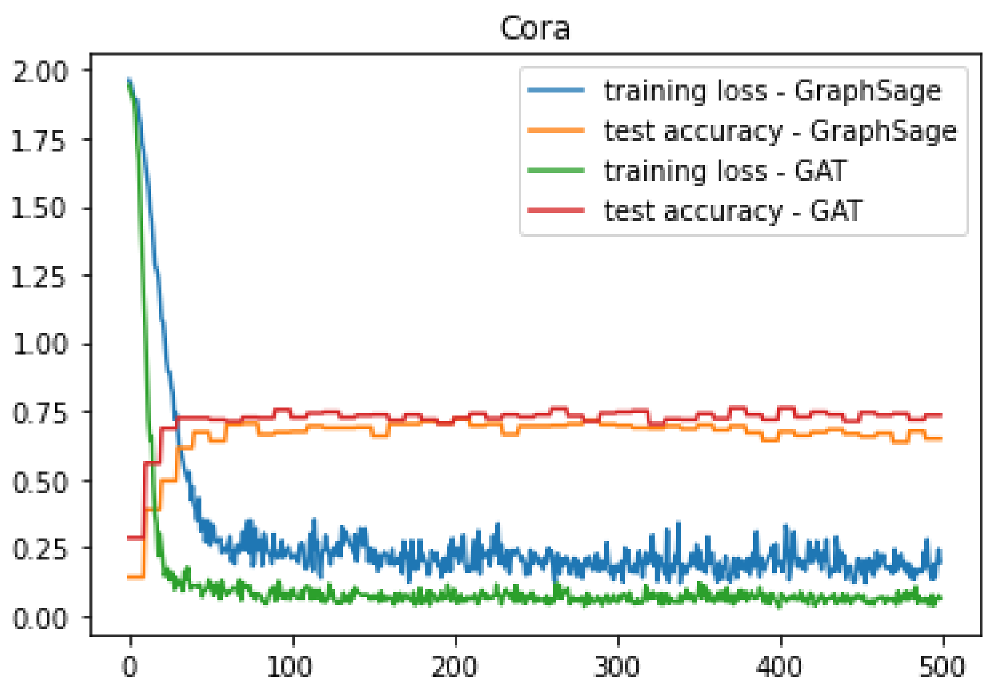
Submit your best accuracy and loss on Gradescope.

[11]:
```
def main():
    for args in [
        {'model_type': 'GraphSage', 'dataset': 'cora', 'num_layers': 2, 'heads':
↪ 1, 'batch_size': 32, 'hidden_dim': 32, 'dropout': 0.5, 'epochs': 500, 'opt':
↪ 'adam', 'opt_scheduler': 'none', 'opt_restart': 0, 'weight_decay': 5e-3,␣
↪'lr': 0.01},
    ]:
        args = objectview(args)
        for model in ['GraphSage', 'GAT']:
            args.model_type = model

            # Match the dimension.
            if model == 'GAT':
              args.heads = 2
            else:
              args.heads = 1

            if args.dataset == 'cora':
                dataset = Planetoid(root='/tmp/cora', name='Cora')
            else:
                raise NotImplementedError("Unknown dataset")
            test_accs, losses = train(dataset, args)

            print("Maximum accuracy: {0}".format(max(test_accs)))
            print("Minimum loss: {0}".format(min(losses)))

            plt.title(dataset.name)
            plt.plot(losses, label="training loss" + " - " + args.model_type)
            plt.plot(test_accs, label="test accuracy" + " - " + args.model_type)
        plt.legend()
        plt.show()

if __name__ == '__main__':
    main()
```

```
Downloading https://github.com/kimiyoung/planetoid/raw/master/data/ind.cora.x
Downloading https://github.com/kimiyoung/planetoid/raw/master/data/ind.cora.tx
```

```
Downloading https://github.com/kimiyoung/planetoid/raw/master/data/ind.cora.allx
Downloading https://github.com/kimiyoung/planetoid/raw/master/data/ind.cora.y
Downloading https://github.com/kimiyoung/planetoid/raw/master/data/ind.cora.ty
Downloading https://github.com/kimiyoung/planetoid/raw/master/data/ind.cora.ally
Downloading
https://github.com/kimiyoung/planetoid/raw/master/data/ind.cora.graph
Downloading
https://github.com/kimiyoung/planetoid/raw/master/data/ind.cora.test.index
Processing…
Done!
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/torch_geometric/deprecation.py:13: UserWarning: 'data.DataLoader' is
deprecated, use 'loader.DataLoader' instead
  warnings.warn(out)
```

```
Node task. test set size: 140
Maximum accuracy: 0.722
Minimum loss: 0.11809366941452026
Node task. test set size: 140
Maximum accuracy: 0.76
Minimum loss: 0.02870200015604496
```



3.8 Question 1.1: What is the maximum accuracy you could get on test set for GraphSage? (10 points)

Submit your answers on Gradescope.

Maximum accuracy for GraphSage: 72.2%

3.9 Question 1.2: What is the maximum accuracy you could get on test set for GAT? (10 points)

Submit your answers on Gradescope.

Maximum accuracy for GAT: 76%

4 2 DeepSNAP Basics

In previous Colabs we used both of graph class (NetworkX) and tensor (PyG) representations of graphs separately. The graph class `nx.Graph` provides rich analysis and manipulation functionalities, such as the clustering coefficient and PageRank. To feed the graph into the model, we need to transform the graph into tensor representations including edge tensor `edge_index` and node attributes tensors `x` and `y`. But only using tensors (as the graphs formatted in PyG `datasets` and `data`) will make many graph manipulations and analysis less efficient and harder. So, in this Colab we will use DeepSNAP which combines both representations and offers a full pipeline for GNN training / validation / testing.

In general, DeepSNAP is a Python library to assist efficient deep learning on graphs. DeepSNAP features in its support for flexible graph manipulation, standard pipeline, heterogeneous graphs and simple API.

1. DeepSNAP is easy to be used for the sophisticated graph manipulations, such as feature computation, pretraining, subgraph extraction etc. during/before the training.
2. In most frameworks, standard pipelines for node, edge, link, graph-level tasks under inductive or transductive settings are left to the user to code. In practice, there are additional design choices involved (such as how to split dataset for link prediction). DeepSNAP provides such a standard pipeline that greatly saves repetitive coding efforts, and enables fair comparision for models.
3. Many real-world graphs are heterogeneous graphs. But packages support for heterogeneous graphs, including data storage and flexible message passing, is lacking. DeepSNAP provides an efficient and flexible heterogeneous graph that supports both the node and edge heterogeneity.

DeepSNAP is a newly released project and it is still under development. If you find any bugs or have any improvement ideas, feel free to raise issues or create pull requests on the GitHub directly :)

In this Colab, we will focus on DeepSNAP graph manipulations and splitting settings.

4.1 Setup

[12]:
```
import torch
import networkx as nx
import matplotlib.pyplot as plt

from deepsnap.graph import Graph
from deepsnap.batch import Batch
```

```
from deepsnap.dataset import GraphDataset
from torch_geometric.datasets import Planetoid, TUDataset

from torch.utils.data import DataLoader

def visualize(G, color_map=None, seed=123):
  if color_map is None:
    color_map = '#c92506'
  plt.figure(figsize=(8, 8))
  nodes = nx.draw_networkx_nodes(G, pos=nx.spring_layout(G, seed=seed), \
                                 label=None, node_color=color_map,␣
↪node_shape='o', node_size=150)
  edges = nx.draw_networkx_edges(G, pos=nx.spring_layout(G, seed=seed), alpha=0.
↪5)
  if color_map is not None:
    plt.scatter([],[], c='#c92506', label='Nodes with label 0',␣
↪edgecolors="black", s=140)
    plt.scatter([],[], c='#fcec00', label='Nodes with label 1',␣
↪edgecolors="black", s=140)
    plt.legend(prop={'size': 13}, handletextpad=0)
  nodes.set_edgecolor('black')
  plt.show()
```

4.2 DeepSNAP Graph

The `deepsnap.graph.Graph` class is the core class of DeepSNAP. It not only represents a graph in tensor format but also references to a graph object from graph manipulation package.

Currently DeepSNAP supports NetworkX and Snap.py as the back end graph manipulation package.

