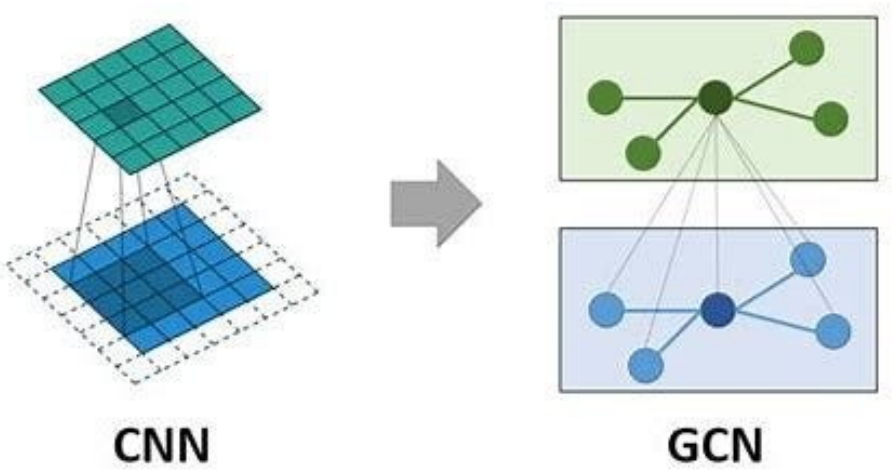
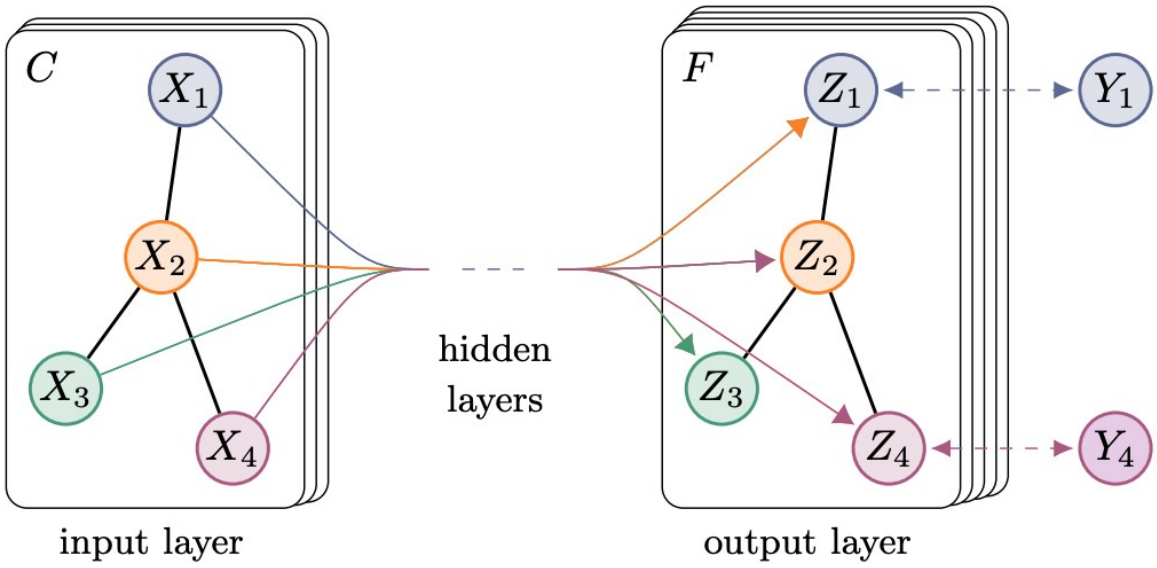
# Graph Convolutional Networks

Graph convolution, primarily implemented through Graph Convolutional Networks (GCNs), is a powerful deep learning technique designed to process data structured as graphs. Unlike traditional convolutional neural networks (CNNs) that operate on grid-like data (like images), GCNs are specifically tailored for non-Euclidean data where relationships between entities are represented as nodes and edges.




The core idea behind a GCN is to learn node embeddings (vector representations of nodes) that capture both the node's features and the graph's structural information. This is achieved by iteratively aggregating information from a node's neighbors.

