GNN with PyTorch Geometric

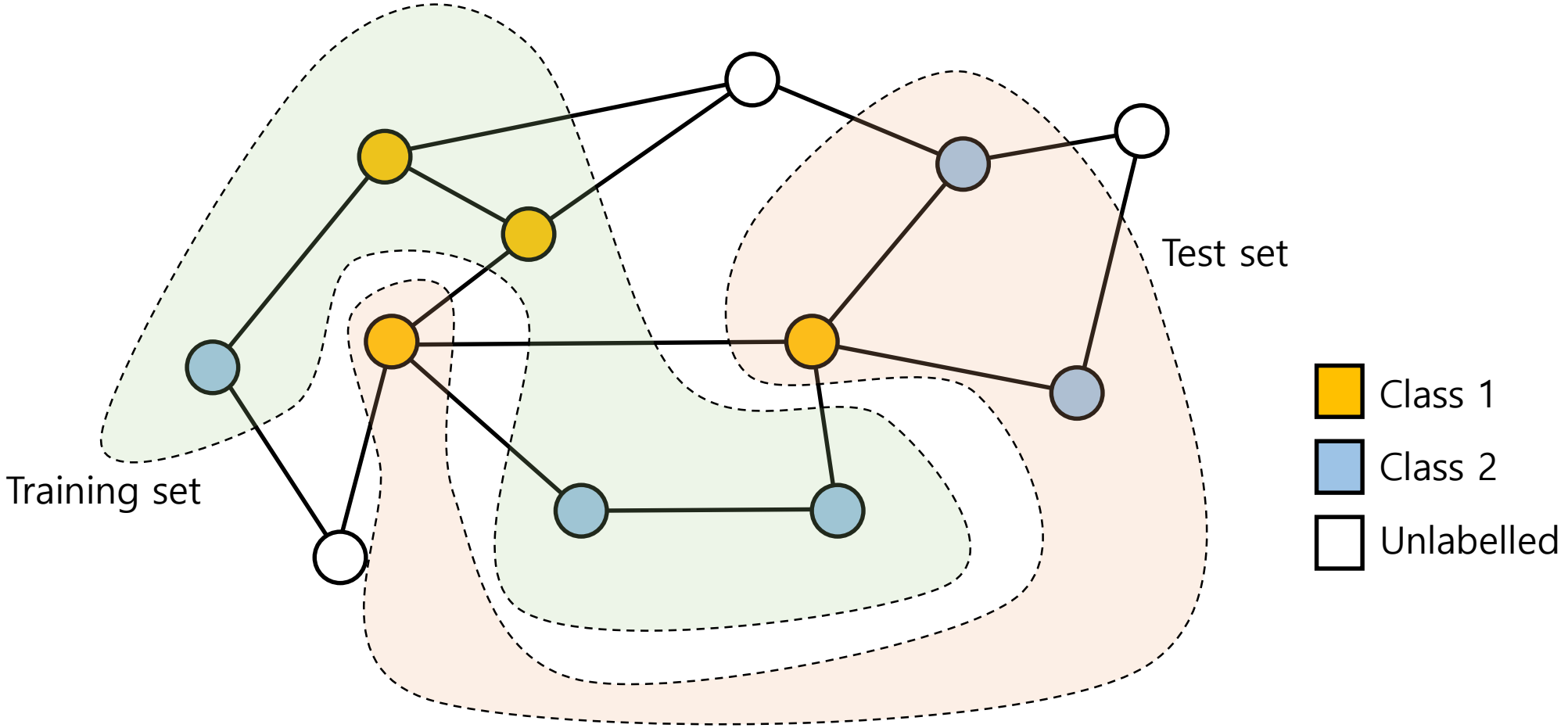
김성환

ALDE@PNU

2021.07.27

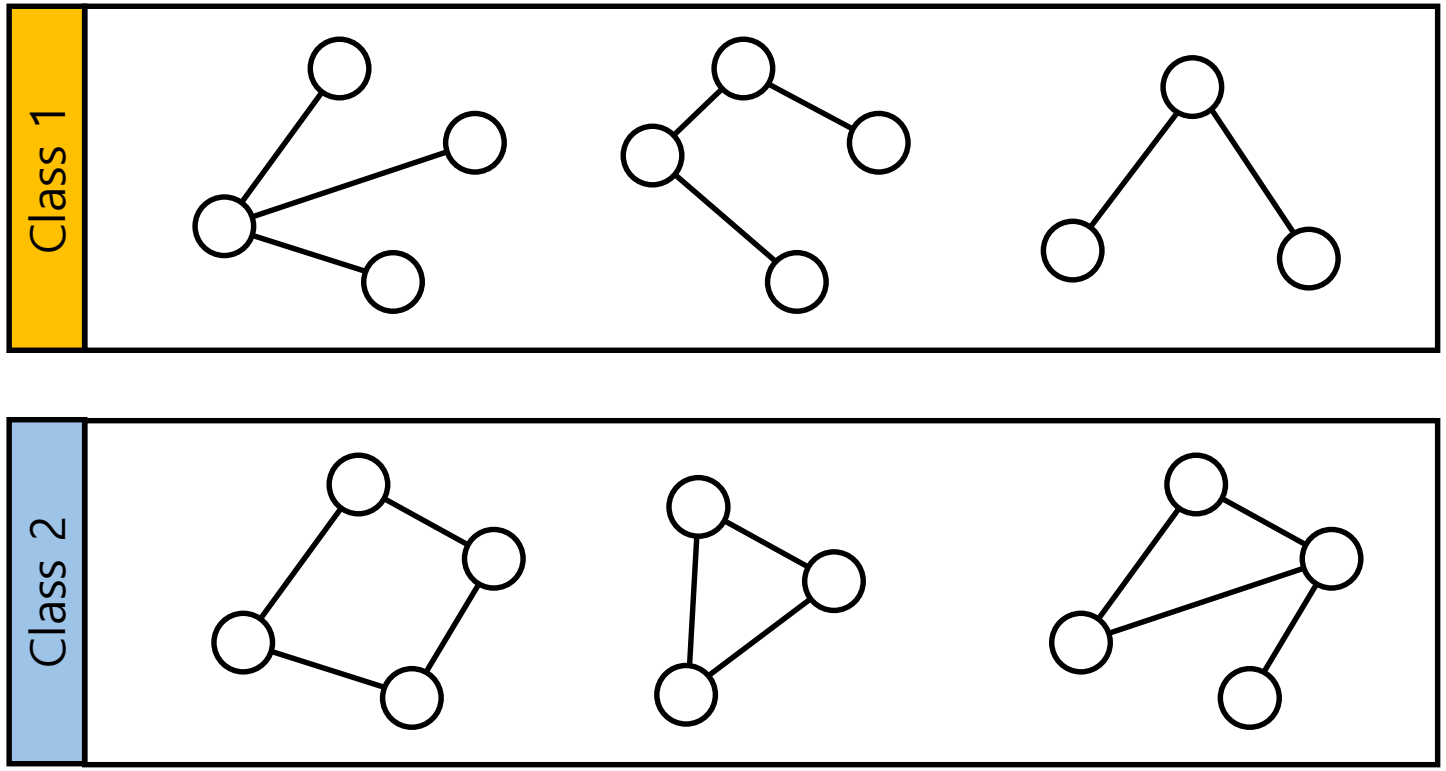
그래프 관련 문제 1: Node Classification

- 학습/입력: 노드의 일부분이 레이블링된 하나의 그래프
- 출력: 모든 노드의 레이블



