GCN, GAT, GraphSAGE 框架回顾及其 PyG 复现

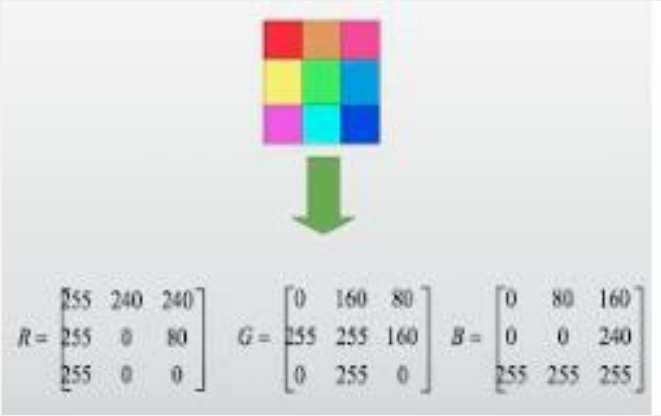
报告人：王硕

Index

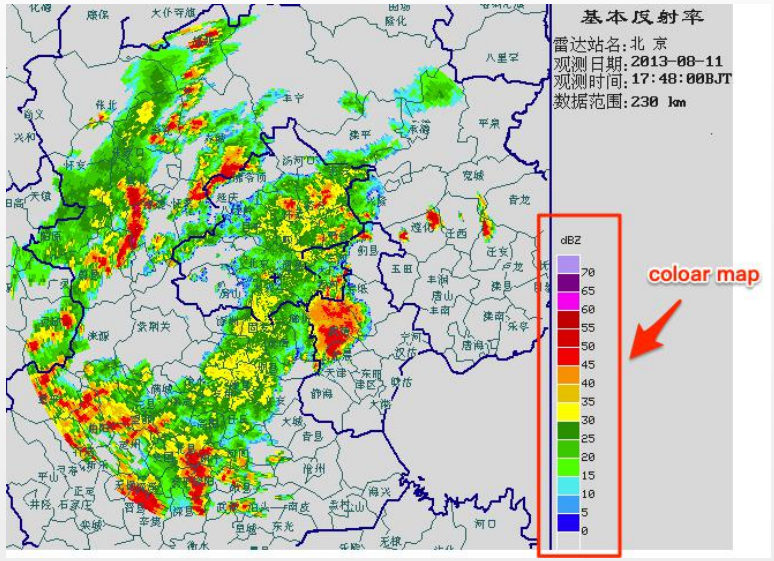
- 图表示学习
- 图神经网络的发展
- PyG 图神经网络开发框架
- cora 节点分类代码实例
- 总结与讨论

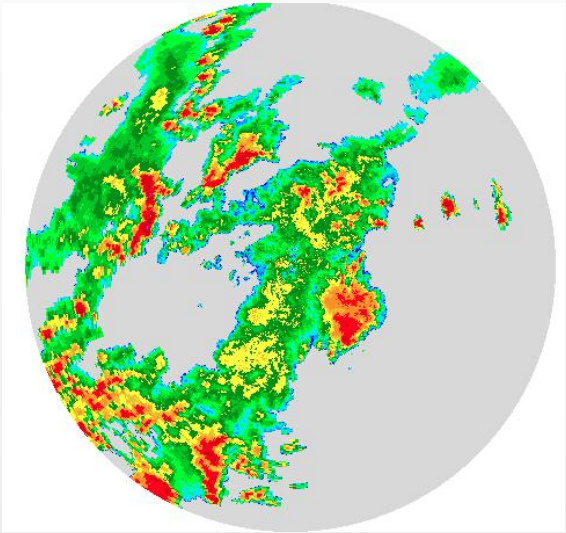
图表示学习

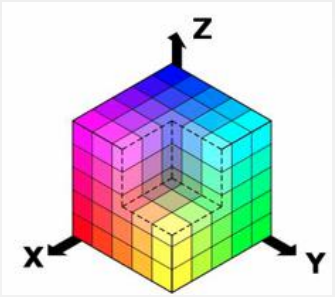
- 图表示学习 == 网络嵌入 == 网络表示学习 == 图嵌入
- 将节点用低维、稠密、实值向量来表示。
- 配合完成 graph 上的下游任务，节点分类、链路预测、社区发现等。
- 嵌入前后的不变性，eg. 节点的相邻关系不会变。
- Image，原生就是低维、稠密、实值向量。
- 算法类型
  - 通用嵌入，eg. deepwalk
  - end-to-end，针对下游任务优化的嵌入，eg. dygraph2vec