A GCN typically consists of multiple graph convolutional layers stacked together. Each layer updates the feature representation of a node by performing an aggregation and transformation operation:



## Difference between GNN and GCN :

Think of base GNNs as a framework that:

- Compute messages
- Aggregate them
- Update node features

A GCN is one instantiation of that framework where you:

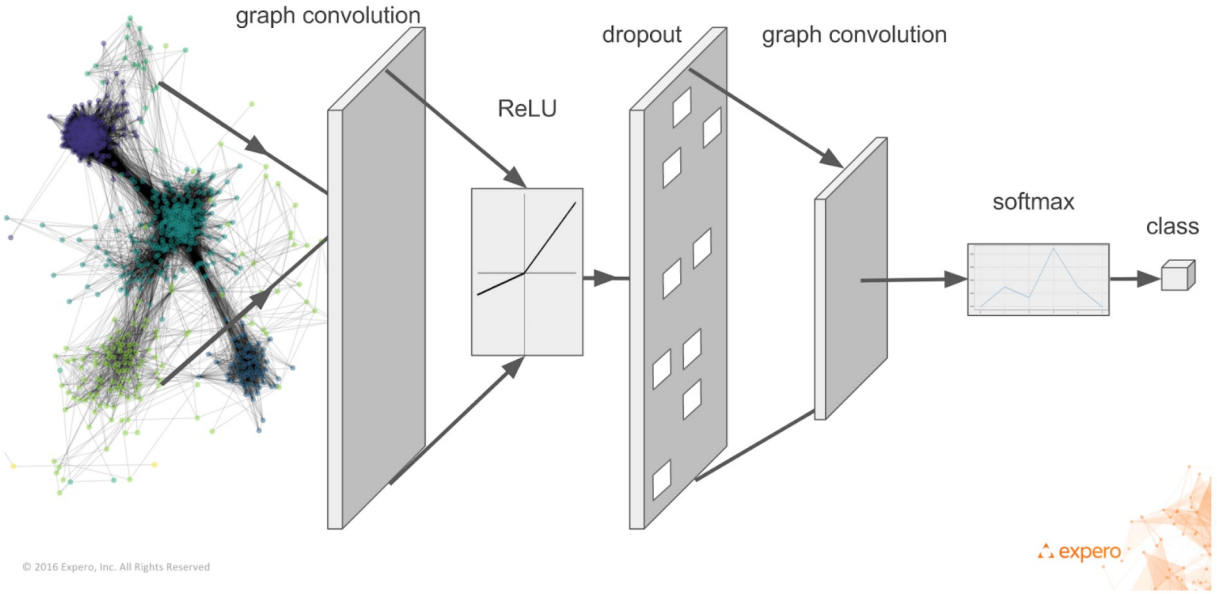
- Combine steps 1+2 via a normalized adjacency matrix (averaging neighbor features)
- Use a single linear transform + activation for updating

In [3]:
```
def load_dataset():
    print('..Loading Dataset../n')
    dataset = Planetoid(root = 'data/Cora', name='Cora');
    data = dataset[0];
    print(f'Dataset                   : {dataset}')
    print(f'Number of graphs          : {len(dataset)}')
    print(f'Number of nodes           : {data.x.shape[0]}')
    print(f'Number of edges           : {data.edge_index.shape[1]}')
    print(f'Number of node features   : {data.x.shape[1]}')
    print(f'Number of classes         : {dataset.num_classes}')
    print(f'Number of training nodes  : {data.train_mask.sum()}')
    print(f'Number of validation nodes: {data.val_mask.sum()}')
    print(f'Number of test nodes      : {data.test_mask.sum()}')
    print()
    return data, dataset.num_classes
```

# Model Architecture Breakdown

- A GCN applies repeated rounds of “neighborhood averaging + linear transformation + nonlinearity”, implemented efficiently via normalized adjacency matrix multiplication.
- You stack a few such layers, and finally decode the resulting node embeddings with a softmax (for nodes) or a global readout + MLP (for graphs).
- The core abstraction is that each layer smooths features across the graph structure and then mixes them through learnable weights.

for more details : https://arxiv.org/abs/1609.02907v4




## Dataset: Cora

- A citation network of scientific papers
- Nodes: 2,708 research papers
- Edges: Citation relationships (5,429 edges)
- Features: 1,433-dimensional bag-of-words vectors
- Task: Classify papers into 7 research areas

In [8]:
```
data, num_classes = load_dataset()
device = torch.device('cuda' if torch.cuda.is_available else 'cpu')
```