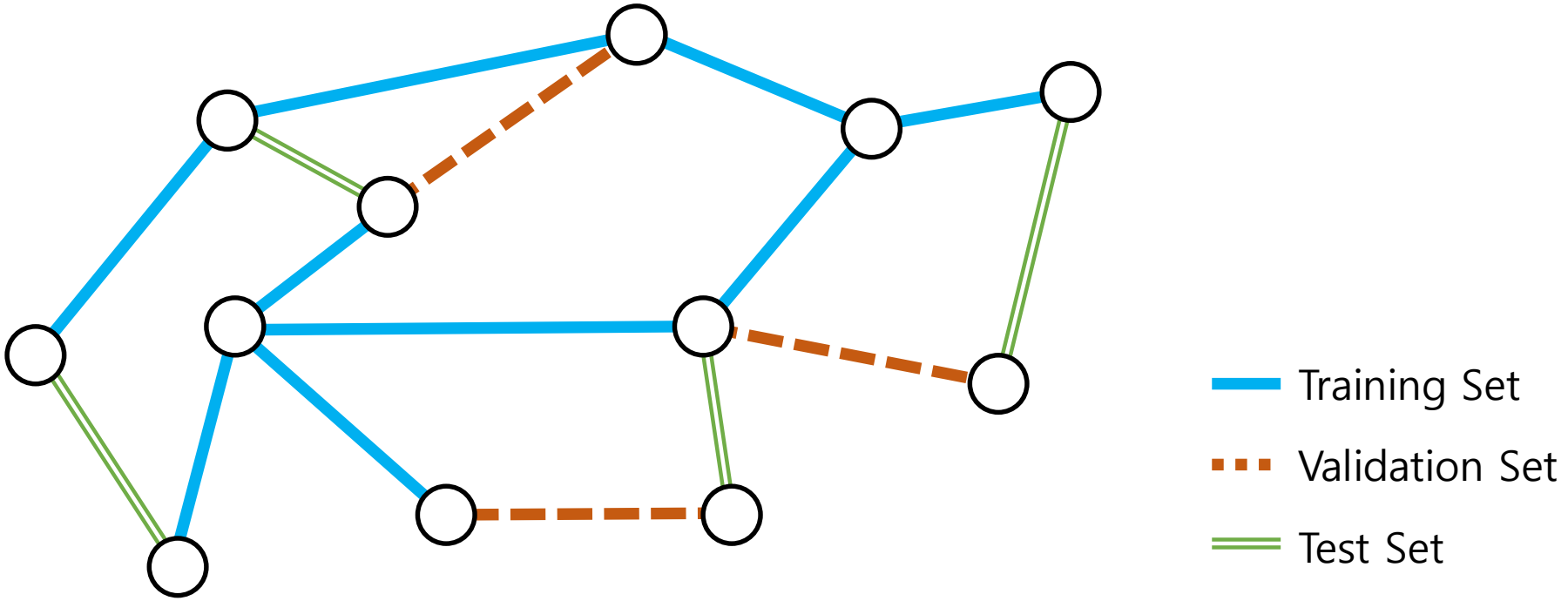
그래프 관련 문제 2: Graph Classification

- 학습: 레이블링된 그래프 집합
- 입력: 그래프
- 출력: 그래프의 레이블



그래프 관련 문제 3: Link Prediction

- 학습/입력: 그래프 (+노드 쌍 및 에지 존재 여부)
- 출력: 노드 쌍 간의 에지 존재 여부