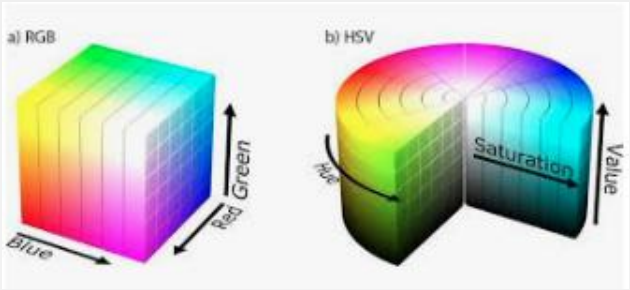


图表示学习

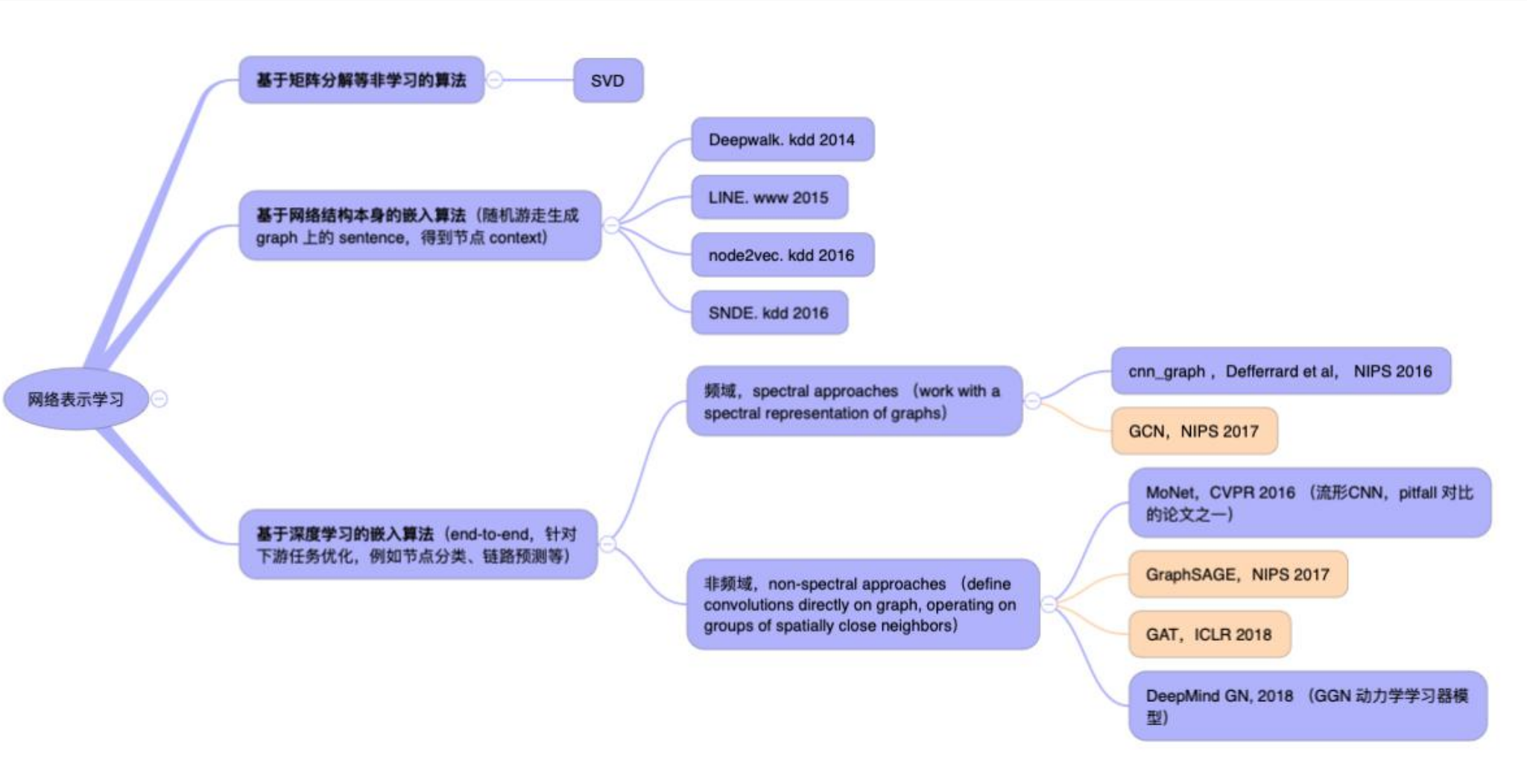

基本反射率
雷达站名:北 京
观测日期:2013-08-11
观测时间:17:48:00BJT
数据范围:230 km
dBZ
70
65
60
55
50
45
40
35
30
25
20
15
10
5
0
coloar map




Z
X
Y

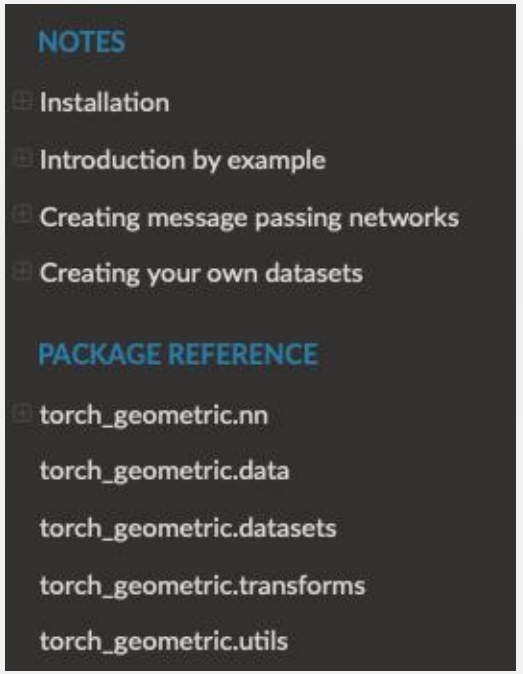

a) RGB
Blue
Red
Green
b) HSV
Saturation
Value

图神经网络的发展


网络表示学习
基于矩阵分解等非学习的算法
SVD
基于网络结构本身的嵌入算法（随机游走生成graph 上的 sentence，得到节点 context）
Deepwalk. kdd 2014
LINE. www 2015
node2vec. kdd 2016
SNDE. kdd 2016
基于深度学习的嵌入算法（end-to-end，针对下游任务优化，例如节点分类、链路预测等）
频域，spectral approaches （work with a spectral representation of graphs）
cnn_graph，Defferrard et al， NIPS 2016
GCN，NIPS 2017
非频域，non-spectral approaches （define convolutions directly on graph, operating on groups of spatially close neighbors）
MoNet，CVPR 2016 （流形CNN，pitfall 对比的论文之一）
GraphSAGE，NIPS 2017
GAT，ICLR 2018
DeepMind GN, 2018 （GGN 动力学学习器模型）

PyG 图神经网络开发框架

- PyTorch Geometric (PyG) is a geometric deep learning extension library for PyTorch.
- 其他图神经网络开发框架 or benchmark
  - 清华刘知远组，https://github.com/thunlp/OpenNE
  - 南加大，https://github.com/palash1992/GEM
  - Deep Graph Library (DGL) https://github.com/dmlc/dgl
  - Pitfalls of Graph Neural Network Evaluation，https://github.com/shchur/gnn-benchmark
- PyG 特点
  - 设计理念和 PyTorch 一脉相承
  - Pythonic
  - 可读性好，使用简单
  - 官方已经实现了大量的模型可以参考。
  - 动态图，调试方便。



Neighborhood Aggregation & MessagePassing

Neighborhood Aggregation. Generalizing the convolutional operator to irregular domains is typically expressed as a *neighborhood aggregation* or *message passing* scheme (Gilmer et al., 2017)

$$\vec{x}_i^{(k)} = \gamma^{(k)}\left(\vec{x}_i^{(k-1)}, \square_{j \in \mathcal{N}(i)} \phi^{(k)}\left(\vec{x}_i^{(k-1)}, \vec{x}_j^{(k-1)}, \vec{e}_{i,j}\right)\right) \quad (1)$$

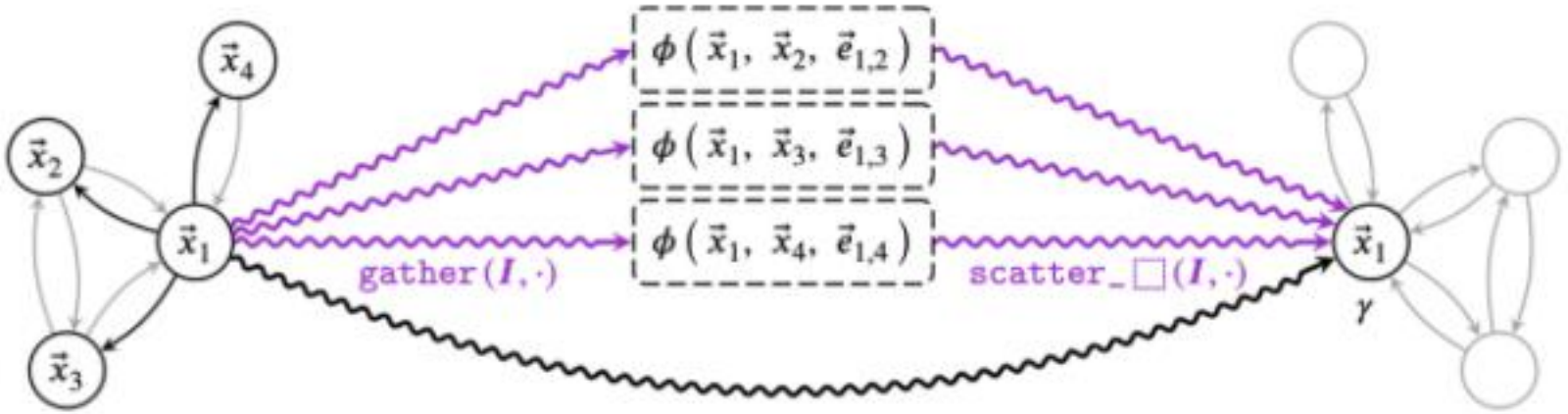


Figure 1: Computation scheme of a GNN layer by leveraging gather and scatter methods based on edge indices I, hence alternating between node parallel space and edge parallel space.