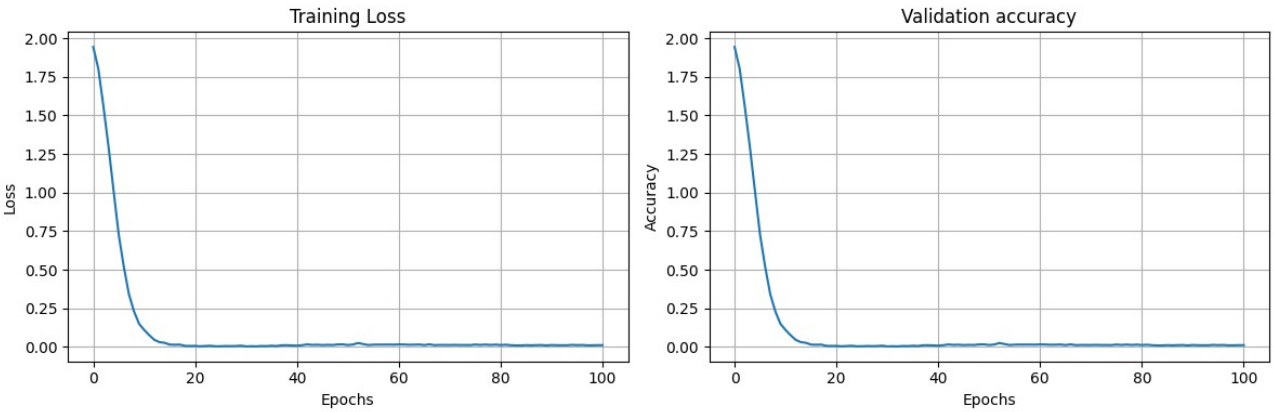
```
..Loading Dataset../n
Downloading https://github.com/kimiyoung/planetoid/raw/master/data/ind.cora.x
Downloading https://github.com/kimiyoung/planetoid/raw/master/data/ind.cora.tx
Downloading https://github.com/kimiyoung/planetoid/raw/master/data/ind.cora.allx
Downloading https://github.com/kimiyoung/planetoid/raw/master/data/ind.cora.y
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
Dataset                    : Cora()
Number of graphs           : 1
Number of nodes            : 2708
Number of edges            : 10556
Number of node features    : 1433
Number of classes          : 7
Number of training nodes   : 140
Number of validation nodes: 500
Number of test nodes       : 1000

Downloading https://github.com/kimiyoung/planetoid/raw/master/data/ind.cora.test.i
ndex
Processing...
Done!
```