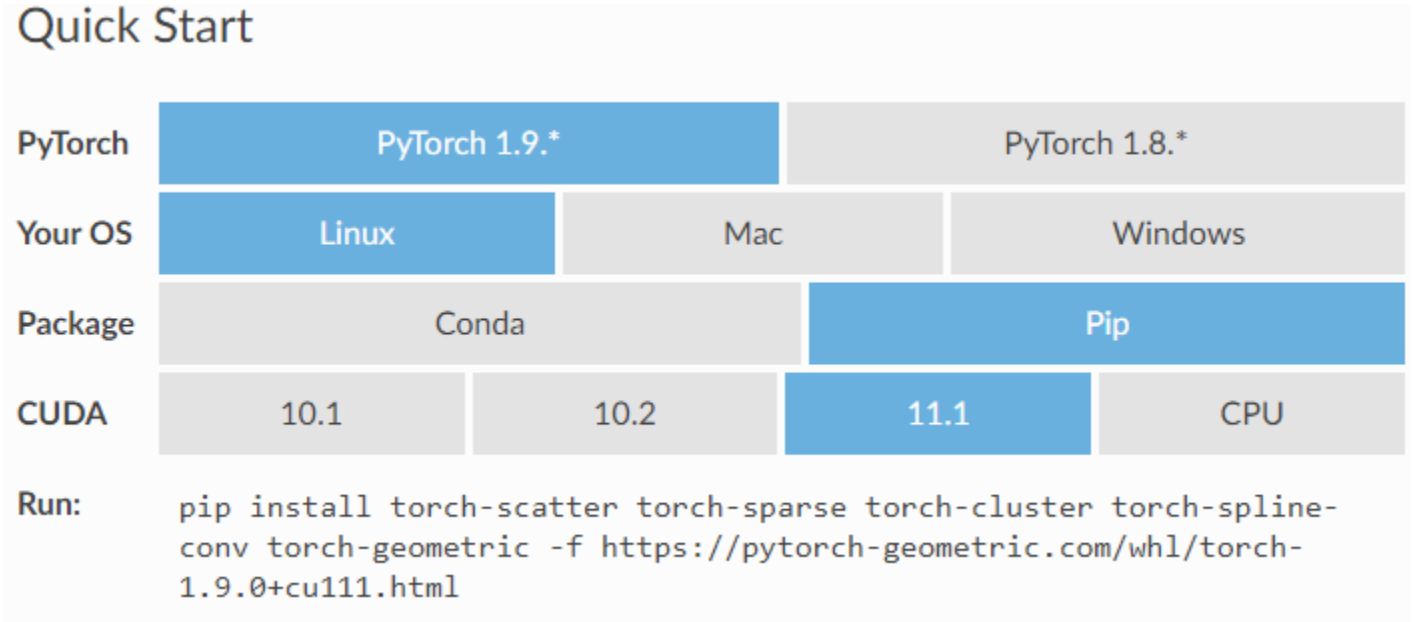


PyTorch Geometric 설치

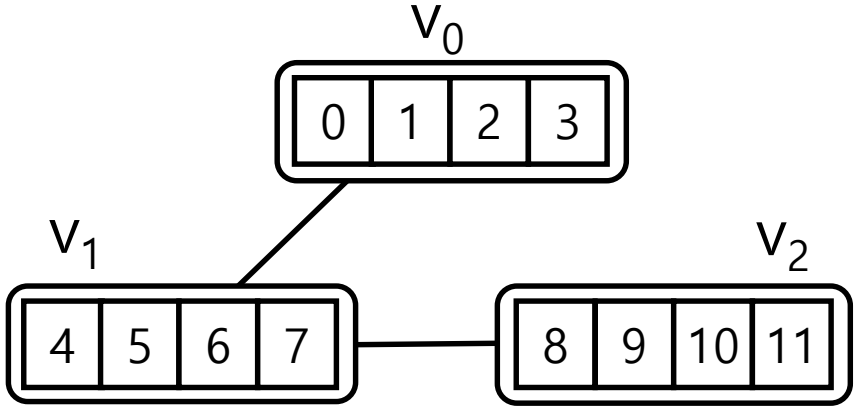
- https://pytorch-geometric.readthedocs.io/en/latest/notes/installation.html

- 아래와 같이 자신의 환경에 맞는 설치 명령어를 얻을 수 있음

예)