PyG 安装

- 推荐基于 pyenv 安装
- 先安装 python 3.6.3 及以上版本
- 创建虚拟环境
- 安装 PyTorch 1.0
- 安装 PyG
- Mac 端安装命令如下，可以解决 xcode gcc 编译器问题

```
MACOSX_DEPLOYMENT_TARGET=10.9 CC=clang CXX=clang++ pip install -i
https://pypi.tuna.tsinghua.edu.cn/simple torch-scatter torch-sparse torch-cluster torch-
spline-conv torch-geometric
```

代码实现

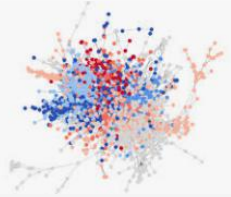
PyTorch/PyG Code Frame
* model
  * model is a combination of
    * conv (GCNconv/Conv2d.....)
    * Linear
    * relu / elu
    * pooling
    * dropout
  * endwith
    * log_softmax (for classification problems.)
* dataset & dataloader
  * batch_size, work number, path, etc.
  * normalize (mean=0, std=1 or min=-1, max=1)
* train
  * output = model(input)
  * loss.backward()
  * optimizer.step()
* test
  * output = model(input)
  * without loss.backward()
  * show accuracy and metric.
* main
  * define optimizer (lr, weight_decay...)
  * use gpu
  * iterate over each epoch

pytorch/example/mnist.py


```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 20, 5, 1)
        self.conv2 = nn.Conv2d(20, 50, 5, 1)
        self.fc1 = nn.Linear(4*4*50, 500)
        self.fc2 = nn.Linear(500, 10)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = F.max_pool2d(x, 2, 2)
        x = F.relu(self.conv2(x))
        x = F.max_pool2d(x, 2, 2)
        x = x.view(-1, 4*4*50)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return F.log_softmax(x, dim=1)
```

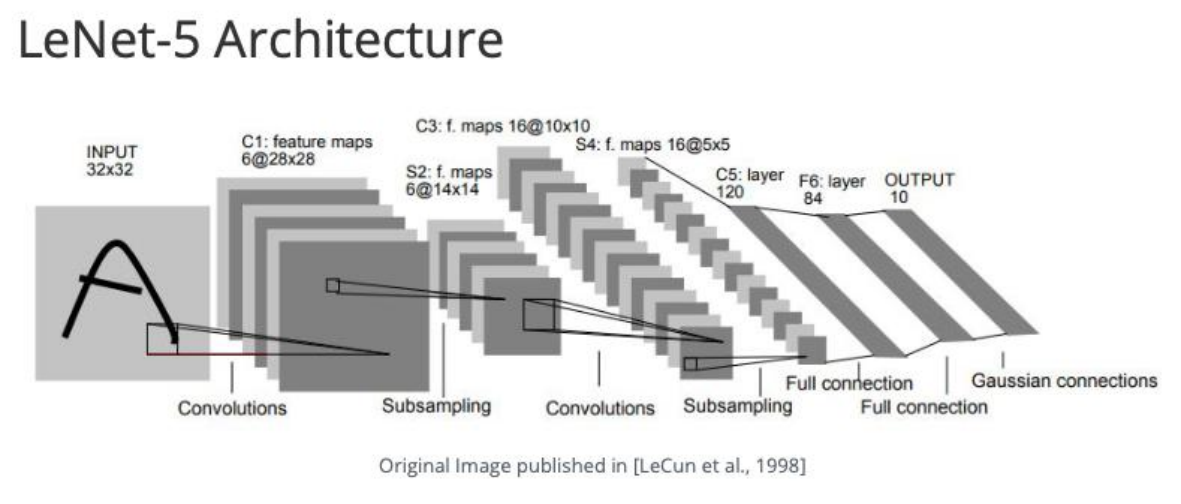
pyg/example/cora.py

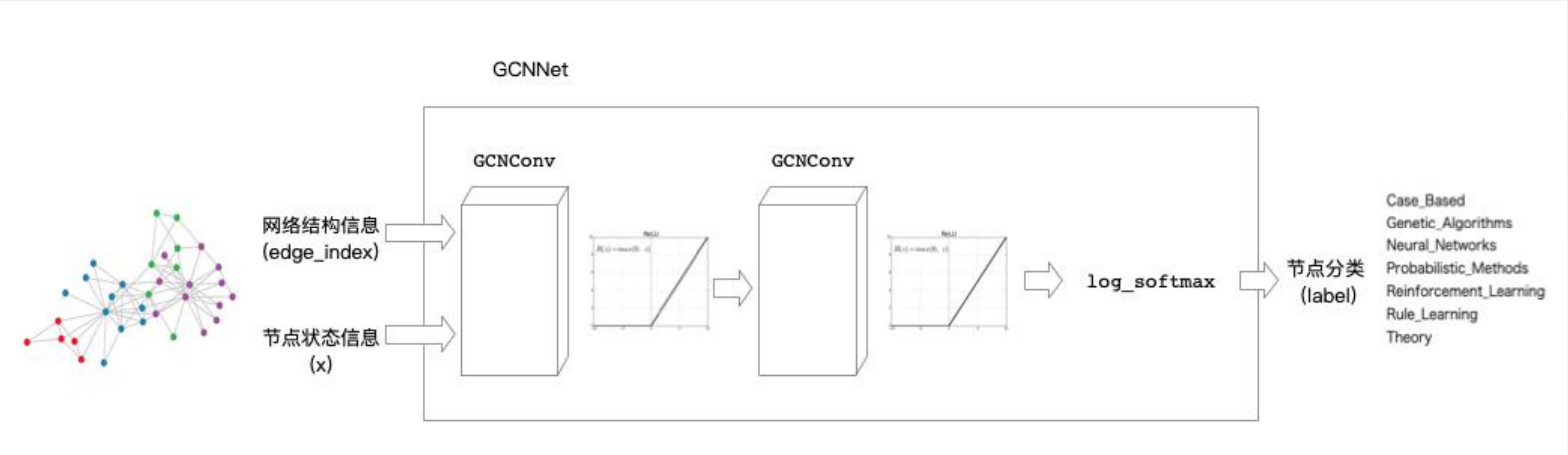


```
class Net(torch.nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = GCNConv(dataset.num_features, 16, cached=True)
        self.conv2 = GCNConv(16, dataset.num_classes, cached=True)
        # self.conv1 = ChebConv(data.num_features, 16, K=2)
        # self.conv2 = ChebConv(16, data.num_features, K=2)

    def forward(self):
        x, edge_index = data.x, data.edge_index
        x = F.relu(self.conv1(x, edge_index))
        x = F.dropout(x, training=self.training)
        x = self.conv2(x, edge_index)
        return F.log_softmax(x, dim=1)
```