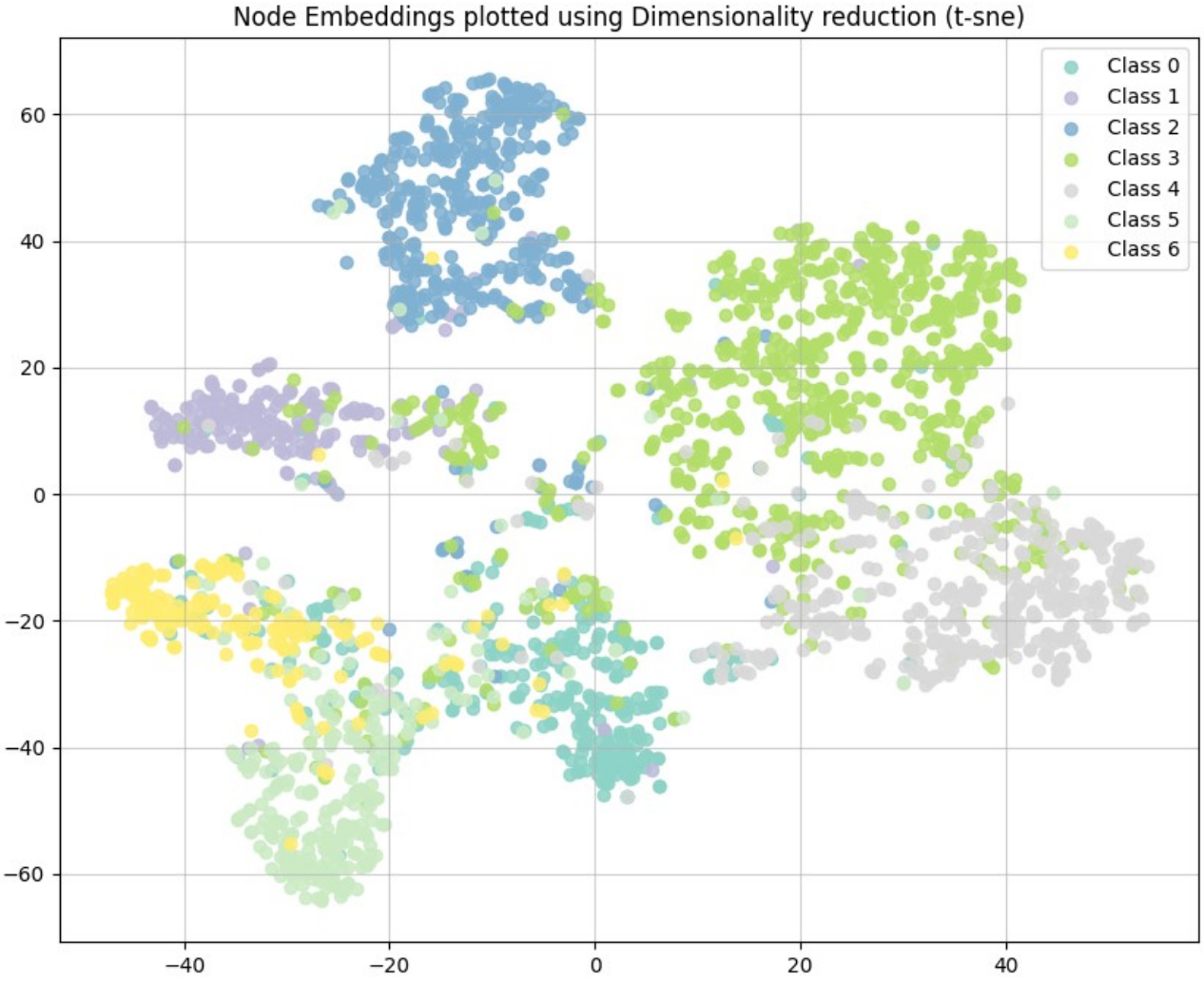
## Simple GNN with average aggreator

In [10]:
```
Graph_nn = GNN(
```

```
    inp_dim = data.x.shape[1], # 1433 features for CORA
    hidden_dim = 128,
    out_dim = num_classes,
    num_layers = 2
)
print(f"Number of parameter : {sum(p.numel() for p in Graph_nn.parameters())}\n")
print('Model Architecture:')
print(Graph_nn)
print()

Graph_nn, data = Graph_nn.to(device), data.to(device)

Graph_nn, train_loss, val_accuracy = train(Graph_nn, data, epochs = 100)

visualize_results(train_loss, val_accuracy)
visualize_embed(Graph_nn, data, num_classes)
```

```
Number of parameter : 184455

Model Architecture:
GNN(
  (layers): ModuleList(
    (0): Linear(in_features=1433, out_features=128, bias=True)
    (1): Linear(in_features=128, out_features=7, bias=True)
  )
  (dropout): Dropout(p=0.5, inplace=False)
)

..Training Model..

Epoch 0/100,  Loss : 1.9442,  Val acc : 0.0580
Epoch 20/100,  Loss : 0.0063,  Val acc : 0.7640
Epoch 40/100,  Loss : 0.0077,  Val acc : 0.7640
Epoch 60/100,  Loss : 0.0158,  Val acc : 0.7700
Epoch 80/100,  Loss : 0.0117,  Val acc : 0.7680
Epoch 100/100,  Loss : 0.0109,  Val acc : 0.7800

..Training Complete..
```



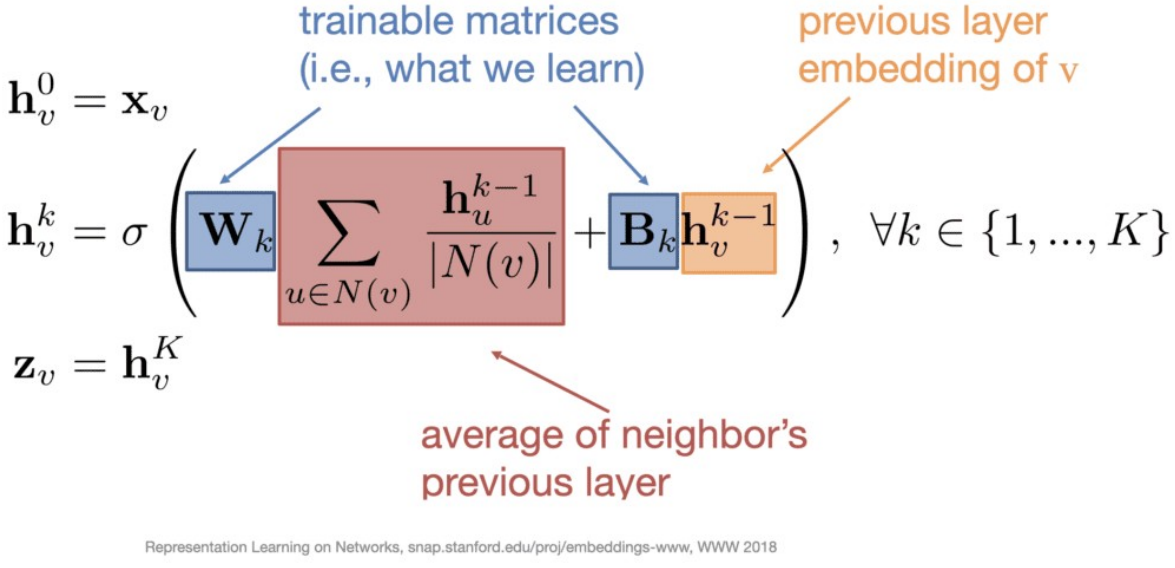



## Custom Graph Convolvution network

The equation to calculate the hidden output from the previous can be generalized as:

$$H^{(l+1)} = f(H^{(l)}, A)$$

Different choices of f result in different variants of the models. As a preview, the GCN paper applies the propagation rule below