그래프 표현



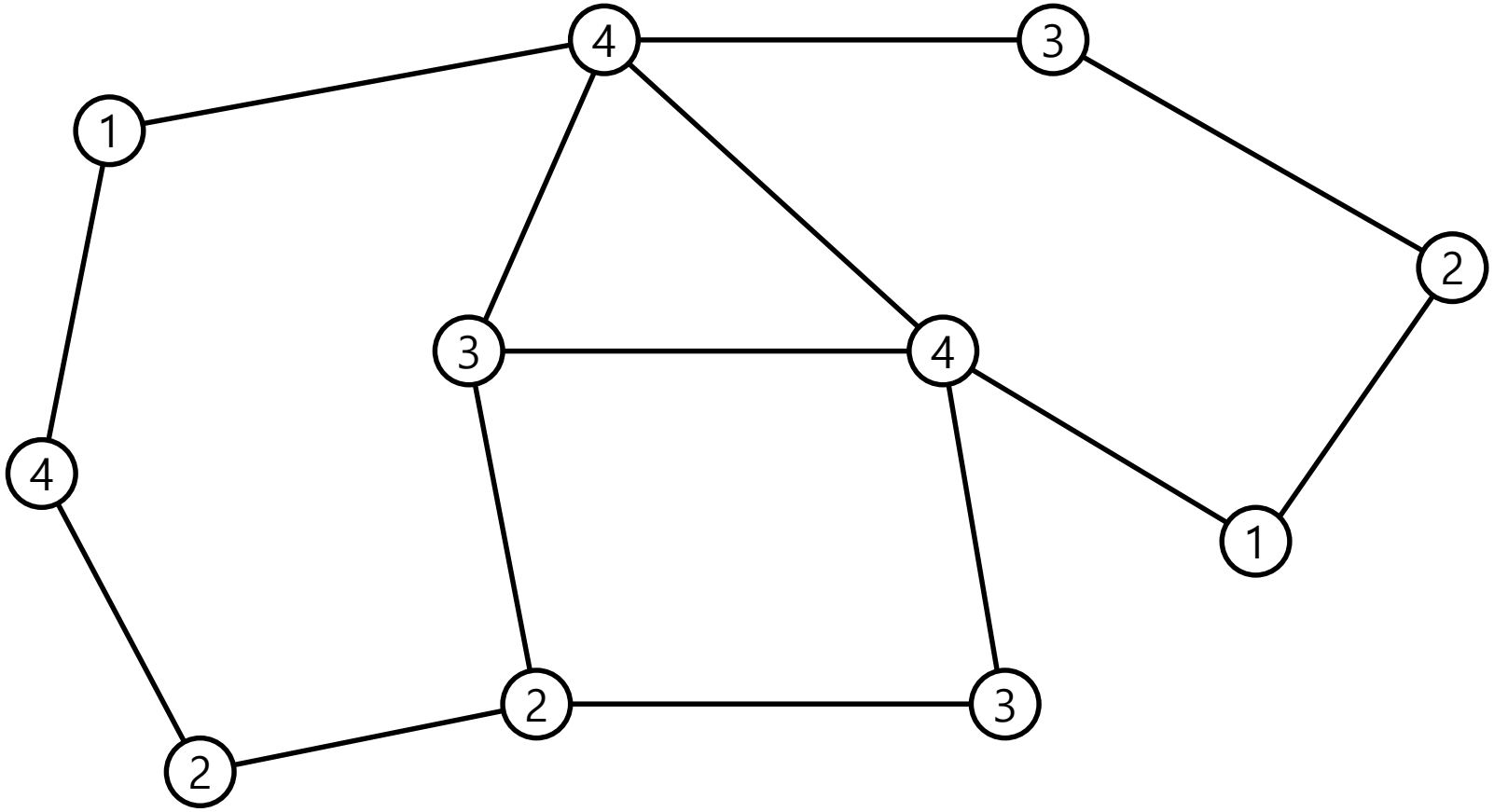
- 노드

  x = torch.tensor([[0,1,2,3],[4,5,6,7],[8,9,10,11]], dtype=torch.float32)
  - N×K 행렬에 표현 (N:노드수, K:특징벡터차원)
  - feature vector외 all-1, one-hot (embedding), one-hot-degree 등으로 표현 가능
- 에지

  edge_index = torch.tensor([[0, 1, 1, 2],
  [1, 0, 2, 1]], dtype=torch.long)
  - 2×M 행렬에 표현
  - undirected graph 표현 시 양방향에지를 모두 표현해야 함
- 그래프