代码实现--Model





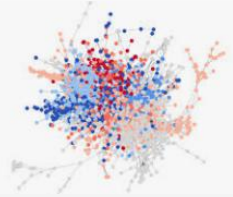
代码实现

PyTorch/PyG Code Frame
* model
  * model is a combination of
    * conv (GCNconv/Conv2d.....)
    * Linear
    * relu / elu
    * pooling
    * dropout
  * endwith
    * log_softmax (for classification problems.)
* dataset & dataloader
  * batch_size, work number, path, etc.
  * normalize (mean=0, std=1 or min=-1, max=1)
* train
  * output = model(input)
  * loss.backward()
  * optimizer.step()
* test
  * output = model(input)
  * without loss.backward()
  * show accuracy and metric.
* main
  * define optimizer (lr, weight_decay...)
  * use gpu
  * iterate over each epoch

pytorch/example/mnist.py


```
kwargs = {'num_workers': 1, 'pin_memory': True} if use_cuda else {}
train_loader = torch.utils.data.DataLoader(
    datasets.MNIST('../data', train=True, download=True,
                   transform=transforms.Compose([
                       transforms.ToTensor(),
                       transforms.Normalize((0.1307,), (0.3081,))
                   ])),
    batch_size=args.batch_size, shuffle=True, **kwargs)
test_loader = torch.utils.data.DataLoader(
    datasets.MNIST('../data', train=False, transform=transforms.Compose([
                       transforms.ToTensor(),
                       transforms.Normalize((0.1307,), (0.3081,))
                   ])),
    batch_size=args.test_batch_size, shuffle=True, **kwargs)
```

pyg/example/cora.py



```
dataset = 'Cora'
path = osp.join(osp.dirname(osp.realpath(__file__)), '..', 'data', dataset)
dataset = Planetoid(path, dataset, T.NormalizeFeatures())
data = dataset[0]
```

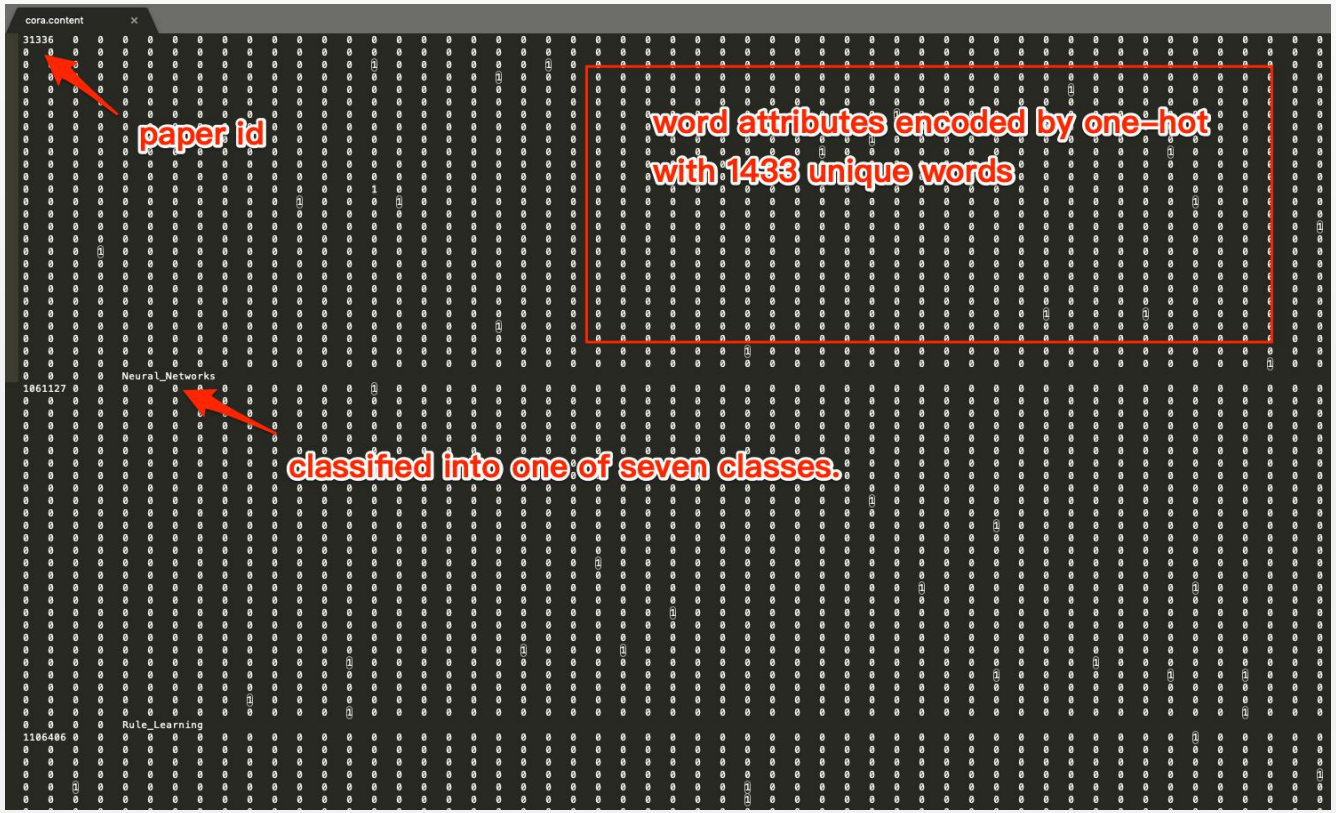
Cora 数据

Cora

-https://github.com/tkipf/pygcn/tree/master/data/cora

- cora.content: <paper_id> <word_attributes>+ <class_label>

- cora.cites: <ID of cited paper> <ID of citing paper>
  - If a line is represented by "paper1 paper2" then the link is "paper2->paper1".
  - class_label
    * Case_Based
    * Genetic_Algorithms
    * Neural_Networks
    * Probabilistic_Methods
    * Reinforcement_Learning
    * Rule_Learning
    * Theory