Representation Learning on Networks, snap.stanford.edu/proj/embeddings-www, WWW 2018

or in closed form

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right)$$

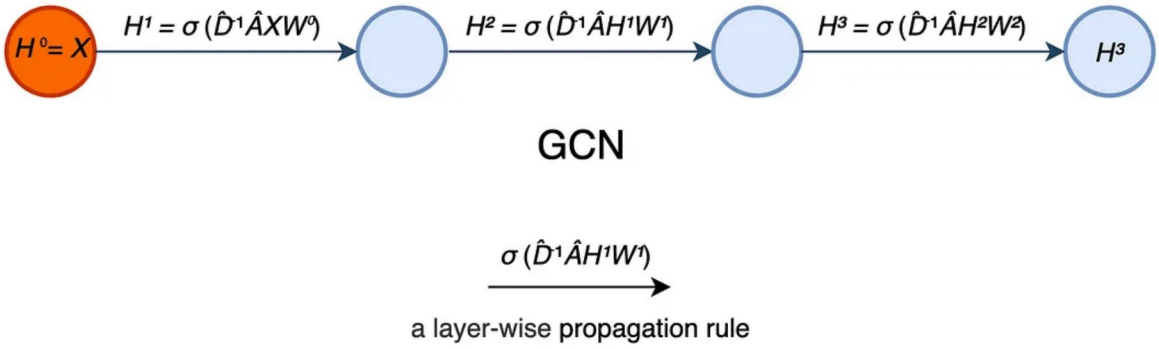
where

$$\tilde{A} = A + I_N$$

$$\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$$

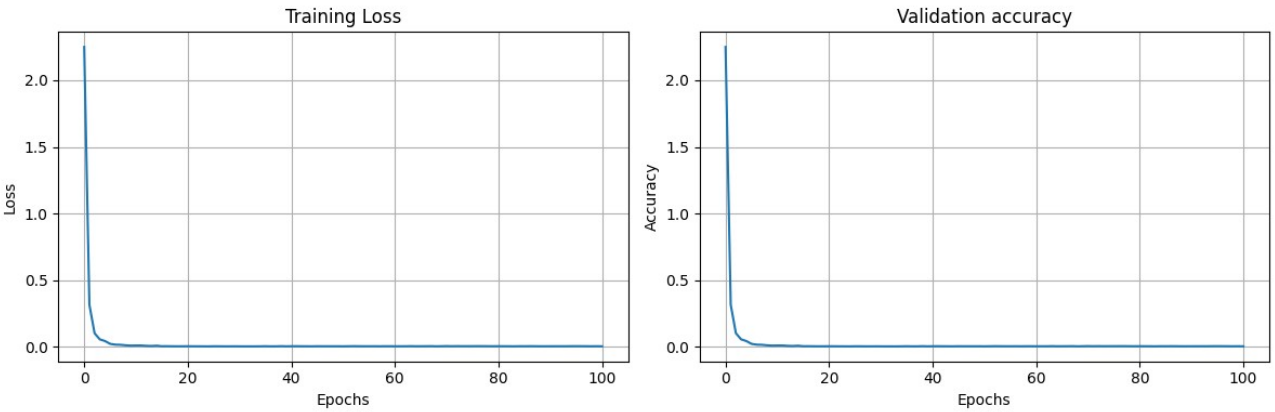
source : https://jonathan-hui.medium.com/graph-convolutional-networks-gcn-pooling-839184205692