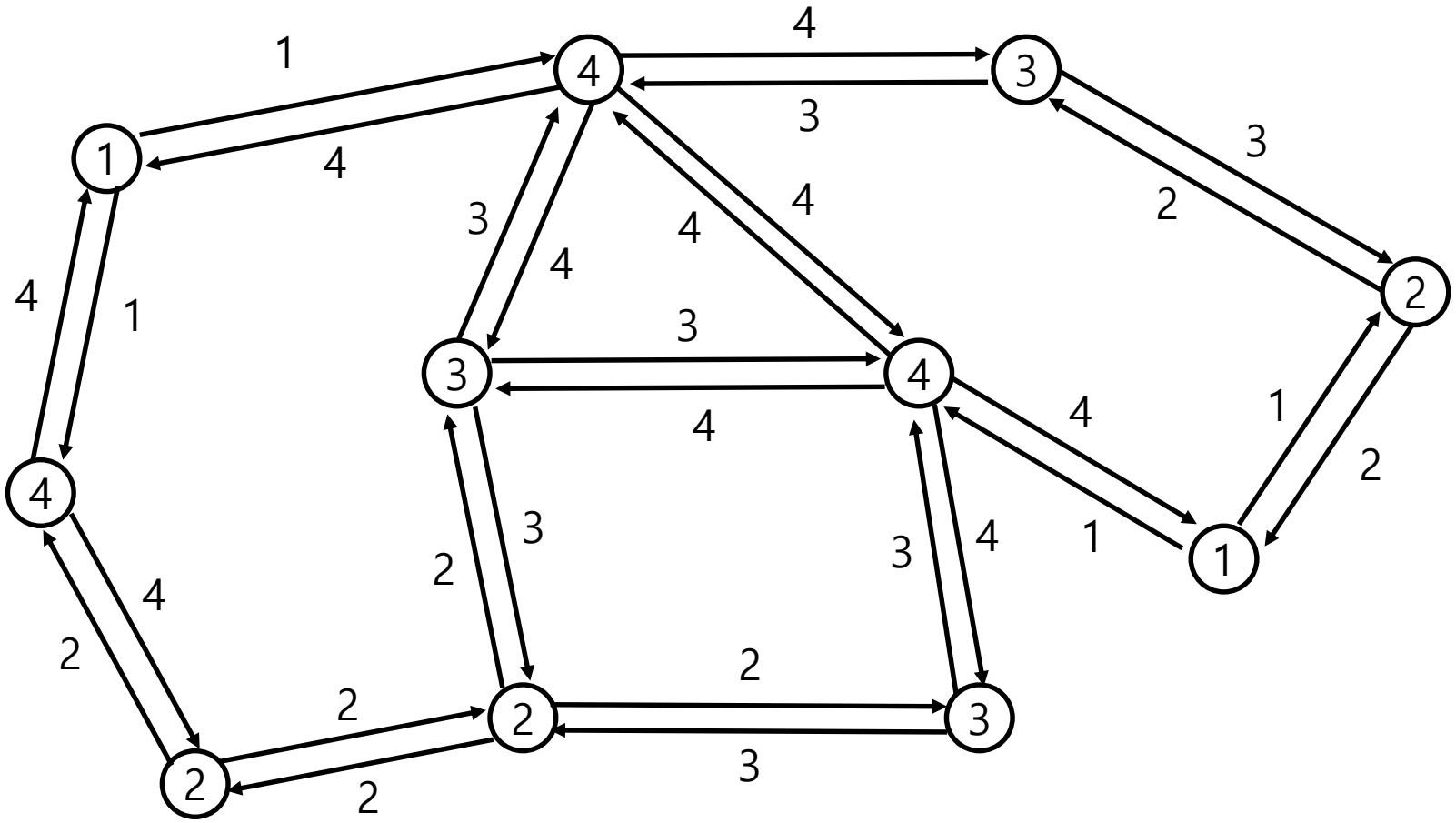
  data = Data(x=x, edge_index=edge_index)