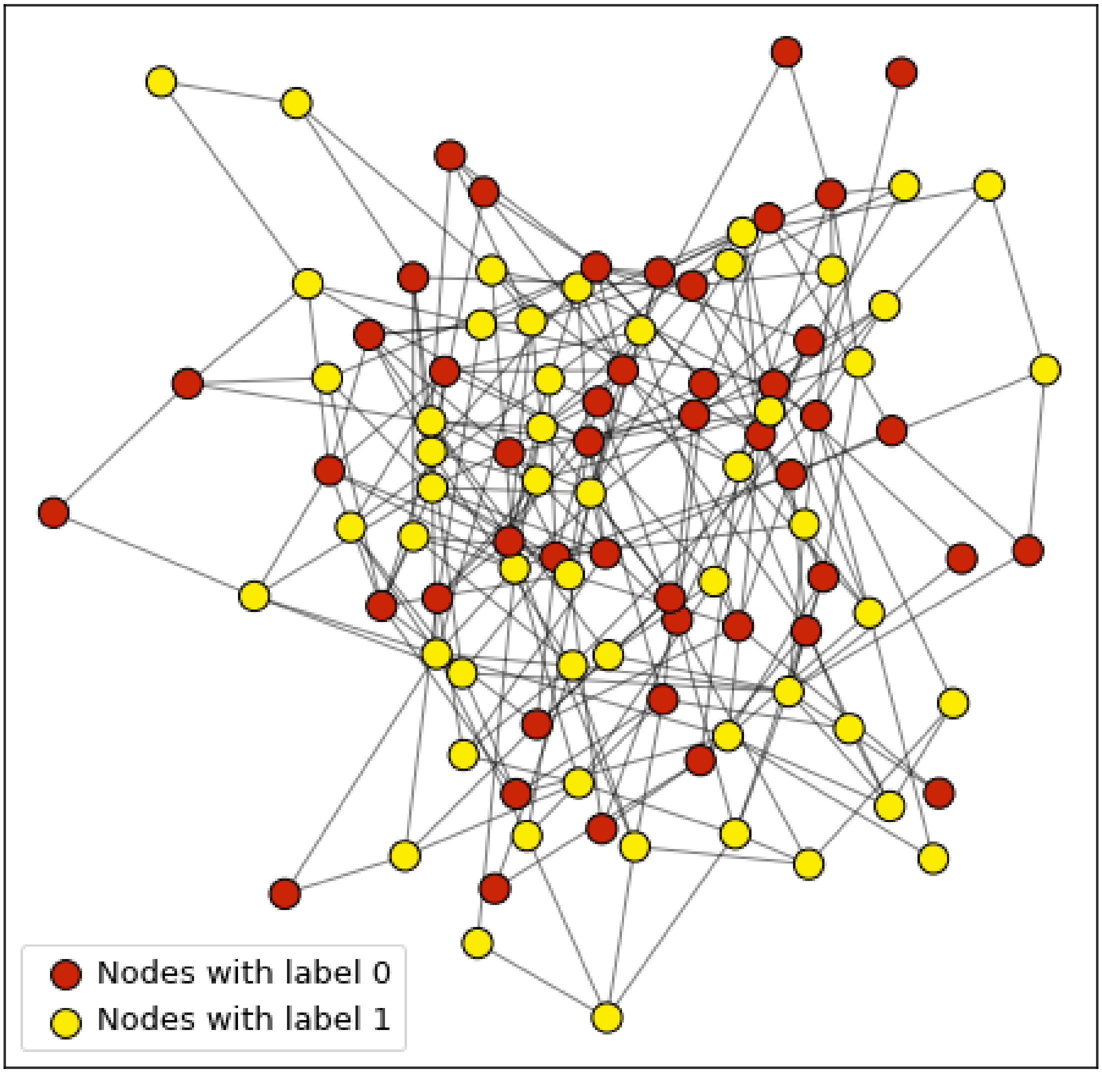
In this Colab, we will use the NetworkX as the back end graph manipulation package.

Lets first try to convert a simple random NetworkX graph to a DeepSNAP graph.

[13]:
```
num_nodes = 100
p = 0.05
seed = 100

# Generate a networkx random graph
G = nx.gnp_random_graph(num_nodes, p, seed=seed)

# Generate some random node features and labels
node_feature = {node : torch.rand([5, ]) for node in G.nodes()}
node_label = {node : torch.randint(0, 2, ()) for node in G.nodes()}

# Set the random features and labels to G
nx.set_node_attributes(G, node_feature, name='node_feature')
```

```python
nx.set_node_attributes(G, node_label, name='node_label')

# Print one node example
for node in G.nodes(data=True):
  print(node)
  break

color_map = ['#c92506' if node[1]['node_label'].item() == 0 else '#fcec00' for␣
↪node in G.nodes(data=True)]

# Visualize the graph
visualize(G, color_map=color_map)

# Transform the networkx graph into the deepsnap graph
graph = Graph(G)

# Print out the general deepsnap graph information
print(graph)

# DeepSNAP will convert node attributes to tensors
# Notice the type of tensors
print("Node feature (node_feature) has shape {} and type {}".format(graph.
↪node_feature.shape, graph.node_feature.dtype))
print("Node label (node_label) has shape {} and type {}".format(graph.
↪node_label.shape, graph.node_label.dtype))

# DeepSNAP will also generate the edge_index tensor
print("Edge index (edge_index) has shape {} and type {}".format(graph.
↪edge_index.shape, graph.edge_index.dtype))

# Different from only storing tensors, deepsnap graph also references to the␣
↪networkx graph
# We will discuss why the reference will be helpful later
print("The DeepSNAP graph has {} as the internal manupulation graph".
↪format(type(graph.G)))
```

```
(0, {'node_feature': tensor([0.2746, 0.2966, 0.6027, 0.0072, 0.5447]),
'node_label': tensor(1)})
```




```
Graph(G=[], edge_index=[2, 524], edge_label_index=[2, 524], node_feature=[100,
5], node_label=[100], node_label_index=[100])
Node feature (node_feature) has shape torch.Size([100, 5]) and type
torch.float32
Node label (node_label) has shape torch.Size([100]) and type torch.int64
Edge index (edge_index) has shape torch.Size([2, 524]) and type torch.int64
The DeepSNAP graph has <class 'networkx.classes.graph.Graph'> as the internal
manupulation graph
```

In DeepSNAP we have three levels of attributes. In this example, we have the **node level** attributes including `node_feature` and `node_label`. The other two levels of attributes are graph and edge attributes. The usage is similar to the node level one except that the feature becomes `edge_feature` or `graph_feature` and label becomes `edge_label` or `graph_label` etc.

Similar to the NetworkX graph, we can easily get some basic information of the graph through class properties directly.

```
[14]: # Number of nodes
print("The random graph has {} nodes".format(graph.num_nodes))

# Number of edges
print("The random graph has {} edges".format(graph.num_edges))
```

```
The random graph has 100 nodes
The random graph has 262 edges
```

DeepSNAP also provides functions that can automatically transform the PyG datasets into a list of DeepSNAP graphs.

Here we transform the CORA dataset into a list of DeepSNAP graphs.

```
[15]: root = './tmp/cora'
name = 'Cora'

# The Cora dataset
pyg_dataset= Planetoid(root, name)

# PyG dataset to a list of deepsnap graphs
graphs = GraphDataset.pyg_to_graphs(pyg_dataset)

# Get the first deepsnap graph (CORA only has one graph)
graph = graphs[0]
print(graph)
```

```
Downloading https://github.com/kimiyoung/planetoid/raw/master/data/ind.cora.x
Downloading https://github.com/kimiyoung/planetoid/raw/master/data/ind.cora.tx
Downloading https://github.com/kimiyoung/planetoid/raw/master/data/ind.cora.allx
Downloading https://github.com/kimiyoung/planetoid/raw/master/data/ind.cora.y
Downloading https://github.com/kimiyoung/planetoid/raw/master/data/ind.cora.ty
Downloading https://github.com/kimiyoung/planetoid/raw/master/data/ind.cora.ally
Downloading
https://github.com/kimiyoung/planetoid/raw/master/data/ind.cora.graph
Downloading
https://github.com/kimiyoung/planetoid/raw/master/data/ind.cora.test.index
```

```
Graph(G=[], edge_index=[2, 10556], edge_label_index=[2, 10556],
node_feature=[2708, 1433], node_label=[2708], node_label_index=[2708])
```

```
Processing…
Done!
```

4.3 Question 2.1: What is the number of classes and number of features in the CORA graph? (5 points)

Submit your answers on Gradescope.

[16]:
```
def get_num_node_classes(graph):
  # TODO: Implement this function that takes a deepsnap graph object
  # and return the number of node classes of that graph.

    num_node_classes = 0

  ############# Your code here #############
  ## (~1 line of code)
  ## Note
  ## 1. Colab autocomplete functionality might be useful
  ## 2. DeepSNAP documentation might be useful https://snap.stanford.edu/
 ↪deepsnap/modules/graph.html

    num_node_classes = graph.num_node_labels
  ##########################################

    return num_node_classes

def get_num_node_features(graph):
  # TODO: Implement this function that takes a deepsnap graph object
  # and return the number of node features of that graph.

    num_node_features = 0

  ############# Your code here #############
  ## (~1 line of code)
  ## Note
  ## 1. Colab autocomplete functionality might be useful
  ## 2. DeepSNAP documentation might be useful https://snap.stanford.edu/
 ↪deepsnap/modules/graph.html
    num_node_features = graph.num_node_features

  ##########################################

    return num_node_features

num_node_classes = get_num_node_classes(graph)
num_node_features = get_num_node_features(graph)
print("{} has {} classes".format(name, num_node_classes))
print("{} has {} features".format(name, num_node_features))
```

```
Cora has 7 classes
Cora has 1433 features
```

4.4 DeepSNAP Dataset

Now, lets talk about DeepSNAP dataset. A `deepsnap.dataset.GraphDataset` contains a list of `deepsnap.graph.Graph` objects. In addition to list of graphs, you can also specify what task the dataset will be used on, such as node level task (`task=node`), edge level task (`task=link_pred`) and graph level task (`task=graph`).