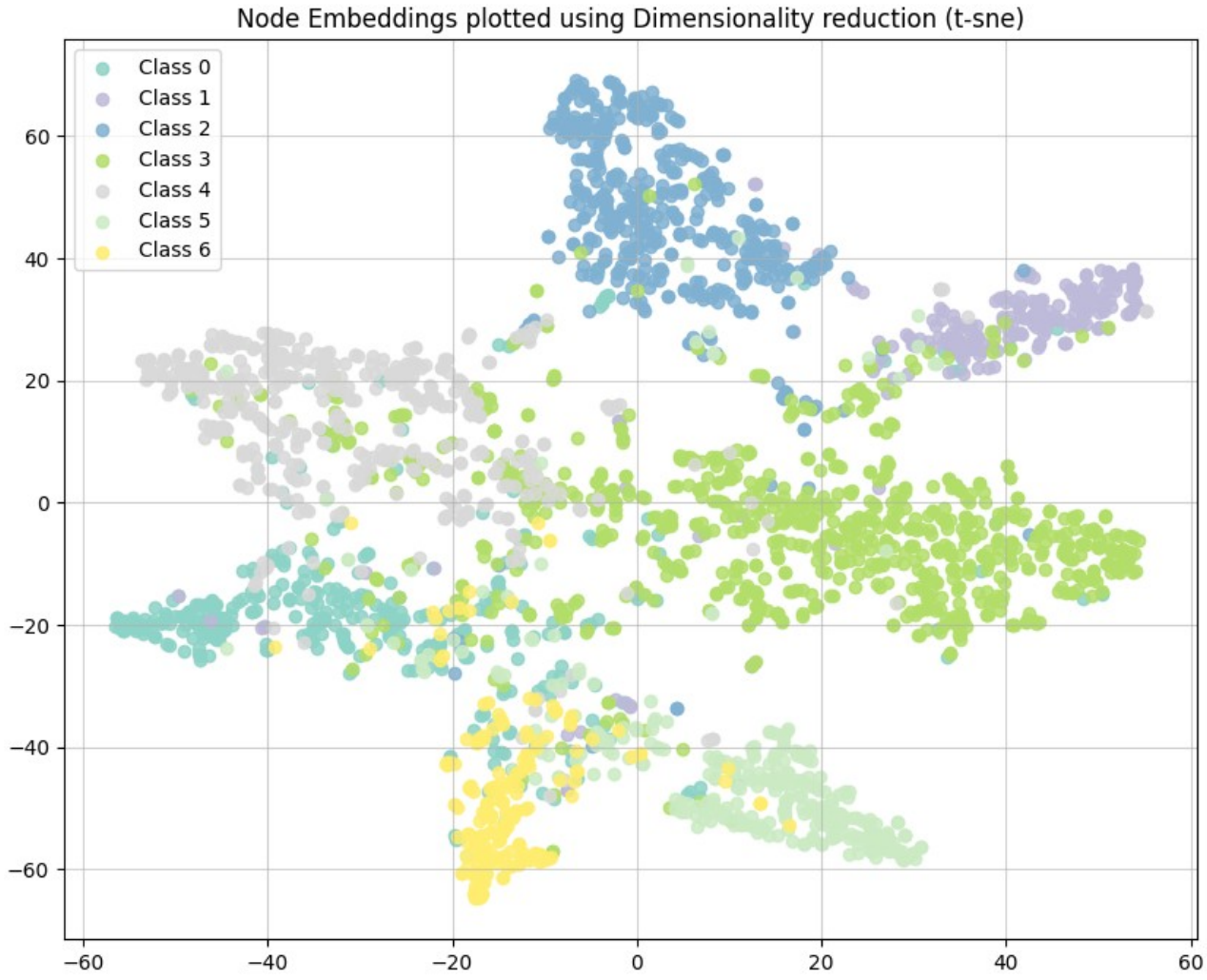
In [11]:
```
class GCNlayer(nn.Module):
    def __init__(self, in_feat, out_feat):
        super(GCNlayer, self).__init__()
        self.in_feat = in_feat
        self.out_feat = out_feat

        self.W = nn.Parameter(torch.FloatTensor(in_feat, out_feat))
        self.B = nn.Parameter(torch.FloatTensor(out_feat))

        self.reset_param()

    def reset_param(self):
        nn.init.xavier_uniform_(self.W)
        nn.init.zeros_(self.B)

    def forward(self, x, edges):
        # x : (N, D)   edges: (2, E) as [[src], [tgt]]
        N = x.size(0)

        edge_w_sloops, _ = add_self_loops(edges, num_nodes = N)

        # Calculating node degrees for normalization
        row, col = edge_w_sloops
        deg = torch.zeros(N, device = x.device)
        deg.scatter_add_(0, row, torch.ones(row.size(0), device=x.device))

        # compute normalization coefficients given by 1/(sqrt(deg(i) * deg(j)))
        deg_inv_sqrt = deg.pow(-0.5)
        deg_inv_sqrt[deg_inv_sqrt == float('inf')] = 0
        norm = deg_inv_sqrt[row] * deg_inv_sqrt[col]

        x_trans = torch.mm(x, self.W)
        out = torch.zeros_like(x_trans)

        messages = x_trans[row] * norm.view(-1, 1)
        out.scatter_add_(0, col.view(-1,1).expand(-1, x_trans.size(1)), messages)

        return out + self.B
```




In [13]:
```
Graph_nn = GCN(
    inp_dim = data.x.shape[1], # 1433 features for CORA
    hidden_dim = 128,
    out_dim = num_classes,
    num_layers = 2
)
print(f"Number of parameter : {sum(p.numel() for p in Graph_nn.parameters())}\n")
print('Model Architecture:')
print(Graph_nn)
print()

Graph_nn, data = Graph_nn.to(device), data.to(device)

Graph_nn, train_loss, val_accuracy = train(Graph_nn, data, epochs = 100)

visualize_results(train_loss, val_accuracy)
visualize_embed(Graph_nn, data, num_classes)
```

