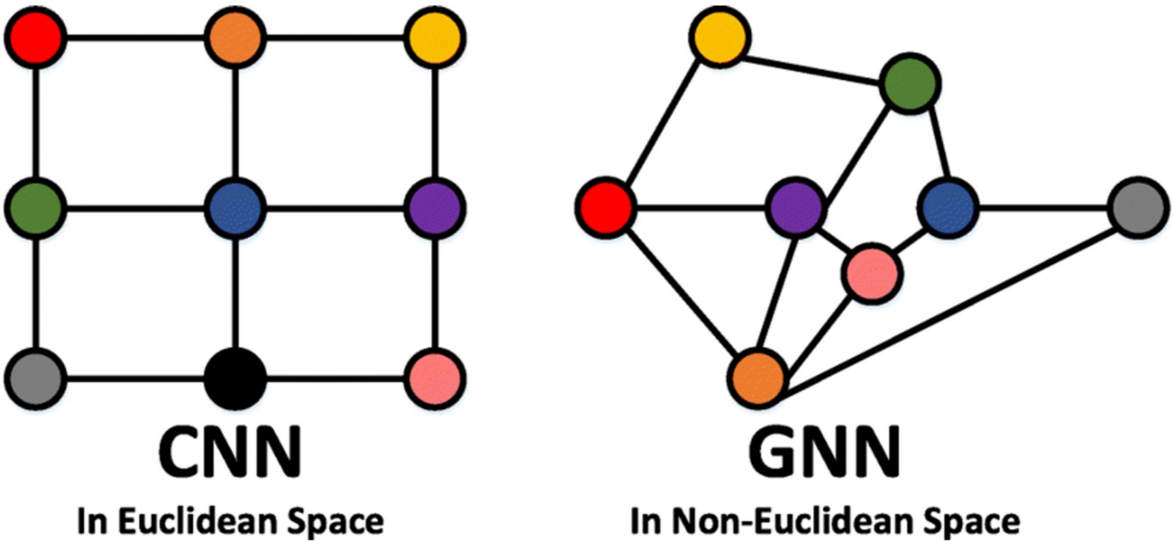
# Graph Neural Networks

Graph Neural Networks (GNNs) are needed despite other architectures because traditional neural networks (like CNNs and RNNs) are designed for data structured in Euclidean spaces (e.g., images as 2D grids, text as sequences). However, a vast amount of real-world data is inherently non-Euclidean and represented as graphs, where entities (nodes) are connected by relationships (edges).




A GNN processes graph-structured data where we have nodes (entities) connected by edges (relationships). The key idea is that each node's representation should be influenced by its neighbors in the graph.

## Message Passing Framework:

- **Aggregate**: Collect information from neighboring nodes
- **Update**: Combine neighbor information with node's own features
- **Transform**: Apply neural network layers

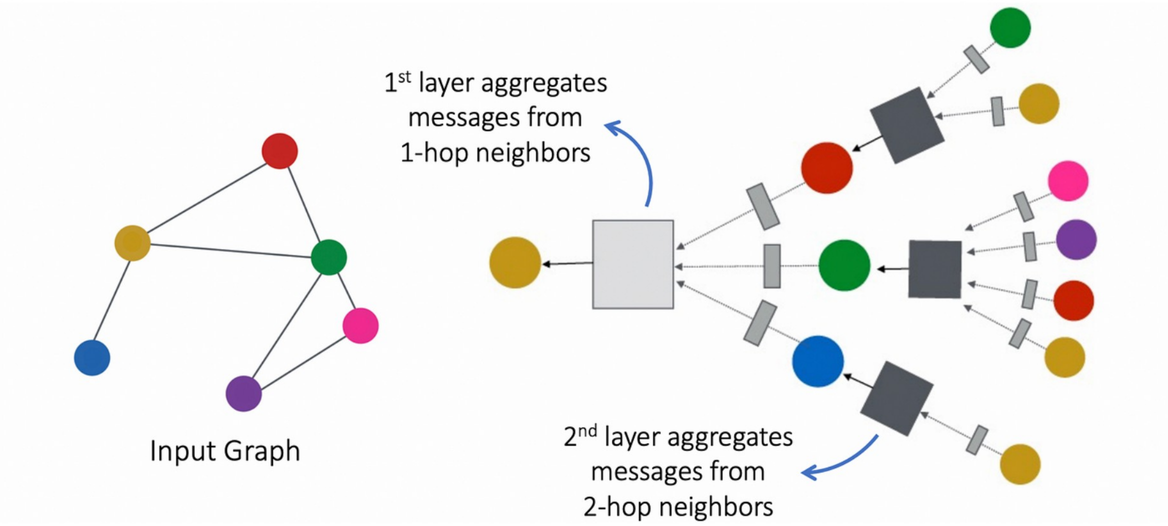
In [2]:
```
import numpy as np
import matplotlib.pyplot as plt
import torch
from torch import nn
import torch.nn.functional as F
from torch_geometric.datasets import Planetoid
from torch_geometric.data import Data
from torch_geometric.utils import to_dense_adj
from sklearn.manifold import TSNE
```