It also contains many other useful parameters during initialization and other functinoalities. If you are interested, you can take a look at the documentation.

Lets now use COX2 dataset which contains a list of graphs and specify the task to `graph` when we initialize the DeepSNAP dataset.

[17]:
```python
root = './tmp/cox2'
name = 'COX2'

# Load the dataset through PyG
pyg_dataset = TUDataset(root, name)

# Convert to a list of deepsnap graphs
graphs = GraphDataset.pyg_to_graphs(pyg_dataset)

# Convert list of deepsnap graphs to deepsnap dataset with specified task=graph
dataset = GraphDataset(graphs, task='graph')
print(dataset)
```

```
Downloading https://www.chrsmrrs.com/graphkerneldatasets/COX2.zip
Extracting tmp/cox2/COX2/COX2.zip
Processing…
Done!
```

```
GraphDataset(467)
```

4.5 Question 2.2: What is the label of the graph (index 100 in the COX2 dataset)? (5 points)

Submit your answers on Gradescope.

[18]:
```python
def get_graph_class(dataset, idx):
  # TODO: Implement this function that takes a deepsnap dataset object,
  # the index of the graph in the dataset, and returns the class/label
  # of the graph (in integer).

    label = -1

  ############# Your code here ############
  ## (~1 line of code)
  ## Note
  ## 1. The label refers to the graph-level attribute
    label = dataset[idx].graph_label

  #########################################

    return label
```

```
graph_0 = dataset[0]
print(graph_0)
idx = 100
label = get_graph_class(dataset, idx)
print('Graph with index {} has label {}'.format(idx, label))
```

```
Graph(G=[], edge_index=[2, 82], edge_label_index=[2, 82], graph_label=[1],
node_feature=[39, 35], node_label_index=[39], task=[])
Graph with index 100 has label tensor([0])
```

4.6 Question 2.3: What is the number of edges for the graph (index 200 in the COX2 dataset)? (5 points)

Submit your answers on Gradescope.

[19]:
```
def get_graph_num_edges(dataset, idx):
  # TODO: Implement this function that takes a deepsnap dataset object,
  # the index of the graph in dataset, and returns the number of
  # edges in the graph (in integer).

    num_edges = 0

  ############# Your code here ############
  ## (~1 lines of code)
  ## Note
  ## 1. You can use the class property directly

    num_edges = dataset[idx].num_edges
  #########################################

    return num_edges

idx = 200
num_edges = get_graph_num_edges(dataset, idx)
print('Graph with index {} has {} edges'.format(idx, num_edges))
```

```
Graph with index 200 has 49 edges
```

5 3 DeepSNAP Advanced

We have learned the basic use of DeepSNAP graph and dataset :)

Lets move on to some more advanced functionalities.

In this section we will use DeepSNAP for faeture computation and transductive/inductive splittings.

5.1 Setup

[20]:
```
import torch
import networkx as nx
import matplotlib.pyplot as plt

from deepsnap.graph import Graph
from deepsnap.batch import Batch
from deepsnap.dataset import GraphDataset
from torch_geometric.datasets import Planetoid, TUDataset

from torch.utils.data import DataLoader
```

5.2 Data Split in Graphs

Data splitting in graphs can be much harder than that in CV or NLP.

In general, the data splitting in graphs can be divided into two settings, **inductive** and **transductive**.

5.3 Inductive Split

As what we have learned in the lecture, inductive setting will split multiple graphs into each training/valiation and test sets.

Here is an example of DeepSNAP inductive splitting for a list of graphs in the graph level task (graph classification etc.)

[21]:
```
root = './tmp/cox2'
name = 'COX2'

pyg_dataset = TUDataset(root, name)

graphs = GraphDataset.pyg_to_graphs(pyg_dataset)

# Here we specify the task as graph-level task such as graph classification
task = 'graph'
dataset = GraphDataset(graphs, task=task)

# Specify transductive=False (inductive)
dataset_train, dataset_val, dataset_test = dataset.split(transductive=False,␣
↪split_ratio=[0.8, 0.1, 0.1])

print("COX2 train dataset: {}".format(dataset_train))
print("COX2 validation dataset: {}".format(dataset_val))
print("COX2 test dataset: {}".format(dataset_test))
```

```
COX2 train dataset: GraphDataset(373)
COX2 validation dataset: GraphDataset(46)
```

```
COX2 test dataset: GraphDataset(48)
```

5.4 Transductive Split

In transductive setting, the training /validation / test sets are on the same graph.

Here we transductively split the CORA graph in the node level task.

(Notice that in DeepSNAP default setting the split is random, but you can also make a fixed split by specifying `fixed_split=True` when loading the dataset from PyG or changing the `node_label_index` directly).

[22]:
```python
root = './tmp/cora'
name = 'Cora'

pyg_dataset = Planetoid(root, name)

graphs = GraphDataset.pyg_to_graphs(pyg_dataset)

# Here we specify the task as node-level task such as node classification
task = 'node'

dataset = GraphDataset(graphs, task=task)

# Specify we want the transductive splitting
dataset_train, dataset_val, dataset_test = dataset.split(transductive=True,␣
 ↪split_ratio=[0.8, 0.1, 0.1])

print("Cora train dataset: {}".format(dataset_train))
print("Cora validation dataset: {}".format(dataset_val))
print("Cora test dataset: {}".format(dataset_test))

print("Original Cora has {} nodes".format(dataset.num_nodes[0]))

# The nodes in each set can be find in node_label_index
print("After the split, Cora has {} training nodes".format(dataset_train[0].
 ↪node_label_index.shape[0]))
print("After the split, Cora has {} validation nodes".format(dataset_val[0].
 ↪node_label_index.shape[0]))
print("After the split, Cora has {} test nodes".format(dataset_test[0].
 ↪node_label_index.shape[0]))
```

```
Cora train dataset: GraphDataset(1)
Cora validation dataset: GraphDataset(1)
Cora test dataset: GraphDataset(1)
Original Cora has 2708 nodes
After the split, Cora has 2166 training nodes
After the split, Cora has 270 validation nodes
After the split, Cora has 272 test nodes
```

5.5 Edge Level Split

Compared to the node and graph level splitting, edge level splitting is a little bit tricky ;)

Usually in edge level splitting, we need to sample negative edges, split positive edges into different datasets, split training edges into message passing edges and supervision edges, and resample the negative edges during the training etc.

5.5.1 All Mode

Now lets start with a simpler edge level splitting mode, the `edge_train_mode="all"` mode in DeepSNAP.

[23]:
```
root = './tmp/cora'
name = 'Cora'

pyg_dataset = Planetoid(root, name)

graphs = GraphDataset.pyg_to_graphs(pyg_dataset)

# Specify task as link_pred for edge-level task
task = 'link_pred'

# Specify the train mode, "all" mode is default for deepsnap dataset
edge_train_mode = "all"

dataset = GraphDataset(graphs, task=task, edge_train_mode=edge_train_mode)

# Transductive link prediction split
dataset_train, dataset_val, dataset_test = dataset.split(transductive=True,␣
↪split_ratio=[0.8, 0.1, 0.1])

print("Cora train dataset: {}".format(dataset_train))
print("Cora validation dataset: {}".format(dataset_val))
print("Cora test dataset: {}".format(dataset_test))
```

```
Cora train dataset: GraphDataset(1)
Cora validation dataset: GraphDataset(1)
Cora test dataset: GraphDataset(1)
```

```
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
```

In DeepSNAP, the indices of supervision edges are stored in `edge_label_index` tensor and the corresponding edge labels are stored in `edge_label` tensor.

[24]:
```
print("Original Cora graph has {} edges".format(dataset[0].num_edges))
print("Because Cora graph is undirected, the original edge_index has shape {}".
 ↪format(dataset[0].edge_index.shape))

print("The training set has message passing edge index shape {}".
 ↪format(dataset_train[0].edge_index.shape))
print("The training set has supervision edge index shape {}".
 ↪format(dataset_train[0].edge_label_index.shape))

print("The validation set has message passing edge index shape {}".
 ↪format(dataset_val[0].edge_index.shape))
print("The validation set has supervision edge index shape {}".
 ↪format(dataset_val[0].edge_label_index.shape))

print("The test set has message passing edge index shape {}".
 ↪format(dataset_test[0].edge_index.shape))
print("The test set has supervision edge index shape {}".format(dataset_test[0].
 ↪edge_label_index.shape))
```

```
Original Cora graph has 5278 edges
Because Cora graph is undirected, the original edge_index has shape
torch.Size([2, 10556])
The training set has message passing edge index shape torch.Size([2, 8444])
The training set has supervision edge index shape torch.Size([2, 16888])
The validation set has message passing edge index shape torch.Size([2, 8444])
The validation set has supervision edge index shape torch.Size([2, 2108])
The test set has message passing edge index shape torch.Size([2, 9498])
The test set has supervision edge index shape torch.Size([2, 2116])
```

We can see that both training and validation sets have the same message passing edges (`edge_index`) in the `all` mode. Also, in training set, the postive supervision edges (`edge_label_index`) are same with the message passing edges. However, in the test set the message passing edges are the combination of message passing edges from training and validation sets.