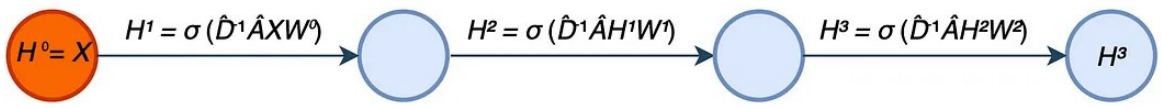
```
Number of parameter : 184711

Model Architecture:
GCN(
  (layers): ModuleList(
    (0-1): 2 x GCNlayer()
  )
  (b_norms): ModuleList(
    (0): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stat
s=True)
  )
)

..Training Model..

Epoch 0/100,  Loss : 2.2506,  Val acc : 0.0740
Epoch 20/100,  Loss : 0.0022,  Val acc : 0.7640
Epoch 40/100,  Loss : 0.0022,  Val acc : 0.7620
Epoch 60/100,  Loss : 0.0019,  Val acc : 0.7640
Epoch 80/100,  Loss : 0.0026,  Val acc : 0.7620
Epoch 100/100,  Loss : 0.0017,  Val acc : 0.7560

..Training Complete..
```




# GCN using PyTorch's GCNConv layer

**DCNConv** : The graph convolutional operator from the "Semi-supervised Classification with Graph Convolutional Networks" paper.