# 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

# Model Architecture Breakdown

- The model takes node features and edge connections as input
- Applies average aggregation to collect neighbor information
- Uses linear layers to transform features
- Outputs class predictions




In [4]:
```
class GNN(nn.Module):
    def __init__(self, inp_dim, hidden_dim, out_dim, num_layers=2):
        super(GNN, self).__init__()
        self.num_layers = num_layers
        self.layers = nn.ModuleList()
        self.layers.append(nn.Linear(inp_dim, hidden_dim))
        for _ in range(num_layers-2):
            self.layers.append(nn.Linear(hidden_dim, hidden_dim))
        self.layers.append(nn.Linear(hidden_dim, out_dim))
        self.dropout = nn.Dropout(0.5)

    def agg_neighbors(self, x, edge_index):
        # x : (num_nodes, feature_dim)
        # edge_index : (2, num_edge)
        device = x.device
        num_nodes = x.shape[0]
        agg = torch.zeros_like(x)
        neighbor_count = torch.zeros(num_nodes, dtype=torch.float, devi
```

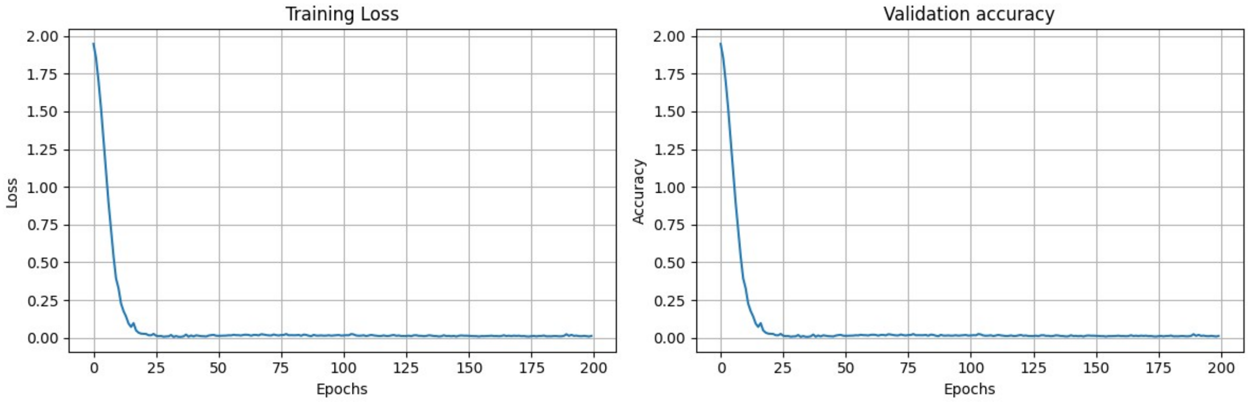
```python
        src_nodes = edge_index[0]
        tgt_nodes = edge_index[1]

        # for each node add src node features to tgt node's agg
        for i in range(edge_index.size(1)):
            src = src_nodes[i]
            tgt = tgt_nodes[i]

            # adding src node features to tgt node's agg
            agg[tgt] += x[src]
            neighbor_count[tgt] += 1

        neighbor_count = torch.clamp(neighbor_count, min = 1.0)
        agg = agg/neighbor_count.unsqueeze(1)
        return agg

    def forward(self, x, edge_ind):
        # returns embeddings after GNN processing
        for i, layer in enumerate(self.layers):
            agg_feat = self.agg_neighbors(x, edge_ind)
            x = agg_feat
            x = layer(x)
            if i < len(self.layers) - 1:
                x = F.relu(x)
                x = self.dropout(x)
        return x
```