Notice that the `edge_label` and `edge_label_index` have included the negative edges (default number of negative edges is same with the number of positive edges).

Now, lets implement a function that checks whether two edge index tensors are disjoint and explore more edge splitting properties by using that function.

5.6 Question 3.1 - 3.5: Implement the function that checks whether two edge_index tensors are disjoint. Then answer the True/False questions below. (5 points)

Submit your answers on Gradescope.

[27]:
```
def edge_indices_disjoint(edge_index_1, edge_index_2):
  # TODO: Implement this function that takes two edge index tensors,
  # and returns whether these two edge index tensors are disjoint.
    disjoint = None

  ############# Your code here ############
  ## (~5 lines of code)
  ## Note
  ## 1. Here disjoint means that there is no single edge belongs to either edge␣
 ↪index tensors
  ## 2. You do not need to consider the undirected case. For example, if␣
 ↪edge_index_1 contains
  ## edge (a, b) and edge_index_2 contains edge (b, a). We will treat them as␣
 ↪disjoint in this
  ## function.
    disjoint = True
    edge_set_1 = set([t for t in edge_index_1.reshape(-1, 2)])
    for edge in edge_index_2.reshape(-1,2):
        if edge in edge_set_1:
            disjoint = False
            break

  #########################################

    return disjoint
```

[28]:
```
num_train_edges = dataset_train[0].edge_label_index.shape[1] // 2
train_pos_edge_index = dataset_train[0].edge_label_index[:, :num_train_edges]
train_neg_edge_index = dataset_train[0].edge_label_index[:, num_train_edges:]
print("3.1 Training (supervision) positve and negative edges are disjoint = {}"\
        .format(edge_indices_disjoint(train_pos_edge_index,␣
 ↪train_neg_edge_index)))

num_val_edges = dataset_val[0].edge_label_index.shape[1] // 2
val_pos_edge_index = dataset_val[0].edge_label_index[:, :num_val_edges]
val_neg_edge_index = dataset_val[0].edge_label_index[:, num_val_edges:]
print("3.2 Validation (supervision) positve and negative edges are disjoint =␣
 ↪{}"\
        .format(edge_indices_disjoint(val_pos_edge_index, val_neg_edge_index)))

num_test_edges = dataset_test[0].edge_label_index.shape[1] // 2
test_pos_edge_index = dataset_test[0].edge_label_index[:, :num_test_edges]
test_neg_edge_index = dataset_test[0].edge_label_index[:, num_test_edges:]
print("3.3 Test (supervision) positve and negative edges are disjoint = {}"\
        .format(edge_indices_disjoint(test_pos_edge_index,␣
 ↪test_neg_edge_index)))
```

```
print("3.4 Test (supervision) positve and validation (supervision) positve␣
↪edges are disjoint = {}"\
        .format(edge_indices_disjoint(test_pos_edge_index, val_pos_edge_index)))
print("3.5 Validation (supervision) positve and training (supervision) positve␣
↪edges are disjoint = {}"\
        .format(edge_indices_disjoint(val_pos_edge_index,␣
↪train_pos_edge_index)))
```

```
3.1 Training (supervision) positve and negative edges are disjoint = True
3.2 Validation (supervision) positve and negative edges are disjoint = True
3.3 Test (supervision) positve and negative edges are disjoint = True
3.4 Test (supervision) positve and validation (supervision) positve edges are
disjoint = True
3.5 Validation (supervision) positve and training (supervision) positve edges
are disjoint = True
```

5.6.1 Disjoint Mode

Now lets look at a relatively more complex transductive edge split setting, which is the `edge_train_mode="disjoint"` mode in DeepSNAP (also the transductive link prediction splitting talked in the lecture)

[29]:
```
edge_train_mode = "disjoint"

dataset = GraphDataset(graphs, task='link_pred',␣
↪edge_train_mode=edge_train_mode)
orig_edge_index = dataset[0].edge_index
dataset_train, dataset_val, dataset_test = dataset.split(
    transductive=True, split_ratio=[0.8, 0.1, 0.1])

train_message_edge_index = dataset_train[0].edge_index
train_sup_edge_index = dataset_train[0].edge_label_index
val_sup_edge_index = dataset_val[0].edge_label_index
test_sup_edge_index = dataset_test[0].edge_label_index

print("The edge index of original graph has shape: {}".format(orig_edge_index.
↪shape))
print("The edge index of training message edges has shape: {}".
↪format(train_message_edge_index.shape))
print("The edge index of training supervision edges has shape: {}".
↪format(train_sup_edge_index.shape))
print("The edge index of validation message edges has shape: {}".
↪format(dataset_val[0].edge_index.shape))
print("The edge index of validation supervision edges has shape: {}".
↪format(val_sup_edge_index.shape))
print("The edge index of test message edges has shape: {}".
↪format(dataset_test[0].edge_index.shape))
```

```
print("The edge index of test supervision edges has shape: {}".
 ↪format(test_sup_edge_index.shape))
```

```
The edge index of original graph has shape: torch.Size([2, 10556])
The edge index of training message edges has shape: torch.Size([2, 6754])
The edge index of training supervision edges has shape: torch.Size([2, 3380])
The edge index of validation message edges has shape: torch.Size([2, 8444])
The edge index of validation supervision edges has shape: torch.Size([2, 2108])
The edge index of test message edges has shape: torch.Size([2, 9498])
The edge index of test supervision edges has shape: torch.Size([2, 2116])
```

You can see that the training / validation message passing edges and training supervision edges are splitted differently in those two modes!

5.6.2 Resample Negative Edges

During each training iteration, we usually need to resample the negative edges.

Below we print the training and validation sets negative edges in two training iterations.

You should find that the negative edges in training set will be resampled.

[30]:
```
dataset = GraphDataset(graphs, task='link_pred', edge_train_mode="disjoint")
datasets = {}
follow_batch = []
datasets['train'], datasets['val'], datasets['test'] = dataset.split(
    transductive=True, split_ratio=[0.8, 0.1, 0.1])
dataloaders = {
  split: DataLoader(
    ds, collate_fn=Batch.collate(follow_batch),
    batch_size=1, shuffle=(split=='train')
  )
  for split, ds in datasets.items()
}
neg_edges_1 = None
for batch in dataloaders['train']:
  num_edges = batch.edge_label_index.shape[1] // 2
  neg_edges_1 = batch.edge_label_index[:, num_edges:]
  print("First iteration training negative edges:")
  print(neg_edges_1)
  break
neg_edges_2 = None
for batch in dataloaders['train']:
  num_edges = batch.edge_label_index.shape[1] // 2
  neg_edges_2 = batch.edge_label_index[:, num_edges:]
  print("Second iteration training negative edges:")
  print(neg_edges_2)
  break
```

```python
neg_edges_1 = None
for batch in dataloaders['val']:
  num_edges = batch.edge_label_index.shape[1] // 2
  neg_edges_1 = batch.edge_label_index[:, num_edges:]
  print("First iteration validation negative edges:")
  print(neg_edges_1)
  break
neg_edges_2 = None
for batch in dataloaders['val']:
  num_edges = batch.edge_label_index.shape[1] // 2
  neg_edges_2 = batch.edge_label_index[:, num_edges:]
  print("Second iteration validation negative edges:")
  print(neg_edges_2)
  break
```

```
First iteration training negative edges:
tensor([[1885, 1546, 2175,  …,  151,   48, 1436],
        [2363, 1027,  347,  …, 1603, 2609, 2138]])
Second iteration training negative edges:
tensor([[1059, 2543, 1496,  …, 2520, 2072,  348],
        [ 768,  330, 1624,  …, 1498,  649,   27]])
First iteration validation negative edges:
tensor([[2282,  558,  641,  …, 1590,   30,  364],
        [1986, 2308, 1206,  …,   73,  822, 2651]])
Second iteration validation negative edges:
tensor([[2282,  558,  641,  …, 1590,   30,  364],
        [1986, 2308, 1206,  …,   73,  822, 2651]])
```

If you are interested in more graph splitting settings, please refer to the DeepSNAP dataset documentation.

5.7 Graph Transformation and Feature Computation

The other DeepSNAP core functionality is graph transformation / feature computation.

In DeepSNAP, we divide graph transformation / feature computation into two different types. One is the transformation before training (transform the whole dataset before training directly) and another one is the transformation during training (transform batches of graphs).

Here is an example that uses NetworkX back end to calculate the PageRank value and update the value to tensors before the training (transform the dataset).

[31]:
```python
def pagerank_transform_fn(graph):

  # Get the referenced networkx graph
  G = graph.G

  # Calculate the pagerank by using networkx
  pr = nx.pagerank(G)
```

```
  # Transform the pagerank values to tensor
  pr_feature = torch.tensor([pr[node] for node in range(graph.num_nodes)],␣
↪dtype=torch.float32)
  pr_feature = pr_feature.view(graph.num_nodes, 1)

  # Concat the pagerank values to the node feature
  graph.node_feature = torch.cat([graph.node_feature, pr_feature], dim=-1)

root = './tmp/cox2'
name = 'COX2'
pyg_dataset = TUDataset(root, name)
graphs = GraphDataset.pyg_to_graphs(pyg_dataset)
dataset = GraphDataset(graphs, task='graph')
print("Number of features before transformation: {}".format(dataset.
↪num_node_features))
dataset.apply_transform(pagerank_transform_fn, update_tensor=False)
print("Number of features after transformation: {}".format(dataset.
↪num_node_features))
```

```
Number of features before transformation: 35
Number of features after transformation: 36
```

5.8 Question 3.6: Implement the transformation below and report the clustering coefficient of the node (index 3) of the graph (index 406) in the COX2 dataset. Rounded the answer to two decimal places. (5 points)

[32]:
```
def cluster_transform_fn(graph):
  # TODO: Implement this function that takes an deepsnap graph object,
  # transform the graph by adding nodes clustering coefficient into the
  # graph.node_feature

  ############# Your code here ############
  ## (~5 lines of code)
  ## Note
  ## 1. Compute the clustering coefficient value for each node and
  ## concat them to the last dimension of graph.node_feature
    G = graph.G
    cc = nx.clustering(G)
    cc_feature = torch.tensor([cc[node] for node in range(graph.num_nodes)],␣
↪dtype=torch.float32)
    cc_feature = cc_feature.view(graph.num_nodes, 1)
    graph.node_features = torch.cat([graph.node_feature, cc_feature], dim=-1)

  #########################################

root = './cox2'
```

```
name = 'COX2'
pyg_dataset = TUDataset(root, name)
graphs = GraphDataset.pyg_to_graphs(pyg_dataset)
dataset = GraphDataset(graphs, task='graph')

# Transform the dataset
dataset.apply_transform(cluster_transform_fn, update_tensor=False)

node_idx = 3
graph_idx = 406
node_feature = dataset[graph_idx].node_feature

print("The node has clustering coefficient: {}".
 ↪format(round(node_feature[node_idx][-1].item(), 2)))
```

```
Downloading https://www.chrsmrrs.com/graphkerneldatasets/COX2.zip
Extracting cox2/COX2/COX2.zip
Processing…
Done!
```

```
The node has clustering coefficient: 0.0
```

Apart from transforming the dataset, DeepSNAP can also transform the graph (usually the `deepsnap.batch.Batch`) during each training iteration.

Also, DeepSNAP supports the synchronization of the transformation between the referenced graph objects and tensor representations. For example, you can just update the NetworkX graph object in the transform function, and by specifying `update_tensor=True` the internal tensor representations will be automatically updated.

For more information, please refer to the DeepSNAP documentation.

6 4 Edge Level Prediction

From last section, we know how DeepSNAP transductive split the edges in the link prediction task.

Now lets use DeepSNAP and PyG together to implement a edge level prediction (link prediction) model!

[33]:
```
import copy
import torch
import numpy as np
import networkx as nx
import matplotlib.pyplot as plt

from deepsnap.graph import Graph
from deepsnap.batch import Batch
from deepsnap.dataset import GraphDataset
from torch_geometric.datasets import Planetoid, TUDataset
```

```python
from torch.utils.data import DataLoader

import torch.nn.functional as F
from torch_geometric.nn import SAGEConv

class LinkPredModel(torch.nn.Module):
    def __init__(self, input_dim, hidden_dim, num_classes, dropout=0.2):
        super(LinkPredModel, self).__init__()

        self.conv1 = SAGEConv(input_dim, hidden_dim)
        self.conv2 = SAGEConv(hidden_dim, num_classes)

        self.loss_fn = None

        ############# Your code here ############
        ## (~1 line of code)
        ## Note
        ## 1. Initialize the loss function to BCEWithLogitsLoss
        self.loss_fn = nn.BCEWithLogitsLoss()

        ##########################################

        self.dropout = dropout

    def reset_parameters(self):
        self.conv1.reset_parameters()
        self.conv2.reset_parameters()

    def forward(self, batch):
        node_feature, edge_index, edge_label_index = batch.node_feature, batch.
↪edge_index, batch.edge_label_index

        ############# Your code here ############
        ## (~6 line of code)
        ## Note
        ## 1. Feed the node feature into the first conv layer
        ## 2. Add a ReLU after the first conv layer
        ## 3. Add dropout after the ReLU (with probability self.dropout)
        ## 4. Feed the output to the second conv layer
        ## 5. Select the embeddings of the source nodes and destination nodes
        ## by using the edge_label_index and compute the similarity of each pair
        ## by dot product
        out = self.conv1(node_feature, edge_index)
        out = F.relu(out)
        out = F.dropout(out, p=self.dropout)
        out = self.conv2(out, edge_index)
        out = out[edge_label_index[0]] * out[edge_label_index[1]]
```

```
        pred = torch.sum(out, -1)


        ##########################################

        return pred

    def loss(self, pred, link_label):
        return self.loss_fn(pred, link_label)
```

[35]:
```
from sklearn.metrics import *

def train(model, dataloaders, optimizer, args):
    val_max = 0
    best_model = model

    for epoch in range(1, args["epochs"]):
        for i, batch in enumerate(dataloaders['train']):

            batch.to(args["device"])

            ############# Your code here ############
            ## (~6 lines of code)
            ## Note
            ## 1. Zero grad the optimizer
            ## 2. Compute loss and backpropagate
            ## 3. Update the model parameters
            optimizer.zero_grad()
            pred = model(batch)
            loss = model.loss(pred, batch.edge_label.float())
            loss.backward()
            optimizer.step()



            ##########################################

            log = 'Epoch: {:03d}, Train: {:.4f}, Val: {:.4f}, Test: {:.4f},␣
 ↪Loss: {}'
            score_train = test(model, dataloaders['train'], args)
            score_val = test(model, dataloaders['val'], args)
            score_test = test(model, dataloaders['test'], args)

            print(log.format(epoch, score_train, score_val, score_test, loss.
 ↪item()))
            if val_max < score_val:
                val_max = score_val
```

```
                best_model = copy.deepcopy(model)
    return best_model

def test(model, dataloader, args):
    model.eval()

    score = 0

    ############# Your code here ############
    ## (~5 lines of code)
    ## Note
    ## 1. Loop through batches in the dataloader
    ## 2. Feed the batch to the model
    ## 3. Feed the model output to sigmoid
    ## 4. Compute the ROC-AUC score by using sklearn roc_auc_score function
    ## 5. Edge labels are stored in batch.edge_label
    for i, batch in enumerate(dataloaders['test']):
        batch.to(args["device"])
        pred = model(batch)
        pred = torch.sigmoid(pred)
        score += roc_auc_score(batch.edge_label.cpu().detach().numpy(), pred.
↪cpu().detach().numpy())
    score /= len(dataloaders['test'])


    ##########################################

    return score
```

[36]:

```
# Please don't change any parameters
args = {
    "device" : 'cuda' if torch.cuda.is_available() else 'cpu',
    "hidden_dim" : 128,
    "epochs" : 200,
}
```