GCN

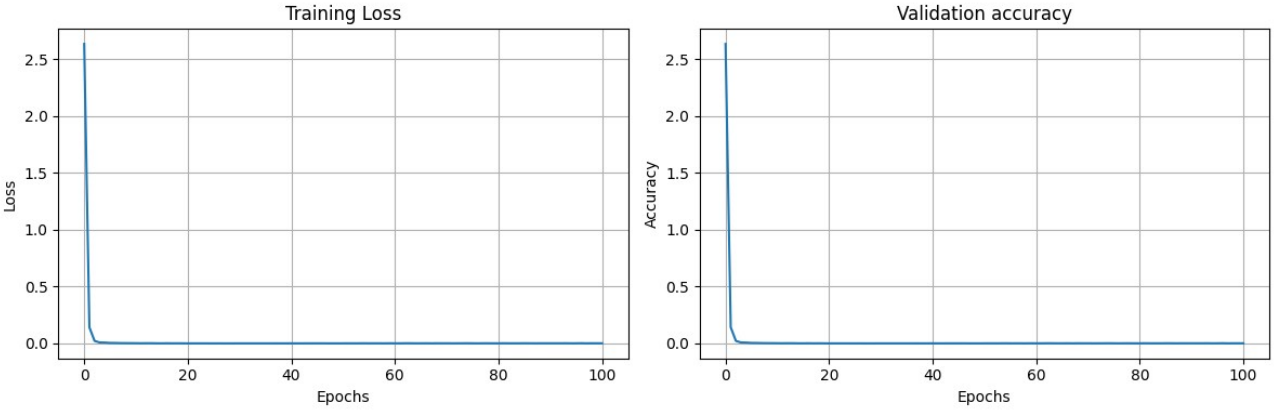
$\sigma(\hat{D}^{-1}\hat{A}H^1W^1)$

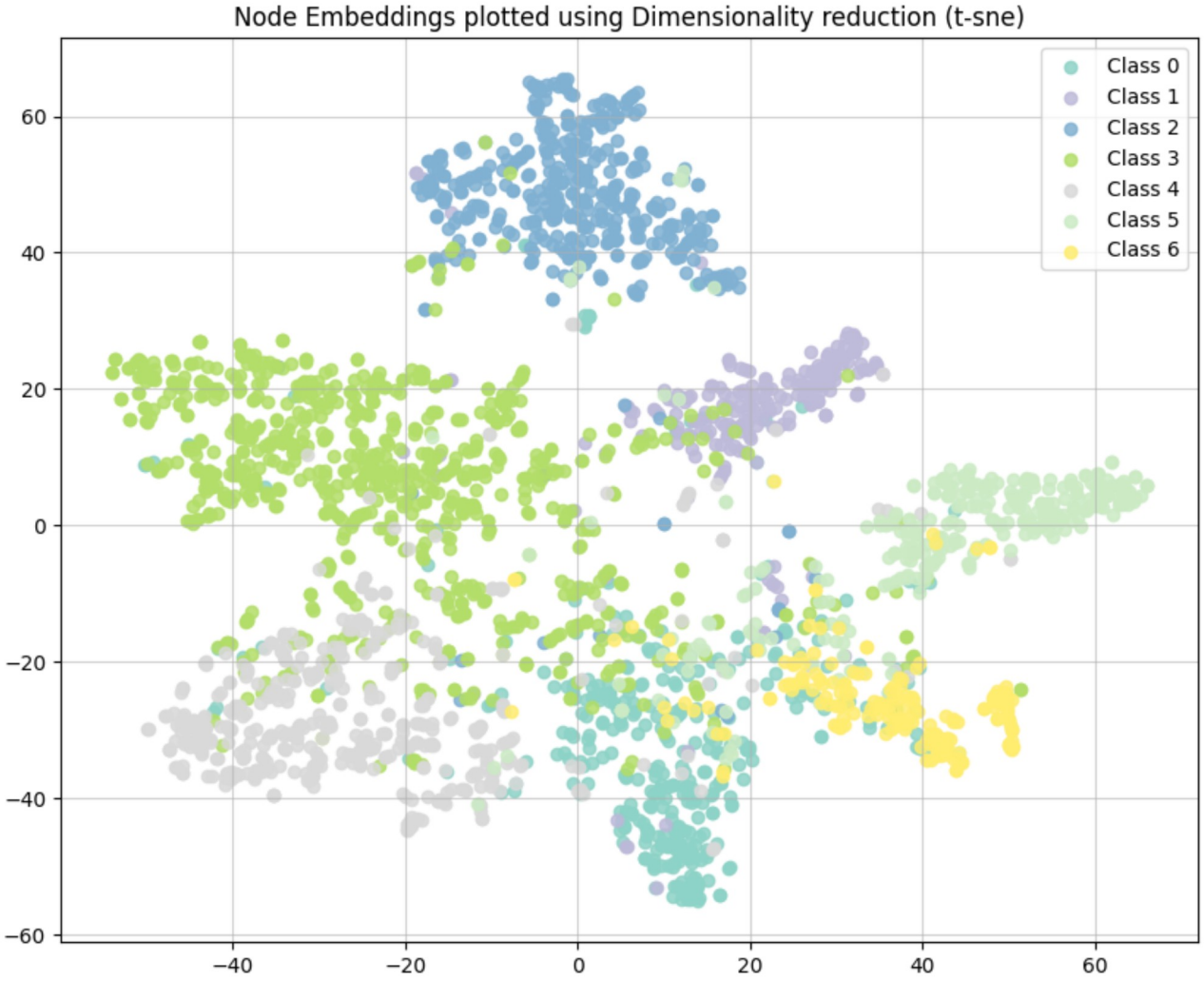
a layer-wise propagation rule

In [14]:
```
class GCN_torch(nn.Module):
    def __init__(self, inp_dim, hidden_dim, out_dim, num_layers=2, dropout=0.5):
        super(GCN_torch, self).__init__()
        self.num_layers = num_layers
        self.layers = nn.ModuleList()
        self.BN = nn.ModuleList()

        self.layers.append(GCNConv(inp_dim, hidden_dim))
        self.BN.append(nn.BatchNorm1d(hidden_dim))

        for _ in range(num_layers-2):
            self.layers.append(GCNConv(hidden_dim, hidden_dim))
            self.BN.append(nn.BatchNorm1d(hidden_dim))

        self.layers.append(GCNConv(hidden_dim, out_dim))
        self.dropout = dropout

    def forward(self, x, edge_index):

        for i in range(len(self.layers)-1):
            x = self.layers[i](x, edge_index)
            x = self.BN[i](x)
            if i == 0:
                res = torch.zeros_like(x, device=x.device)
            x = x+res
            res = torch.zeros_like(x, device=x.device)
            x = F.dropout(x, self.dropout, training = self.training)
            res = x

        out = self.layers[-1](x, edge_index)
        return out
```

In [15]:
```
Graph_nn = GCN_torch(
    inp_dim = data.x.shape[1], # 1433 features for CORA
    hidden_dim = 128,
    out_dim = num_classes,
    num_layers = 2
)
print(f"Number of parameter : {sum(p.numel() for p in Graph_nn.parameters())}\n")
print('Model Architecture:')
print(Graph_nn)
print()

Graph_nn, data = Graph_nn.to(device), data.to(device)

Graph_nn, train_loss, val_accuracy = train(Graph_nn, data, epochs = 100)

visualize_results(train_loss, val_accuracy)
visualize_embed(Graph_nn, data, num_classes)
```

```
Number of parameter : 184711

Model Architecture:
GCN_torch(
  (layers): ModuleList(
    (0): GCNConv(1433, 128)
    (1): GCNConv(128, 7)
  )
  (BN): ModuleList(
    (0): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stat
s=True)
  )
)

..Training Model..

Epoch 0/100,  Loss : 2.6344,  Val acc : 0.1580
Epoch 20/100,  Loss : 0.0008,  Val acc : 0.7300
Epoch 40/100,  Loss : 0.0012,  Val acc : 0.7500
Epoch 60/100,  Loss : 0.0013,  Val acc : 0.7500
Epoch 80/100,  Loss : 0.0014,  Val acc : 0.7360
Epoch 100/100,  Loss : 0.0016,  Val acc : 0.7440

..Training Complete..
```




Node Embeddings plotted using Dimensionality reduction (t-sne)
Class 0
Class 1
Class 2
Class 3
Class 4
Class 5
Class 6
60
40
20
0
−20
−40
−60
−40
−20
0
20
40
60

In [ ]: