# Implement a Graph Convolutional Network in PyTorch

(Node Classification)

```
import dgl
```

→ Library for GNN

```
import torch
import torch.nn as nn
jmport torch.nn.functional as F

inport dgl.data
dataset = dgl.data.CoraGraphDataset()
```

dataset of papers & citations

this dataset Consist of one graph.

## Build the Model

```
from dgl.nn import GraphConv

class GCN(nn.Module):
    def __init__(self, in_feats, h_feats, n_classes):
        super(GCN,self).__init__()
        self.Conv1 = GraphConv(in_feats, h_feats)
        self.conv2 = GraphConv(h_feats, n_classes)
```

← just like any torch module.

```
def forward(self, g, in_feats):
    h = self.Conv1 (g, in_feats)
    h = F.relu(h)
    h = self. Conv2 (g, h)
    return h
```

## Training

```
def train (g, model):
    opt = torch.optim. Adam (model.parameters(), lr=0.01)
    best_val_acc, best_test_acc = 0, 0

    feats = g.ndata['feat']
    labels = g.ndata['label']
    train_mask = g.ndata['train_mask']
    val_mask = g.ndata['val_mask']
    test_mask = g.ndata['test_mask']
```

```
for e in epochs :
    logits = model (g, feats)
    pred = logits. argmax (1)

    loss = F.cross_entropy (logits [train_mask],
                                    labels [train_mask])
```

forward

```
train_acc = (pred[train_mask] == labels{train_mask])
            .float().mean()
val_acc = (pred[val_mask] == labels{val_mask])
            .float().mean()
test_acc = (pred[test_mask] == labels{test_mask])
            .float().mean()

if best_val_acc < val_acc:
    best_val_acc = val_acc
    best_test_acc = test_acc
```

```
opt.zero_grad()
loss.backward()
opt.step()
```

backward