Message Passing



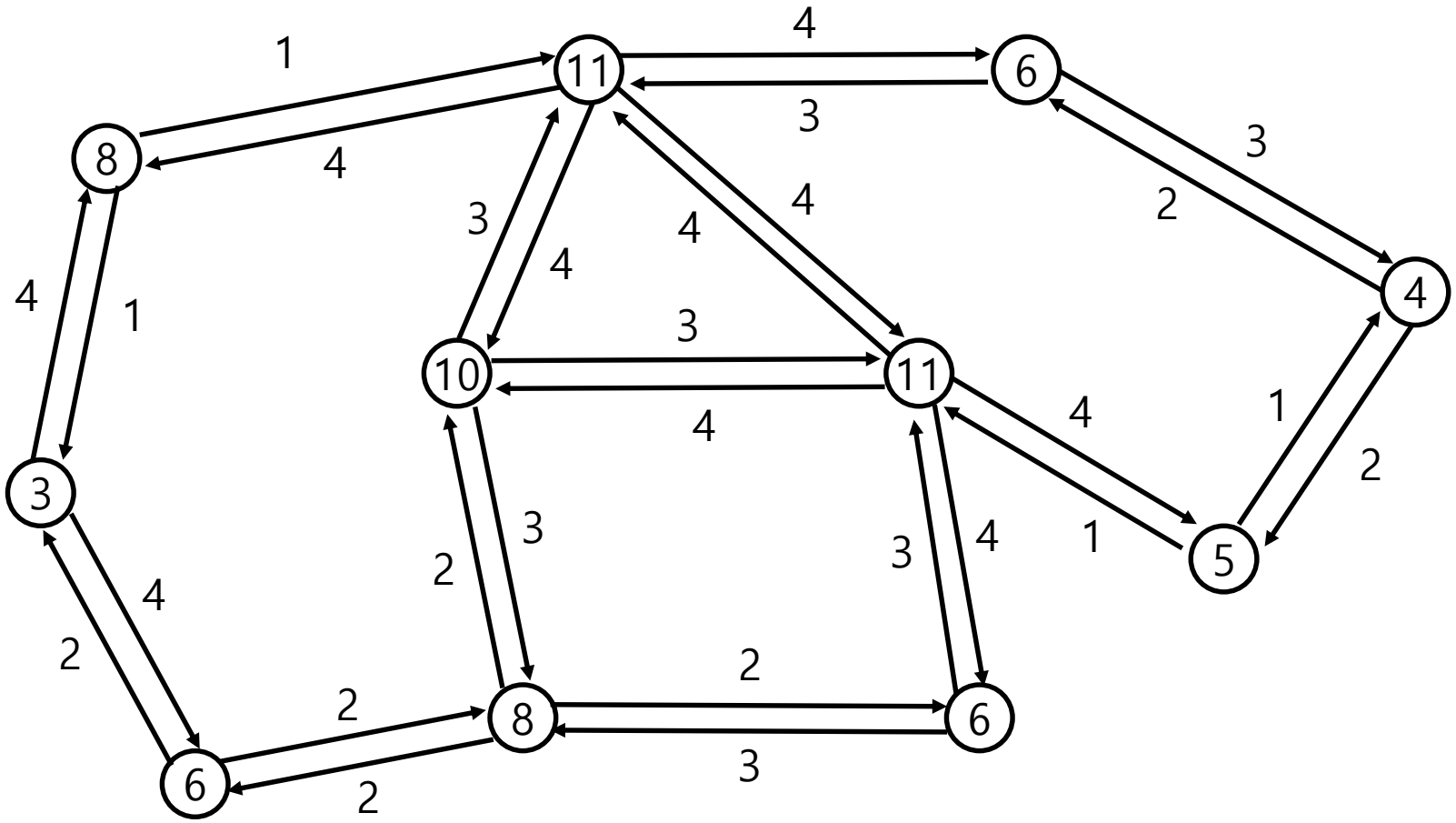
Message Passing

- 메시지 전파



Message Passing

- 합산 및 반영



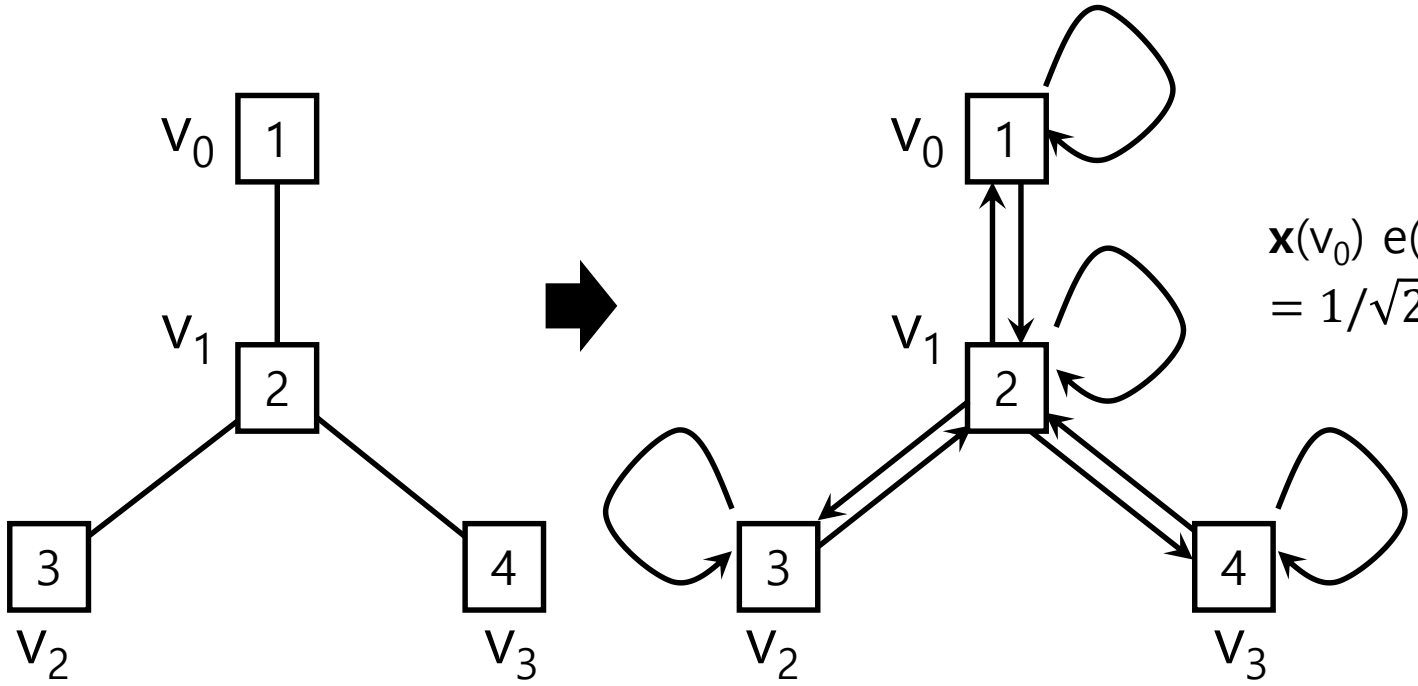
Message Passing in PyTorch Geometric

- 기본적인 계산 구조

  - 각 노드로부터 전파시킬 정보 및 에지 가중치 등 계산 (forward())
    ↓ propagate() 호출
  - 에지 별 전파시킬 값 계산 (message())
    ↓
  - 노드 별 합산 (aggregate(), 보통 {add,mean,max} 중 하나)
    ↓
  - 합산 결과 반영 (update(), 보통 그대로 assign)