[37]:

```
pyg_dataset = Planetoid('./tmp/cora', 'Cora')
graphs = GraphDataset.pyg_to_graphs(pyg_dataset)

dataset = GraphDataset(
        graphs,
        task='link_pred',
        edge_train_mode="disjoint"
    )
datasets = {}
datasets['train'], datasets['val'], datasets['test']= dataset.split(
            transductive=True, split_ratio=[0.85, 0.05, 0.1])
```

```python
input_dim = datasets['train'].num_node_features
num_classes = datasets['train'].num_edge_labels

model = LinkPredModel(input_dim, args["hidden_dim"], num_classes).
 ↪to(args["device"])
model.reset_parameters()

optimizer = torch.optim.SGD(model.parameters(), lr=0.1, momentum=0.9,␣
 ↪weight_decay=5e-4)

dataloaders = {split: DataLoader(
            ds, collate_fn=Batch.collate([]),
            batch_size=1, shuffle=(split=='train'))
            for split, ds in datasets.items()}
best_model = train(model, dataloaders, optimizer, args)
log = "Train: {:.4f}, Val: {:.4f}, Test: {:.4f}"
best_train_roc = test(best_model, dataloaders['train'], args)
best_val_roc = test(best_model, dataloaders['val'], args)
best_test_roc = test(best_model, dataloaders['test'], args)
print(log.format(best_train_roc, best_val_roc, best_test_roc))
```

```
Epoch: 001, Train: 0.5268, Val: 0.5335, Test: 0.5404, Loss: 0.6930190920829773
Epoch: 002, Train: 0.5374, Val: 0.5359, Test: 0.5394, Loss: 0.6930177807807922
Epoch: 003, Train: 0.5423, Val: 0.5289, Test: 0.5404, Loss: 0.6930444836616516
Epoch: 004, Train: 0.5382, Val: 0.5325, Test: 0.5315, Loss: 0.6930093765258789
```

```
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
```

```
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
```

```
Epoch: 005, Train: 0.5394, Val: 0.5337, Test: 0.5315, Loss: 0.693001389503479
Epoch: 006, Train: 0.5513, Val: 0.5494, Test: 0.5243, Loss: 0.6930056810379028
Epoch: 007, Train: 0.5493, Val: 0.5348, Test: 0.5308, Loss: 0.6929584741592407
Epoch: 008, Train: 0.5425, Val: 0.5293, Test: 0.5531, Loss: 0.6930720210075378
```

```
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
```

```
Epoch: 009, Train: 0.5392, Val: 0.5298, Test: 0.5436, Loss: 0.6929808259010315
Epoch: 010, Train: 0.5363, Val: 0.5512, Test: 0.5408, Loss: 0.692920446395874
Epoch: 011, Train: 0.5511, Val: 0.5433, Test: 0.5155, Loss: 0.6929579973220825
Epoch: 012, Train: 0.5295, Val: 0.5365, Test: 0.5483, Loss: 0.6929038166999817
```

```
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
```

```
Epoch: 013, Train: 0.5554, Val: 0.5456, Test: 0.5487, Loss: 0.6928958296775818
Epoch: 014, Train: 0.5388, Val: 0.5412, Test: 0.5456, Loss: 0.6928473114967346
Epoch: 015, Train: 0.5360, Val: 0.5551, Test: 0.5576, Loss: 0.6927154660224915
Epoch: 016, Train: 0.5554, Val: 0.5516, Test: 0.5556, Loss: 0.692834734916687
```

```
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
```

```
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
```

```
Epoch: 017, Train: 0.5462, Val: 0.5373, Test: 0.5372, Loss: 0.6928471326828003
Epoch: 018, Train: 0.5359, Val: 0.5411, Test: 0.5568, Loss: 0.6928033232688904
Epoch: 019, Train: 0.5476, Val: 0.5467, Test: 0.5499, Loss: 0.6928157806396484
Epoch: 020, Train: 0.5527, Val: 0.5494, Test: 0.5382, Loss: 0.692692220211029
```

```
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
```

```
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
```

```
Epoch: 021, Train: 0.5454, Val: 0.5465, Test: 0.5509, Loss: 0.6926034092903137
Epoch: 022, Train: 0.5527, Val: 0.5427, Test: 0.5496, Loss: 0.6925811767578125
Epoch: 023, Train: 0.5501, Val: 0.5500, Test: 0.5678, Loss: 0.6924965381622314
Epoch: 024, Train: 0.5427, Val: 0.5614, Test: 0.5501, Loss: 0.6924853324890137
```

```
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
```

```
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
```

```
Epoch: 025, Train: 0.5582, Val: 0.5393, Test: 0.5638, Loss: 0.6924364566802979
Epoch: 026, Train: 0.5665, Val: 0.5692, Test: 0.5504, Loss: 0.6923744678497314
Epoch: 027, Train: 0.5587, Val: 0.5630, Test: 0.5643, Loss: 0.6921926736831665
Epoch: 028, Train: 0.5726, Val: 0.5521, Test: 0.5585, Loss: 0.6921568512916565
```

```
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
```

```
Epoch: 029, Train: 0.5564, Val: 0.5500, Test: 0.5400, Loss: 0.6921823024749756
Epoch: 030, Train: 0.5586, Val: 0.5630, Test: 0.5692, Loss: 0.6921277642250061
Epoch: 031, Train: 0.5640, Val: 0.5593, Test: 0.5596, Loss: 0.6919339895248413
```

```
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
```

```
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes

Epoch: 032, Train: 0.5663, Val: 0.5537, Test: 0.5608, Loss: 0.6918551325798035
Epoch: 033, Train: 0.5652, Val: 0.5572, Test: 0.5665, Loss: 0.6916908621788025
Epoch: 034, Train: 0.5569, Val: 0.5508, Test: 0.5624, Loss: 0.6917552351951599
Epoch: 035, Train: 0.5543, Val: 0.5689, Test: 0.5736, Loss: 0.6916505694389343

/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
```

```
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
```

```
Epoch: 036, Train: 0.5638, Val: 0.5619, Test: 0.5607, Loss: 0.6913653612136841
Epoch: 037, Train: 0.5628, Val: 0.5689, Test: 0.5624, Loss: 0.6914324164390564
Epoch: 038, Train: 0.5698, Val: 0.5546, Test: 0.5655, Loss: 0.6910073161125183
Epoch: 039, Train: 0.5712, Val: 0.5599, Test: 0.5600, Loss: 0.6909275054931641
```

```
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
```

```
Epoch: 040, Train: 0.5668, Val: 0.5601, Test: 0.5590, Loss: 0.6904020309448242
Epoch: 041, Train: 0.5616, Val: 0.5656, Test: 0.5736, Loss: 0.6903499960899353
Epoch: 042, Train: 0.5679, Val: 0.5678, Test: 0.5615, Loss: 0.6903259754180908
Epoch: 043, Train: 0.5618, Val: 0.5556, Test: 0.5581, Loss: 0.6896516680717468
```

```
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
```

```
Epoch: 044, Train: 0.5626, Val: 0.5614, Test: 0.5613, Loss: 0.6895309090614319
Epoch: 045, Train: 0.5765, Val: 0.5666, Test: 0.5549, Loss: 0.689214289188385
Epoch: 046, Train: 0.5649, Val: 0.5689, Test: 0.5632, Loss: 0.6890933513641357
Epoch: 047, Train: 0.5677, Val: 0.5762, Test: 0.5596, Loss: 0.6889774203300476
```

```
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
```

```
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
```

```
Epoch: 048, Train: 0.5664, Val: 0.5683, Test: 0.5740, Loss: 0.6880175471305847
Epoch: 049, Train: 0.5713, Val: 0.5661, Test: 0.5712, Loss: 0.6883893013000488
Epoch: 050, Train: 0.5571, Val: 0.5691, Test: 0.5668, Loss: 0.6869678497314453
Epoch: 051, Train: 0.5698, Val: 0.5674, Test: 0.5707, Loss: 0.6870943307876587
```

```
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
```

```
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
```

```
Epoch: 052, Train: 0.5668, Val: 0.5720, Test: 0.5742, Loss: 0.6868032813072205
Epoch: 053, Train: 0.5721, Val: 0.5644, Test: 0.5657, Loss: 0.6864400506019592
Epoch: 054, Train: 0.5809, Val: 0.5759, Test: 0.5655, Loss: 0.6847174167633057
Epoch: 055, Train: 0.5721, Val: 0.5699, Test: 0.5800, Loss: 0.6848193407058716
```

```
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
```

```
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
```

```
Epoch: 056, Train: 0.5720, Val: 0.5784, Test: 0.5744, Loss: 0.6840272545814514
Epoch: 057, Train: 0.5815, Val: 0.5735, Test: 0.5765, Loss: 0.6827371120452881
Epoch: 058, Train: 0.5763, Val: 0.5808, Test: 0.5794, Loss: 0.6827693581581116
Epoch: 059, Train: 0.5771, Val: 0.5858, Test: 0.5820, Loss: 0.6804792284965515
```

```
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
```

```
Epoch: 060, Train: 0.5880, Val: 0.5863, Test: 0.5876, Loss: 0.6804771423339844
Epoch: 061, Train: 0.5908, Val: 0.5885, Test: 0.5877, Loss: 0.6788559556007385
Epoch: 062, Train: 0.5907, Val: 0.5886, Test: 0.5902, Loss: 0.6785147786140442
Epoch: 063, Train: 0.5943, Val: 0.5988, Test: 0.5999, Loss: 0.6770277619361877
```

```