Cora 数据

- 节点属性：1433 维度的，one-hot 编码的向量（过一个 linear 层，或者直接上 GCNConv 等。变成，低维，稠密，实值的表示。）

- PyG 可以有两个输入，节点状态信息，和网络结构信息。分别对应cora.content， cora.cites。

- https://github.com/kimiyoung/planetoid/tree/master/data

trans.cora.graph

trans.cora.tx

trans.cora.ty

trans.cora.x

trans.cora.y

```
981: [1506, 1511, 317],
982: [1332, 2148],
983: [1708, 1524, 2022, 1519],
984: [1236, 912, 1285],
985: [1421, 1752, 1458, 1544, 1545, 2446, 2447, 176],
986: [1974],
987: [922],
988: [1760, 366, 628],
989: [1624, 911, 1330, 869, 1861, 869],
990: [1005, 1460],
991: [902, 733, 2044, 2045],
992: [1140, 453, 792],
993: [826, 826],
994: [832, 1435, 1457, 1414, 1403, 223, 1515, 877],
995: [695, 1119, 2630, 1099, 1948, 898],
996: [182, 861, 91, 1894, 656, 198],
997: [1140, 2349, 771],
998: [2273, 1111],
999: [2059, 908, 2206, 1269],
```

Cora 数据


```
In [22]: data.test_mask
Out[22]: tensor([0, 0, 0,  ..., 1, 1, 1], dtype=torch.uint8)

In [24]: data.train_mask
Out[24]: tensor([1, 1, 1,  ..., 0, 0, 0], dtype=torch.uint8)

In [25]: data.val_mask
Out[25]: tensor([0, 0, 0,  ..., 0, 0, 0], dtype=torch.uint8)

In [14]: data
Out[14]: Data(edge_index=[2, 10556], test_mask=[2708], train_mask=[2708], val_mask=[2708], x=[2
708, 1433], y=[2708])

In [26]: data.y
Out[26]: tensor([3, 4, 4,  ..., 3, 3, 3])
```

代码实现

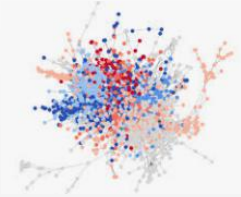
PyTorch/PyG Code Frame
* model
  * model is a combination of
    * conv (GCNconv/Conv2d.....)
    * Linear
    * relu / elu
    * pooling
    * dropout
  * endwith
    * log_softmax (for classification problems.)
* dataset & dataloader
  * batch_size, work number, path, etc.
  * normalize (mean=0, std=1 or min=-1, max=1)
* train
  * output = model(input)
  * loss.backward()
  * optimizer.step()
* test
  * output = model(input)
  * without loss.backward()
  * show accuracy and metric.
* main
  * define optimizer (lr, weight_decay...)
  * use gpu
  * iterate over each epoch

pytorch/example/mnist.py


```python
def train(args, model, device, train_loader, optimizer, epoch):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = data.to(device), target.to(device)
        optimizer.zero_grad()
        output = model(data)
        loss = F.nll_loss(output, target)
        loss.backward()
        optimizer.step()
        if batch_idx % args.log_interval == 0:
            print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
                epoch, batch_idx * len(data), len(train_loader.dataset),
                100. * batch_idx / len(train_loader), loss.item()))

def test(args, model, device, test_loader):
    model.eval()
    test_loss = 0
    correct = 0
    with torch.no_grad():
        for data, target in test_loader:
            data, target = data.to(device), target.to(device)
            output = model(data)
            test_loss += F.nll_loss(output, target, reduction='sum').item() # sum up batch loss
            pred = output.argmax(dim=1, keepdim=True) # get the index of the max log-probability
            correct += pred.eq(target.view_as(pred)).sum().item()
```

pyg/example/cora.py



```python
def train():
    model.train()
    optimizer.zero_grad()
    F.nll_loss(model()[data.train_mask], data.y[data.train_mask]).backward()
    optimizer.step()


def test():
    model.eval()
    logits, accs = model(), []
    for _, mask in data('train_mask', 'test_mask'):
        pred = logits[mask].max(1)[1]
        acc = pred.eq(data.y[mask]).sum().item() / mask.sum().item()
        accs.append(acc)
    return accs
```

注意：

transductive learning，训练的时候，train_data 和 test_data 都输入到网络中（data 包含了 train_data 和 test_data），但是只有 train_data 对网络进行训练，所以，要用到 data.train_mask

代码实现

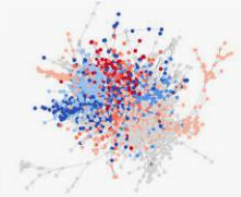
PyTorch/PyG Code Frame
* model
  * model is a combination of
    * conv (GCNconv/Conv2d.....)
    * Linear
    * relu / elu
    * pooling
    * dropout
  * endwith
    * log_softmax (for classification problems.)
* dataset & dataloader
  * batch_size, work number, path, etc.
  * normalize (mean=0, std=1 or min=-1, max=1
* train
  * output = model(input)
  * loss.backward()
  * optimizer.step()
* test
  * output = model(input)
  * without loss.backward()
  * show accuracy and metric.
* main
  * define optimizer (lr, weight_decay...)
  * use gpu
  * iterate over each epoch

pytorch/example/mnist.py


```
model = Net().to(device)
optimizer = optim.SGD(model.parameters(), lr=args.lr, momentum=args.momentum)

for epoch in range(1, args.epochs + 1):
    train(args, model, device, train_loader, optimizer, epoch)
    test(args, model, device, test_loader)
```

pyg/example/cora.py

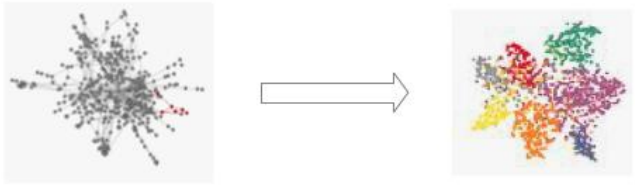


```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model, data = Net().to(device), data.to(device)
optimizer = torch.optim.Adam(model.parameters(), lr=0.01, weight_decay=5e-3)
```

```
for epoch in range(1, 201):
    train()
    log = 'Epoch: {:03d}, Train: {:.4f}, Test: {:.4f}'
    print(log.format(epoch, *test()))
```