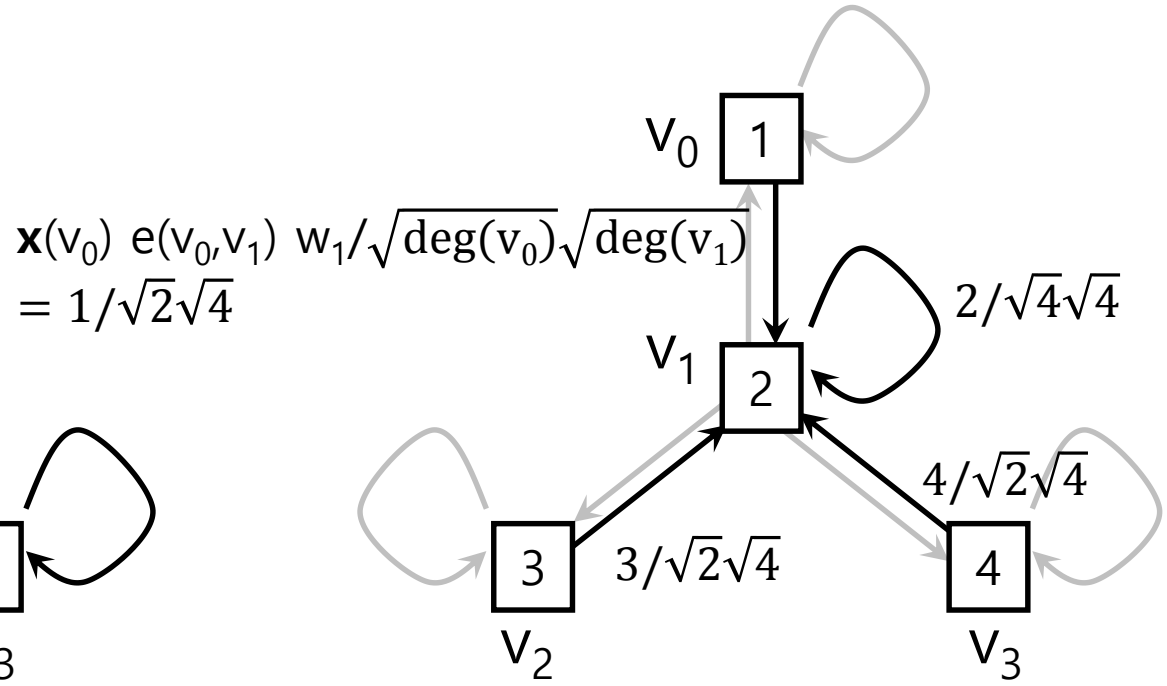
GCNConv

- 가장 기본적인 Graph Convolutional Layer $\mathbf{X}' = \hat{\mathbf{D}}^{-1/2}\hat{\mathbf{A}}\hat{\mathbf{D}}^{-1/2}\mathbf{X}\mathbf{\Theta},$



(a) 원래 그래프

(b) self-loop 추가



$\mathbf{x}'(v_1) = 1/\sqrt{8} + 2/4 + 3/\sqrt{8} + 4/\sqrt{8}$
$\fallingdotseq 3.3284$

(c) convolution 계산과정

GCNConv in PyTorch Geometric

- 각 노드로부터 전파시킬 정보 및 에지 가중치 등 계산 (forward())

```
def forward(self, x: Tensor, edge_index: Adj,
            edge_weight: OptTensor = None) -> Tensor:
    if self.normalize:
        edge_index, edge_weight = gcn_norm(  # yapf: disable
            edge_index, edge_weight, x.size(self.node_dim),
            self.improved, self.add_self_loops)
    x = x @ self.weight

    # propagate_type: (x: Tensor, edge_weight: OptTensor)
    out = self.propagate(edge_index, x=x, edge_weight=edge_weight,
                         size=None)

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

- 에지 별 전파시킬 값 계산 (message())

```
def message(self, x_j: Tensor, edge_weight: OptTensor) -> Tensor:
    return x_j if edge_weight is None else edge_weight.view(-1, 1) * x_j
```

GCNConv in PyTorch Geometric

- 각 노드로부터 전파시킬 정보 및 에지 가중치 등 계산 (forward())

```
def forward(self, x: Tensor, edge_index: Adj,
            edge_weight: OptTensor = None) -> Tensor:
    if self.normalize:
        edge_index, edge_weight = gcn_norm(  # yapf: disable
            edge_index, edge_weight, x.size(self.node_dim),
            self.improved, self.add_self_loops)
    x = x @ self.weight

    # propagate_type: (x: Tensor, edge_weight: OptTensor)
    out = self.propagate(edge_index, x=x, edge_weight=edge_weight,
                         size=None)

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

message의 인자 이름에 _i 또는 _j 가 붙어있는 경우 propagate()를 통해 넘겨준 인자 중에서 같은 이름의 노드별 텐서를 에지별 텐서로 변환해서 넘겨받음
예) x_i = x [edge_index[0]],
x_j = x [edge_index[1]],

- 에지 별 전파시킬 값 계산 (message())

```
def message(self, x_j: Tensor, edge_weight: OptTensor) -> Tensor:
    return x_j if edge_weight is None else edge_weight.view(-1, 1) * x_j
```

GCNConv in PyTorch Geometric

- 노드 별 합산 (aggregate()=add, __init__에서 정의되는 경우가 많음)

```
def __init__(self, in_channels: int, out_channels: int,
             improved: bool = False, cached: bool = False,
             add_self_loops: bool = True, normalize: bool = True,
             bias: bool = True, **kwargs):

    kwargs.setdefault('aggr', 'add')
    super(GCNConv, self).__init__(**kwargs)
```

- 합산 결과 반영 (보통 그대로 assign)

PyTorch의 기본적인 훈련 루프 구조

```
data = load_data()
model = Net()
optimizer = Optimizer()

model.train()
for epoch in range(NUM_EPOCHS):
    optimizer.zero_grad()
    pred = model(data.x)
    loss = compute_loss(pred, data.y)
    loss.backward()
    optimizer.step()
```

신경망 구조 정의 예

```
class Net(torch.nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = GCNConv(16, 8)
        self.conv2 = GCNConv(8, 4)

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

        x = self.conv1(x, edge_index)
        x = F.relu(x)
        x = F.dropout(x, training=self.training)

        x = self.conv2(x, edge_index)

        return F.log_softmax(x, dim=1)

data = Data( x= ..., edge_index= ... )
model = Net()

out = model(data)
```

코드 예제 (Node classification)

```python
from torch_geometric.datasets import Planetoid
from torch_geometric.data import DataLoader
from torch_geometric.utils import accuracy

import torch
import torch.nn.functional as F
from torch.nn import Linear
from torch_geometric.nn import GCNConv

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

class Net(torch.nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = GCNConv(dataset.num_node_features, 32)
        self.conv2 = GCNConv(32, dataset.num_classes)

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

        x = self.conv1(x, edge_index)
        x = F.relu(x)
        x = F.dropout(x, training=self.training)

        x = self.conv2(x, edge_index)

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

```python
dataset = Planetoid(root='.', name='Cora')
data = dataset[0].to(device)

model = Net().to(device)

optimizer = torch.optim.Adam(model.parameters(), lr=0.01, weight_decay=5e-4)

model.train()
for epoch in range(100):
    optimizer.zero_grad()
    out = model(data)
    loss = F.nll_loss(out[data.train_mask], data.y[data.train_mask])
    loss.backward()
    optimizer.step()
```

코드 예제 2 (Graph classification)

```python
class Net(torch.nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv = GCNConv(dataset.num_node_features, 32)
        self.linear = Linear(32, dataset.num_classes)

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

        x = self.conv(x, edge_index)
        x = F.relu(x)
        x = F.dropout(x, training=self.training)

        x = global_mean_pool(x, batch)

        x = self.linear(x)

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

(a) 신경망 모형 구조

```python
dataset = TUDataset(root='.', name='PROTEINS')
dataset = dataset.shuffle()
train_data = dataset[:800]
test_data = dataset[800:]
model = Net().to(device)
optimizer = torch.optim.Adam(model.parameters(), lr=0.01, weight_decay=5e-4)
```

(b) 데이터/모델/optimizer 정의

```python
model.train()
for epoch in range(100):
    loader = DataLoader(train_data, batch_size=32, shuffle=True)
    for batch in loader:
        optimizer.zero_grad()
        batch = batch.to(device)
        out = model(batch)
```

(c) 훈련 루프

그래프 별로 합산(평균)

batch는 32개의 graph의 노드(batch.x), 에지(batch.edge_index) 및 각 노드가 속한 그래프 번호(batch.batch) 등이 있음

코드 예제 3 (Link prediction)

```python
class Net(torch.nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = GCNConv(dataset.num_node_features, 16)
        self.linear1 = Linear(32, 1)

    def forward(self, x, edge_index, node_pair ):
        x = self.conv1(x, edge_index)
        x = F.relu(x)
        x = F.dropout(x, training=self.training)

        x = torch.cat( [x[node_pair[0]],x[node_pair[1]]], dim=1 )

        x = self.linear1(x)
        x = x.squeeze(1)

        return x
```

각 에지의 출발점, 도착점에 해당하는 텐서를 추출

(a) 신경망 모형 구조

```python
dataset = Planetoid(root='.', name='Cora')
data = dataset[0]
```

(b) 데이터 읽기

현존하는 edge를 학습셋/평가셋으로 분할

```python
train_test_split_edges( data )

n_pos = data.train_pos_edge_index.shape[1]
data.train_neg_edge_index = negative_sampling( data.train_pos_edge_index,
                                               num_neg_samples = n_pos )

n_neg = data.train_neg_edge_index.shape[1]
```

에지가 없는 노드쌍을 추출

(c) 노드 쌍 추출

```python
x = data.x
edge_index = data.train_pos_edge_index
node_pair = torch.cat( [ data.train_pos_edge_index, data.train_neg_edge_index ],
                       dim=1 )
y = torch.cat( [ torch.ones(n_pos), torch.zeros(n_neg) ] )
```

(d) 학습 데이터 구성

```python
model = Net()
optimizer = torch.optim.Adam(model.parameters(), lr=0.01, weight_decay=5e-4)
```

(e) 모델 및 optimizer 정의

```python
model.train()
for epoch in range(1):
    optimizer.zero_grad()
    out = model(x, edge_index, node_pair)
    loss = F.binary_cross_entropy_with_logits(out, y)
    loss.backward()
    optimizer.step()
```

(f) 훈련 루프