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
```

```
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes

Epoch: 064, Train: 0.6029, Val: 0.6032, Test: 0.6064, Loss: 0.6762038469314575
Epoch: 065, Train: 0.6089, Val: 0.6157, Test: 0.6065, Loss: 0.6711689233779907
Epoch: 066, Train: 0.6194, Val: 0.6184, Test: 0.6162, Loss: 0.6717539429664612

/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
```

```
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
```

```
Epoch: 067, Train: 0.6283, Val: 0.6330, Test: 0.6273, Loss: 0.6706377267837524
Epoch: 068, Train: 0.6341, Val: 0.6346, Test: 0.6349, Loss: 0.6696358919143677
Epoch: 069, Train: 0.6493, Val: 0.6478, Test: 0.6490, Loss: 0.6683998107910156
```

```
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
```

```
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes

Epoch: 070, Train: 0.6545, Val: 0.6555, Test: 0.6629, Loss: 0.6624838709831238
Epoch: 071, Train: 0.6662, Val: 0.6684, Test: 0.6643, Loss: 0.6571731567382812
Epoch: 072, Train: 0.6693, Val: 0.6625, Test: 0.6674, Loss: 0.6578211784362793
Epoch: 073, Train: 0.6844, Val: 0.6734, Test: 0.6773, Loss: 0.6518868803977966

/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes

Epoch: 074, Train: 0.6823, Val: 0.6809, Test: 0.6802, Loss: 0.6516441106796265
Epoch: 075, Train: 0.6817, Val: 0.6823, Test: 0.6834, Loss: 0.6477174758911133
Epoch: 076, Train: 0.6841, Val: 0.6833, Test: 0.6846, Loss: 0.6425758600234985

/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
```

```
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes

Epoch: 077, Train: 0.6821, Val: 0.6870, Test: 0.6873, Loss: 0.6386141777038574
Epoch: 078, Train: 0.6902, Val: 0.6880, Test: 0.6837, Loss: 0.6279609799385071
Epoch: 079, Train: 0.6915, Val: 0.6889, Test: 0.6912, Loss: 0.6322352290153503

/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
```

```
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
```

```
Epoch: 080, Train: 0.6889, Val: 0.6961, Test: 0.6873, Loss: 0.6288654208183289
Epoch: 081, Train: 0.6922, Val: 0.6900, Test: 0.6947, Loss: 0.6267640590667725
Epoch: 082, Train: 0.6926, Val: 0.6942, Test: 0.6986, Loss: 0.6276502013206482
Epoch: 083, Train: 0.6959, Val: 0.6937, Test: 0.6941, Loss: 0.6141358017921448
```

```
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
```

```
Epoch: 084, Train: 0.6971, Val: 0.6969, Test: 0.6944, Loss: 0.6220294237136841
Epoch: 085, Train: 0.7005, Val: 0.6970, Test: 0.7017, Loss: 0.6170891523361206
Epoch: 086, Train: 0.7038, Val: 0.7041, Test: 0.6996, Loss: 0.6162652373313904
```

```
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
```

```
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
```

```
Epoch: 087, Train: 0.7058, Val: 0.7034, Test: 0.7037, Loss: 0.6139217019081116
Epoch: 088, Train: 0.7086, Val: 0.7056, Test: 0.7103, Loss: 0.6154441833496094
Epoch: 089, Train: 0.7150, Val: 0.7128, Test: 0.7154, Loss: 0.6016436219215393
Epoch: 090, Train: 0.7161, Val: 0.7176, Test: 0.7148, Loss: 0.6029725670814514
```

```
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
```

```
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
```

```
Epoch: 091, Train: 0.7190, Val: 0.7156, Test: 0.7139, Loss: 0.6105474233627319
Epoch: 092, Train: 0.7202, Val: 0.7209, Test: 0.7190, Loss: 0.6028561592102051
Epoch: 093, Train: 0.7257, Val: 0.7275, Test: 0.7239, Loss: 0.6034415364265442
Epoch: 094, Train: 0.7274, Val: 0.7302, Test: 0.7286, Loss: 0.6000949740409851
```

```
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
```

```
Epoch: 095, Train: 0.7301, Val: 0.7313, Test: 0.7301, Loss: 0.5927562117576599
```

```
Epoch: 096, Train: 0.7303, Val: 0.7334, Test: 0.7327, Loss: 0.5974225997924805
Epoch: 097, Train: 0.7328, Val: 0.7341, Test: 0.7341, Loss: 0.5938940644264221
```

```
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
```

```
Epoch: 098, Train: 0.7389, Val: 0.7339, Test: 0.7381, Loss: 0.5967804193496704
Epoch: 099, Train: 0.7419, Val: 0.7377, Test: 0.7420, Loss: 0.5906982421875
Epoch: 100, Train: 0.7402, Val: 0.7394, Test: 0.7366, Loss: 0.5913636684417725
Epoch: 101, Train: 0.7399, Val: 0.7418, Test: 0.7387, Loss: 0.5927600264549255
```

```
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
```

```
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
```

```
Epoch: 102, Train: 0.7409, Val: 0.7421, Test: 0.7388, Loss: 0.5940408110618591
Epoch: 103, Train: 0.7407, Val: 0.7410, Test: 0.7416, Loss: 0.5813894867897034
Epoch: 104, Train: 0.7412, Val: 0.7459, Test: 0.7456, Loss: 0.5862178206443787
Epoch: 105, Train: 0.7447, Val: 0.7433, Test: 0.7460, Loss: 0.5875080227851868
```

```
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
```

```
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
```

```
Epoch: 106, Train: 0.7438, Val: 0.7462, Test: 0.7424, Loss: 0.5725404620170593
Epoch: 107, Train: 0.7477, Val: 0.7474, Test: 0.7499, Loss: 0.5821047425270081
Epoch: 108, Train: 0.7516, Val: 0.7503, Test: 0.7495, Loss: 0.5746185779571533
```

```
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
```

```
Epoch: 109, Train: 0.7483, Val: 0.7493, Test: 0.7479, Loss: 0.5771692991256714
Epoch: 110, Train: 0.7492, Val: 0.7517, Test: 0.7475, Loss: 0.5635170936584473
Epoch: 111, Train: 0.7476, Val: 0.7515, Test: 0.7517, Loss: 0.5708810091018677
Epoch: 112, Train: 0.7501, Val: 0.7495, Test: 0.7515, Loss: 0.5776768326759338
```

```
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
```

```
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes

Epoch: 113, Train: 0.7497, Val: 0.7500, Test: 0.7505, Loss: 0.5812559127807617
Epoch: 114, Train: 0.7536, Val: 0.7445, Test: 0.7512, Loss: 0.5627340078353882
Epoch: 115, Train: 0.7518, Val: 0.7481, Test: 0.7488, Loss: 0.5727423429489136
Epoch: 116, Train: 0.7478, Val: 0.7516, Test: 0.7509, Loss: 0.5653689503669739

/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
```

```
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
```

```
Epoch: 117, Train: 0.7481, Val: 0.7494, Test: 0.7466, Loss: 0.579186737537384
Epoch: 118, Train: 0.7511, Val: 0.7452, Test: 0.7488, Loss: 0.5611461997032166
Epoch: 119, Train: 0.7495, Val: 0.7530, Test: 0.7476, Loss: 0.5589011907577515
Epoch: 120, Train: 0.7484, Val: 0.7519, Test: 0.7528, Loss: 0.5720475912094116
```

```
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
```

```
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes

Epoch: 121, Train: 0.7526, Val: 0.7509, Test: 0.7488, Loss: 0.566372275352478
Epoch: 122, Train: 0.7524, Val: 0.7519, Test: 0.7461, Loss: 0.5681602954864502
Epoch: 123, Train: 0.7471, Val: 0.7491, Test: 0.7488, Loss: 0.5537976622581482
Epoch: 124, Train: 0.7490, Val: 0.7509, Test: 0.7515, Loss: 0.5547548532485962

/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes

Epoch: 125, Train: 0.7491, Val: 0.7470, Test: 0.7454, Loss: 0.5611616373062134
Epoch: 126, Train: 0.7509, Val: 0.7541, Test: 0.7485, Loss: 0.5679927468299866
```

```
Epoch: 127, Train: 0.7513, Val: 0.7530, Test: 0.7480, Loss: 0.548517644405365
Epoch: 128, Train: 0.7484, Val: 0.7485, Test: 0.7541, Loss: 0.5426265597343445
```

```
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
```

```
Epoch: 129, Train: 0.7495, Val: 0.7497, Test: 0.7495, Loss: 0.5573952198028564
Epoch: 130, Train: 0.7476, Val: 0.7493, Test: 0.7488, Loss: 0.5519223809242249
Epoch: 131, Train: 0.7494, Val: 0.7503, Test: 0.7513, Loss: 0.5421897172927856
Epoch: 132, Train: 0.7488, Val: 0.7502, Test: 0.7495, Loss: 0.5451341271400452
```

```
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
```

```
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
```

```
Epoch: 133, Train: 0.7530, Val: 0.7550, Test: 0.7504, Loss: 0.5559791922569275
Epoch: 134, Train: 0.7506, Val: 0.7513, Test: 0.7526, Loss: 0.5580904483795166
Epoch: 135, Train: 0.7517, Val: 0.7513, Test: 0.7460, Loss: 0.5440338850021362