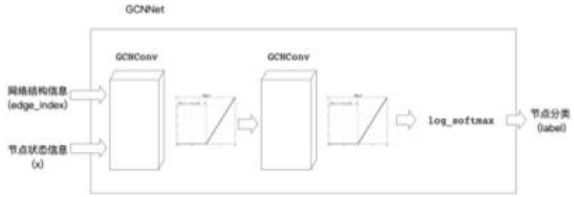
图神经网络框架及PyG文档

要完成的任务，
eg.节点分类



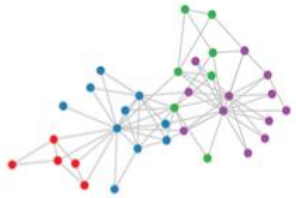
pytorch_geometric/examples/cora.py，节点分类

神经网络结构



eg. pytorch_geometric/examples/cora.py 中的模型

Graph 数据存储及
dataloader

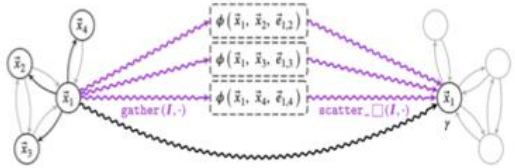


pytorch_geometric.data
pytorch_geometric.datasets

utils 和 transforms 提供常用函数
eg. 计算节点 degree 等

---

graph 的最小单元：
以节点 i 为中心的邻域，
及基于边的信息交互



调用 pytorch_geometric.nn 中提供的组件
or
基于 MessagePassing 创建新的 Conv 模块

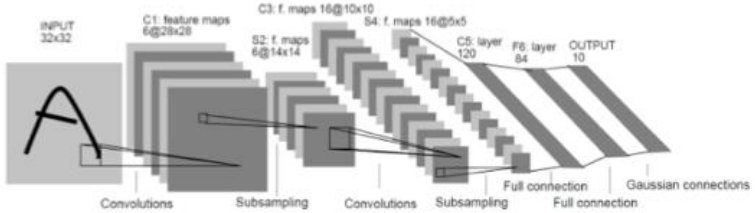
卷积神经网络及PyTorch文档

要完成的任务，
eg. 手写体识别图片分类

5 0 4 1

pytorch/examples/mnist/main.py

神经网络结构

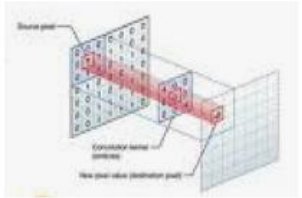


pytorch/examples/mnist/main.py 中的模型

Image 数据存储及
dataloader

torch.utils.data.Dataloader

Image 的最小单元：
以格点 i 为中心的邻域，
及周边格点的信息交互



nn.Conv2D

Neighborhood Aggregation & MessagePassing

Neighborhood Aggregation. Generalizing the convolutional operator to irregular domains is typically expressed as a *neighborhood aggregation* or *message passing* scheme (Gilmer et al., 2017)

$$\vec{x}_i^{(k)} = \gamma^{(k)}\left(\vec{x}_i^{(k-1)}, \square_{j \in \mathcal{N}(i)} \phi^{(k)}\left(\vec{x}_i^{(k-1)}, \vec{x}_j^{(k-1)}, \vec{e}_{i,j}\right)\right) \quad (1)$$

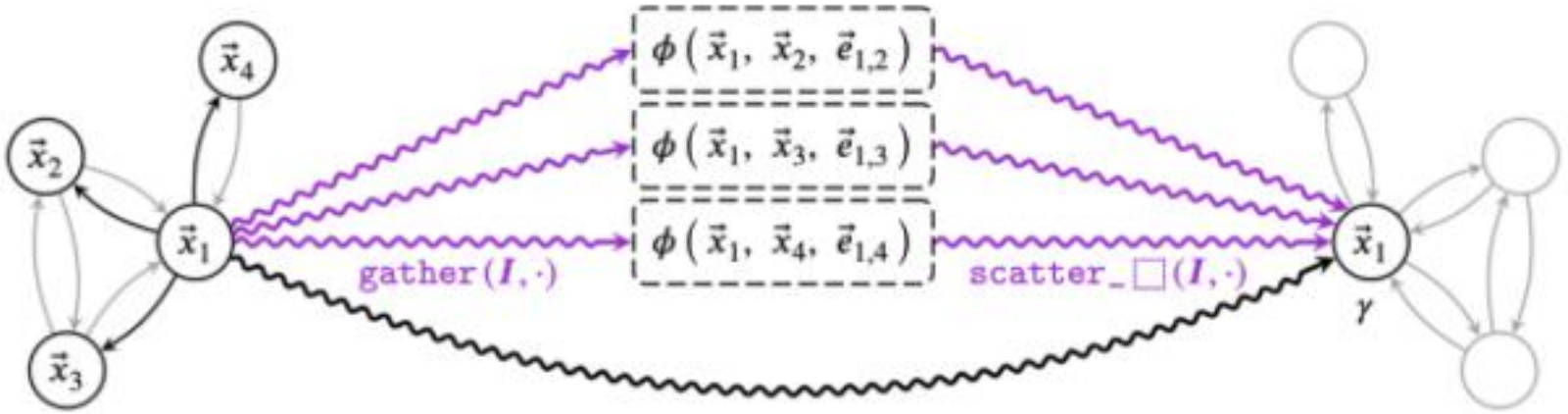


Figure 1: Computation scheme of a GNN layer by leveraging gather and scatter methods based on edge indices I, hence alternating between node parallel space and edge parallel space.

代码实现--GCNConv

```
from torch_geometric.nn import GCNConv
```

OR

```
class GCNConv(MessagePassing):
    def __init__(self, in_channels, out_channels):
        super(GCNConv, self).__init__(aggr='add') # "Add" aggregation.
        self.lin = nn.Linear(in_channels, out_channels)

    def forward(self, x, edge_index):
        # x has shape [N, in_channels]
        # edge_index has shape [2, E]

        # Step 1: Add self-loops to the adjacency matrix.
        edge_index = add_self_loops(edge_index, num_nodes=x.size(0))

        # Step 2: Linearly transform node feature matrix.
        x = self.lin(x)

        # Step 3-5: Start propagating messages.
        return self.propagate('add', edge_index, size=(x.size(0), x.size(0)), x=x)

    def message(self, x_j, edge_index, size):
        # x_j has shape [E, out_channels]
        # Step 3: Normalize node features.
        row, col = edge_index
        deg = degree(row, size[0], dtype=x_j.dtype)
        deg_inv_sqrt = deg.pow(-0.5)
        norm = deg_inv_sqrt[row] * deg_inv_sqrt[col]

        return norm.view(-1, 1) * x_j

    def update(self, aggr_out):
        # aggr_out has shape [N, out_channels]
        # Step 5: Return new node embeddings.
        return aggr_out
```

https://rusty1s.github.io/pytorch_geometric/build/html/notes/create_gnn.html

代码实现--GCNConv

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

OR

```python
class GCNConv(MessagePassing):
    def __init__(self, in_channels, out_channels):
        super(GCNConv, self).__init__(aggr='add') # "Add" aggregation.
        self.lin = nn.Linear(in_channels, out_channels)

    def forward(self, x, edge_index):
        # x has shape [N, in_channels]
        # edge_index has shape [2, E]

        # Step 1: Add self-loops to the adjacency matrix.
        edge_index = add_self_loops(edge_index, num_nodes=x.size(0))

        # Step 2: Linearly transform node feature matrix.
        x = self.lin(x)

        # Step 3-5: Start propagating messages.
        return self.propagate('add', edge_index, size=(x.size(0), x.size(0)), x=x)

    def message(self, x_j, edge_index, size):
        # x_j has shape [E, out_channels]

        # Step 3: Normalize node features.
        row, col = edge_index
        deg = degree(row, size[0], dtype=x_j.dtype)
        deg_inv_sqrt = deg.pow(-0.5)
        norm = deg_inv_sqrt[row] * deg_inv_sqrt[col]

        return norm.view(-1, 1) * x_j

    def update(self, aggr_out):
        # aggr_out has shape [N, out_channels]

        # Step 5: Return new node embeddings.
        return aggr_out
```