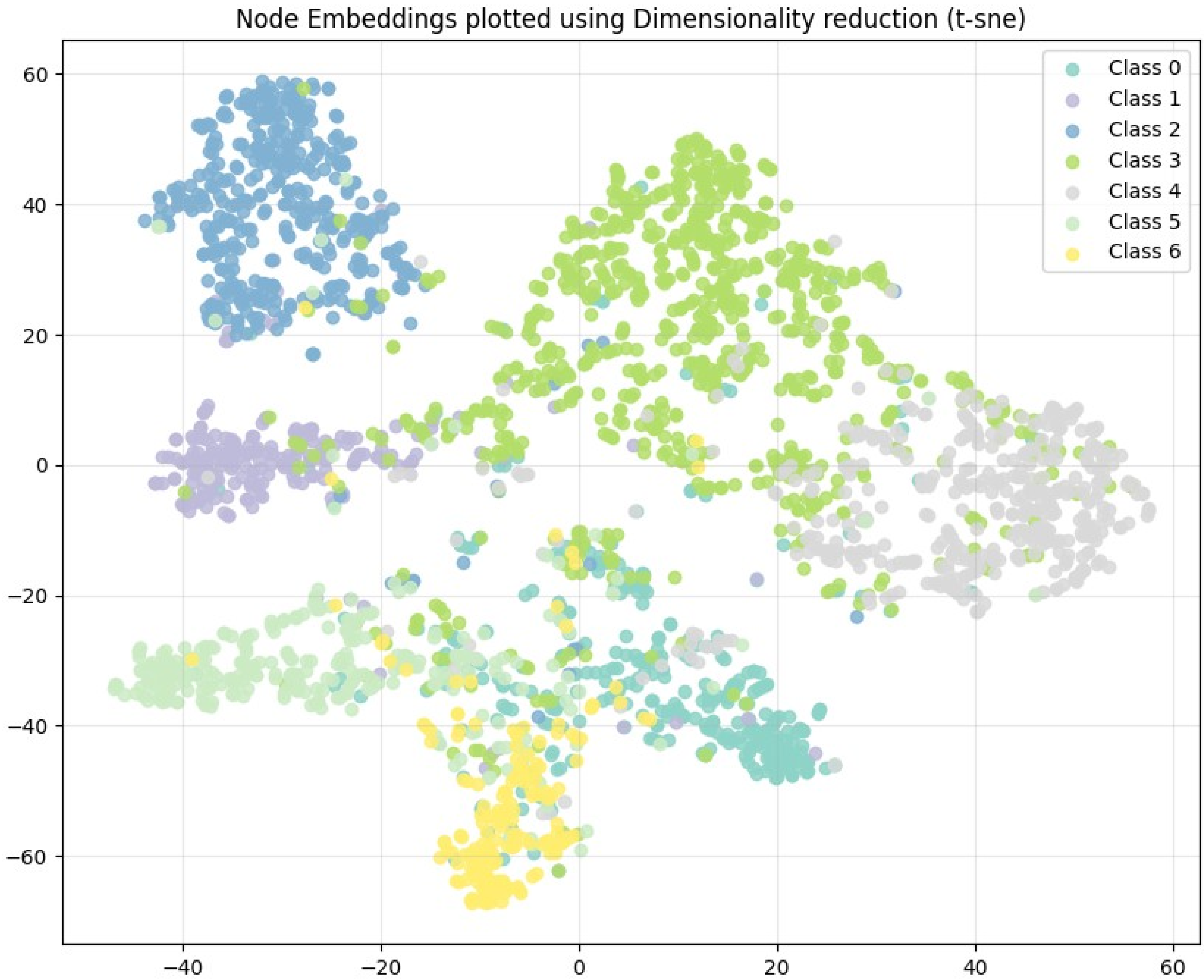
In [9]:

```python
data, num_classes = load_dataset()

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

Graph_nn = GNN(
    inp_dim = data.x.shape[1], # 1433 features for CORA
    hidden_dim = 64,
    out_dim = num_classes,
    num_layers = 2
)
print(f"Number of parameter : {sum(p.numel() for p in Graph_nn.paramete
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 = 200

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

```
..Loading Dataset../n
```

```
Downloading https://github.com/kimiyoung/planetoid/raw/master/data/ind.c
ora.x
Downloading https://github.com/kimiyoung/planetoid/raw/master/data/ind.c
ora.tx
Downloading https://github.com/kimiyoung/planetoid/raw/master/data/ind.c
ora.allx
Downloading https://github.com/kimiyoung/planetoid/raw/master/data/ind.c
ora.y
Downloading https://github.com/kimiyoung/planetoid/raw/master/data/ind.c
ora.ty
Downloading https://github.com/kimiyoung/planetoid/raw/master/data/ind.c
ora.ally
Downloading https://github.com/kimiyoung/planetoid/raw/master/data/ind.c
ora.graph
Downloading https://github.com/kimiyoung/planetoid/raw/master/data/ind.c
ora.test.index
Processing...
Done!
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

Number of parameter : 92231

Model Architecture:
GNN(
  (layers): ModuleList(
    (0): Linear(in_features=1433, out_features=64, bias=True)
    (1): Linear(in_features=64, out_features=7, bias=True)
  )
  (dropout): Dropout(p=0.5, inplace=False)
)

..Training Model..

Epoch 200/200,  Loss : 0.0130,  Val acc : 0.7640

..Training Complete..
```


Training Loss
Loss
Epochs
Validation accuracy
Accuracy
Epochs


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