Epoch: 136, Train: 0.7517, Val: 0.7533, Test: 0.7494, Loss: 0.5406719446182251
```

```
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
```

```
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
```

```
Epoch: 137, Train: 0.7515, Val: 0.7456, Test: 0.7512, Loss: 0.54180508852005
Epoch: 138, Train: 0.7517, Val: 0.7506, Test: 0.7518, Loss: 0.5503693222999573
Epoch: 139, Train: 0.7493, Val: 0.7490, Test: 0.7488, Loss: 0.549671471118927
Epoch: 140, Train: 0.7527, Val: 0.7536, Test: 0.7516, Loss: 0.5533307194709778
```

```
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
```

```
rounding_mode='floor').
  row = perm // num_nodes

Epoch: 141, Train: 0.7532, Val: 0.7522, Test: 0.7510, Loss: 0.5456833243370056
Epoch: 142, Train: 0.7484, Val: 0.7502, Test: 0.7544, Loss: 0.5447911620140076
Epoch: 143, Train: 0.7529, Val: 0.7482, Test: 0.7510, Loss: 0.5440102219581604
Epoch: 144, Train: 0.7509, Val: 0.7486, Test: 0.7517, Loss: 0.5468860268592834

/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes

Epoch: 145, Train: 0.7514, Val: 0.7529, Test: 0.7543, Loss: 0.550417959690094
Epoch: 146, Train: 0.7562, Val: 0.7501, Test: 0.7514, Loss: 0.5524005889892578
Epoch: 147, Train: 0.7522, Val: 0.7538, Test: 0.7522, Loss: 0.5382256507873535
Epoch: 148, Train: 0.7510, Val: 0.7544, Test: 0.7527, Loss: 0.5360032916069031

/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
```

```
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
```

```
Epoch: 149, Train: 0.7494, Val: 0.7527, Test: 0.7494, Loss: 0.5368065237998962
Epoch: 150, Train: 0.7514, Val: 0.7526, Test: 0.7482, Loss: 0.5270652174949646
Epoch: 151, Train: 0.7483, Val: 0.7502, Test: 0.7469, Loss: 0.5309561491012573
Epoch: 152, Train: 0.7519, Val: 0.7506, Test: 0.7514, Loss: 0.5352708101272583
```

```
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
```

```
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
```

```
Epoch: 153, Train: 0.7488, Val: 0.7473, Test: 0.7510, Loss: 0.527461051940918
Epoch: 154, Train: 0.7540, Val: 0.7492, Test: 0.7513, Loss: 0.5275353789329529
Epoch: 155, Train: 0.7500, Val: 0.7507, Test: 0.7458, Loss: 0.5335352420806885
```

```
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
```

```
Epoch: 156, Train: 0.7459, Val: 0.7507, Test: 0.7497, Loss: 0.5302559733390808
Epoch: 157, Train: 0.7500, Val: 0.7492, Test: 0.7513, Loss: 0.5390960574150085
Epoch: 158, Train: 0.7474, Val: 0.7505, Test: 0.7490, Loss: 0.536928653717041
Epoch: 159, Train: 0.7475, Val: 0.7515, Test: 0.7533, Loss: 0.5188227891921997
```

```
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
```

```
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
```

```
Epoch: 160, Train: 0.7478, Val: 0.7500, Test: 0.7488, Loss: 0.5214498043060303
Epoch: 161, Train: 0.7456, Val: 0.7468, Test: 0.7458, Loss: 0.5186651945114136
Epoch: 162, Train: 0.7412, Val: 0.7426, Test: 0.7440, Loss: 0.5224072337150574
Epoch: 163, Train: 0.7436, Val: 0.7466, Test: 0.7461, Loss: 0.5181321501731873
```

```
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
```

```
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
```

```
Epoch: 164, Train: 0.7426, Val: 0.7491, Test: 0.7479, Loss: 0.522769033908844
Epoch: 165, Train: 0.7427, Val: 0.7432, Test: 0.7454, Loss: 0.5243502855300903
Epoch: 166, Train: 0.7468, Val: 0.7449, Test: 0.7500, Loss: 0.5171862840652466
Epoch: 167, Train: 0.7453, Val: 0.7496, Test: 0.7439, Loss: 0.5222576856613159
```

```
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
```

```
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
```

```
Epoch: 168, Train: 0.7464, Val: 0.7445, Test: 0.7440, Loss: 0.5271921753883362
Epoch: 169, Train: 0.7407, Val: 0.7429, Test: 0.7465, Loss: 0.5229383707046509
Epoch: 170, Train: 0.7411, Val: 0.7436, Test: 0.7462, Loss: 0.5199507474899292
```

```
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
```

```
Epoch: 171, Train: 0.7425, Val: 0.7443, Test: 0.7442, Loss: 0.5142338275909424
Epoch: 172, Train: 0.7404, Val: 0.7463, Test: 0.7431, Loss: 0.5178573131561279
Epoch: 173, Train: 0.7417, Val: 0.7393, Test: 0.7442, Loss: 0.51780104637146
Epoch: 174, Train: 0.7432, Val: 0.7315, Test: 0.7378, Loss: 0.5141475200653076
```

```
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
```

```
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
```

```
Epoch: 175, Train: 0.7409, Val: 0.7384, Test: 0.7406, Loss: 0.5155263543128967
Epoch: 176, Train: 0.7402, Val: 0.7358, Test: 0.7408, Loss: 0.5119300484657288
Epoch: 177, Train: 0.7405, Val: 0.7425, Test: 0.7369, Loss: 0.5293910503387451
Epoch: 178, Train: 0.7407, Val: 0.7356, Test: 0.7448, Loss: 0.515716016292572
```

```
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
```

```
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
```

```
Epoch: 179, Train: 0.7374, Val: 0.7420, Test: 0.7335, Loss: 0.5127541422843933
Epoch: 180, Train: 0.7430, Val: 0.7420, Test: 0.7345, Loss: 0.5103680491447449
Epoch: 181, Train: 0.7421, Val: 0.7424, Test: 0.7457, Loss: 0.5196908712387085
Epoch: 182, Train: 0.7355, Val: 0.7346, Test: 0.7364, Loss: 0.5129074454307556
```

```
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
```

```
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
```

```
Epoch: 183, Train: 0.7361, Val: 0.7359, Test: 0.7384, Loss: 0.5240173935890198
Epoch: 184, Train: 0.7397, Val: 0.7383, Test: 0.7363, Loss: 0.5076257586479187
Epoch: 185, Train: 0.7416, Val: 0.7353, Test: 0.7376, Loss: 0.5035434365272522
Epoch: 186, Train: 0.7382, Val: 0.7390, Test: 0.7352, Loss: 0.5019108057022095
```

```
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
```

```
Epoch: 187, Train: 0.7354, Val: 0.7343, Test: 0.7377, Loss: 0.5153831839561462
Epoch: 188, Train: 0.7412, Val: 0.7369, Test: 0.7354, Loss: 0.5110770463943481
Epoch: 189, Train: 0.7344, Val: 0.7391, Test: 0.7360, Loss: 0.49273478984832764
```

```
Epoch: 190, Train: 0.7311, Val: 0.7283, Test: 0.7361, Loss: 0.5020902752876282
```

```
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
```

```
Epoch: 191, Train: 0.7335, Val: 0.7340, Test: 0.7337, Loss: 0.5107552409172058
Epoch: 192, Train: 0.7347, Val: 0.7416, Test: 0.7427, Loss: 0.5051699280738831
Epoch: 193, Train: 0.7356, Val: 0.7312, Test: 0.7356, Loss: 0.4966055154800415
Epoch: 194, Train: 0.7400, Val: 0.7331, Test: 0.7330, Loss: 0.4971802532672882
```

```
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
```

```
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
```

```
Epoch: 195, Train: 0.7344, Val: 0.7375, Test: 0.7340, Loss: 0.5044193267822266
Epoch: 196, Train: 0.7324, Val: 0.7365, Test: 0.7327, Loss: 0.5091291069984436
Epoch: 197, Train: 0.7363, Val: 0.7375, Test: 0.7334, Loss: 0.4940577447414398
Epoch: 198, Train: 0.7348, Val: 0.7328, Test: 0.7284, Loss: 0.4996934235095978
Epoch: 199, Train: 0.7349, Val: 0.7371, Test: 0.7330, Loss: 0.494052916765213
Train: 0.7515, Val: 0.7501, Test: 0.7500
```

```
/home/arch/anaconda3/envs/GNN_env/lib/python3.8/site-
packages/deepsnap/graph.py:2126: UserWarning: __floordiv__ is deprecated, and
its behavior will change in a future version of pytorch. It currently rounds
toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values. To keep the current behavior, use torch.div(a, b,
rounding_mode='trunc'), or for actual floor division, use torch.div(a, b,
rounding_mode='floor').
  row = perm // num_nodes
```

6.1 Question 4: What is the maximum ROC-AUC score you could get for the best_model on test set? (13 points)

Submit your answers on Gradescope.

The best AUC score for the best_model on the test set is 0.75

7 Submission

In order to get credit, you must go submit your answers on Gradescope.

Also, you need to submit the `ipynb` file of Colab 3, by clicking `File` and `Download .ipynb`. Please make sure that your output of each cell is available in your `ipynb` file.