这里实现论文中的公式

https://rusty1s.github.io/pytorch_geometric/build/html/notes/create_gnn.html

代码实现--GCNConv

The GCN layer is mathematically defined as

$$\mathbf{x}_i^{(k)} = \sum_{j \in \mathcal{N}(i) \cup \{i\}} \frac{1}{\sqrt{\deg(i)} \cdot \sqrt{deg(j)}} \cdot \left(\mathbf{\Theta} \cdot \mathbf{x}_j^{(k-1)} \right),$$

where neighboring node features are first transformed by a weight matrix $\mathbf{\Theta}$, normalized by their degree, and finally summed up. This formula can be divided into the following steps:

1. Add self-loops to the adjacency matrix.
2. Linearly transform node feature matrix.
3. Normalize node features in ϕ.
4. Sum up neighboring node features (`"add"` aggregation).
5. Return new node embeddings in γ.

```python
class GCNConv(MessagePassing):
    def __init__(self, in_channels, out_channels):
        super(GCNConv, self).__init__(aggr='add') # "Add" aggregation.
        self.lin = nn.Linear(in_channels, out_channels)

    def forward(self, x, edge_index):
        # x has shape [N, in_channels]
        # edge_index has shape [2, E]

        # Step 1: Add self-loops to the adjacency matrix.
        edge_index = add_self_loops(edge_index, num_nodes=x.size(0))

        # Step 2: Linearly transform node feature matrix.
        x = self.lin(x)

        # Step 3-5: Start propagating messages.
        return self.propagate('add', edge_index, size=(x.size(0), x.size(0)), x=x)

    def message(self, x_j, edge_index, size):
        # x_j has shape [E, out_channels]
        # Step 3: Normalize node features.
        row, col = edge_index
        deg = degree(row, size[0], dtype=x_j.dtype)
        deg_inv_sqrt = deg.pow(-0.5)
        norm = deg_inv_sqrt[row] * deg_inv_sqrt[col]

        return norm.view(-1, 1) * x_j

    def update(self, aggr_out):
        # aggr_out has shape [N, out_channels]
        # Step 5: Return new node embeddings.
        return aggr_out
```

https://rusty1s.github.io/pytorch_geometric/build/html/notes/create_gnn.html

代码实现--GCNConv

调试技巧，print+exit 大法

```
print(edge_index.shape)
edge_index = add_self_loops(edge_index, num_nodes=x.size(0))
print(edge_index.shape)
exit()
```

```
(py3k) Shawn-Wang-MacBook-Pro:graph-neural-network-pyg shawnwang$ python cora.py
torch.Size([2, 10556])
torch.Size([2, 13264])
```

```
In [1]: 13264 - 10556
Out[1]: 2708
```

```
class GCNNet(nn.Module):
    def __init__(self, dataset):
        super(GCNNet, self).__init__()
        self.conv1 = GCNConv(dataset.num_features, 16)
        self.conv2 = GCNConv(16, dataset.num_classes)
```

```
(py3k) Shawn-Wang-MacBook-Pro:graph-neural-network-pyg shawnwang$ python cora.py
torch.Size([2708, 1433])
torch.Size([2708, 16])
```

```
# Step 2: Linearly transform node feature matrix.
print(x.shape)
x = self.lin(x)
print(x.shape)
exit()
```

```
class GCNConv(MessagePassing):
    def __init__(self, in_channels, out_channels):
        super(GCNConv, self).__init__(aggr='add') # "Add" aggregation.
        self.lin = nn.Linear(in_channels, out_channels)

    def forward(self, x, edge_index):
        # x has shape [N, in_channels]
        # edge_index has shape [2, E]

        # Step 1: Add self-loops to the adjacency matrix.
        edge_index = add_self_loops(edge_index, num_nodes=x.size(0))

        # Step 2: Linearly transform node feature matrix.
        x = self.lin(x)

        # Step 3-5: Start propagating messages.
        return self.propagate('add', edge_index, size=(x.size(0), x.size(0)), x=x)

    def message(self, x_j, edge_index, size):
        # x_j has shape [E, out_channels]
        # Step 3: Normalize node features.
        row, col = edge_index
        deg = degree(row, size[0], dtype=x_j.dtype)
        deg_inv_sqrt = deg.pow(-0.5)
        norm = deg_inv_sqrt[row] * deg_inv_sqrt[col]

        return norm.view(-1, 1) * x_j

    def update(self, aggr_out):
        # aggr_out has shape [N, out_channels]
        # Step 5: Return new node embeddings.
        return aggr_out
```

实验

```
Epoch: 084, Train: 0.9857, Test: 0.8040
Epoch: 085, Train: 0.9857, Test: 0.7980
Epoch: 086, Train: 0.9857, Test: 0.7960
Epoch: 087, Train: 0.9857, Test: 0.7960
Epoch: 088, Train: 0.9857, Test: 0.8030
Epoch: 089, Train: 0.9857, Test: 0.8100
Epoch: 090, Train: 0.9857, Test: 0.8110
Epoch: 091, Train: 0.9857, Test: 0.8130
Epoch: 092, Train: 0.9857, Test: 0.8130
Epoch: 093, Train: 0.9857, Test: 0.8140
Epoch: 094, Train: 0.9857, Test: 0.8110
Epoch: 095, Train: 0.9857, Test: 0.8090
Epoch: 096, Train: 0.9857, Test: 0.8070
Epoch: 097, Train: 0.9857, Test: 0.8050
Epoch: 098, Train: 0.9857, Test: 0.8050
Epoch: 099, Train: 0.9857, Test: 0.8050
```

GraphSAGE

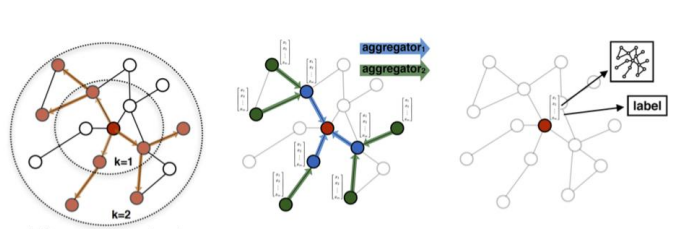
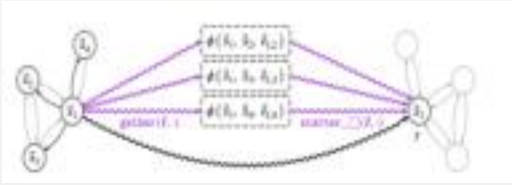


Figure 1: Visual illustration of the GraphSAGE sample and aggregate approach.



Algorithm 1: GraphSAGE embedding generation (i.e., forward propagation) algorithm

Input : Graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$; input features $\{\mathbf{x}_v, \forall v \in \mathcal{V}\}$; depth K; weight matrices $\mathbf{W}^k, \forall k \in \{1, ..., K\}$; non-linearity σ; differentiable aggregator functions $\text{AGGREGATE}_k, \forall k \in \{1, ..., K\}$; neighborhood function $\mathcal{N} : v \rightarrow 2^{\mathcal{V}}$

Output : Vector representations $\mathbf{z}_v$ for all $v \in \mathcal{V}$

$\mathbf{h}_v^0 \leftarrow \mathbf{x}_v, \forall v \in \mathcal{V}$;
for $k = 1...K$ **do**
  for $v \in \mathcal{V}$ **do**
    $\mathbf{h}_{\mathcal{N}(v)}^k \leftarrow \text{AGGREGATE}_k(\{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\})$;
    $\mathbf{h}_v^k \leftarrow \sigma\left(\mathbf{W}^k \cdot \text{CONCAT}(\mathbf{h}_v^{k-1}, \mathbf{h}_{\mathcal{N}(v)}^k)\right)$
  end
  $\mathbf{h}_v^k \leftarrow \mathbf{h}_v^k / \|\mathbf{h}_v^k\|_2, \forall v \in \mathcal{V}$
end
$\mathbf{z}_v \leftarrow \mathbf{h}_v^K, \forall v \in \mathcal{V}$

```python
def forward(self, x, edge_index):
    """"""
    edge_index, _ = remove_self_loops(edge_index)
    edge_index = add_self_loops(edge_index, num_nodes=x.size(0))

    x = x.unsqueeze(-1) if x.dim() == 1 else x
    row, col = edge_index

    x = torch.matmul(x, self.weight)
    out = scatter_mean(x[col], row, dim=0, dim_size=x.size(0))

    if self.bias is not None:
        out = out + self.bias

    if self.normalize:
        out = F.normalize(out, p=2, dim=-1)

    return out
```

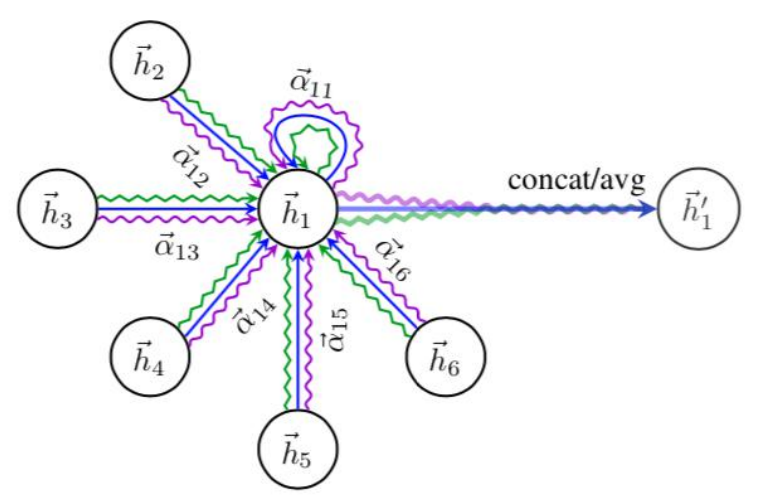
```
normalize (bool, optional): If set to :obj:`False`, output features
    will not be :math:`\ell_2`-normalized.
```

https://github.com/rusty1s/pytorch_geometric/blob/master/torch_geometric/nn/conv/sage_conv.py

GAT

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

$$\alpha_{ij} = \frac{\exp\left(\mathrm{LeakyReLU}\left(\vec{\mathbf{a}}^T[\mathbf{W}\vec{h}_i \| \mathbf{W}\vec{h}_j]\right)\right)}{\sum_{k\in\mathcal{N}_i}\exp\left(\mathrm{LeakyReLU}\left(\vec{\mathbf{a}}^T[\mathbf{W}\vec{h}_i \| \mathbf{W}\vec{h}_k]\right)\right)}$$




```python
def forward(self, x, edge_index):
    """"""
    edge_index, _ = remove_self_loops(edge_index)
    edge_index = add_self_loops(edge_index, num_nodes=x.size(0))

    x = torch.mm(x, self.weight).view(-1, self.heads, self.out_channels)
    return self.propagate(edge_index, x=x, num_nodes=x.size(0))

def message(self, x_i, x_j, edge_index, num_nodes):
    # Compute attention coefficients.
    alpha = (torch.cat([x_i, x_j], dim=-1) * self.att).sum(dim=-1)
    alpha = F.leaky_relu(alpha, self.negative_slope)
    alpha = softmax(alpha, edge_index[0], num_nodes)

    # Sample attention coefficients stochastically.
    if self.training and self.dropout > 0:
        alpha = F.dropout(alpha, p=self.dropout, training=True)

    return x_j * alpha.view(-1, self.heads, 1)
```

https://github.com/rusty1s/pytorch_geometric/blob/master/torch_geometric/nn/conv/gat_conv.py

讨论

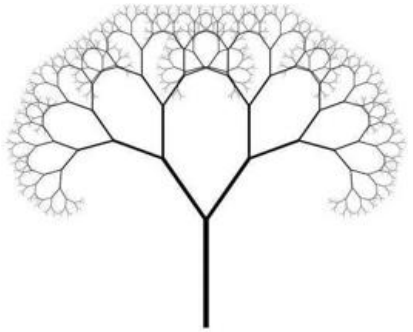
1. 把抽象概念和操作具象化，可视化。例如，画出网络结构，可视化数据的值。

2. 运用类比和对比。例如， mnist 和 cora ， graph 和 image，CNN 和 GCNConv，颜色坐标和图表示学习类比。

3. 调试代码，print+exit，保证写下的每一行是正确的，再往后写。

4. 清楚自己开发的代码是哪个层次的（可以可视化为分形树）。比如这里的 Model 和 GCNConv 的关系。某些部分当成黑箱使用。

5. Don't reinvent the wheel. PyG 提供了很多常用的函数在 utils 里面，比如计算 degree，统计 true_positive，f1_score 这些 metric。



未来方向

一套完善的Graph Network科研开发体系

- sacred（https://github.com/IDSIA/sacred），实验（超参数、复现、大量重复实验）管理
- tensorboard，可视化，debug

算法改进方向

- 更大的图，PyTorch-BigGraph
- 改进反传？Tian Qi Chen, 2018 NIPS Best paper, Neural Ordinary Differential Equations

文献（资源）列表

- 收集论文涉及到的重要资源
  - 代码，https://github.com/shawnwang-tech/graph-neural-network-pyg
  - Peter Battaglia. 2018. Relational inductive biases, deep learning, and graph networks
  - Michael Defferrard. 2017. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering
  - Petar Veličković. 2017. Graph Attention Networks
  - William Hamilton. 2017. Inductive Representation Learning on Large Graphs
  - Oleksandr Shchur. 2018. Pitfalls of Graph Neural Network Evaluation
  - Matthias Fey. 2019. Fast Graph Representation Learning with PyTorch Geometric
  - Palash Goyal. 2018. dyngraph2vec Capturing Network Dynamics using Dynamic Graph Representation Learning
  - Thomas Kipf. 2017. Semi-supervised Classification with Graph Convolutional Networks

联系我们

- 公众号：集智AI学园
- 公众号：集智俱乐部
- 知乎号：集智学园

集智AI学园公众号

集智俱乐部公众号