Graph Convolutional Networks

> Chapter 1 of the [Graph Neural Network Course](#)

❤️ Created by [@maximelabonne](#).

Companion notebook to execute the code from the following article: https://mlabonne.github.io/blog/intrognn/

```
!pip -q install torch_geometric

import torch
import numpy as np
import networkx as nx
import matplotlib.pyplot as plt
```

```
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 63.1/63.1 kB 2.2 MB/s eta 0:00:00
  ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.1/1.1 MB 27.5 MB/s eta 0:00:00
```

```
from torch_geometric.datasets import KarateClub

# Import dataset from PyTorch Geometric
dataset = KarateClub()

# Print information
print(dataset)
print('------------')
print(f'Number of graphs: {len(dataset)}')
print(f'Number of features: {dataset.num_features}')
print(f'Number of classes: {dataset.num_classes}')
```

```
KarateClub()
------------
Number of graphs: 1
Number of features: 34
Number of classes: 4
```

```
#Chat GPT -- make labels 0 or 1 only instead of 0,1,2,3
from torch_geometric.datasets import KarateClub
from torch_geometric.data import Data

class CustomKarateClub(KarateClub):
    def __init__(self, transform=None):
        super().__init__(transform=transform)

        # Modify the labels to have only 2 classes
        # Assuming original classes are labeled as 0, 1, 2, 3
        # You can change the mapping as per your requirements
        new_labels = []
        for label in self.data.y:
            if label in [0, 1]:  # Map original classes 0 and 1 to new class 0
                new_labels.append(0)
            else:  # Map original classes 2 and 3 to new class 1
                new_labels.append(1)
        # Update the labels in the data object
        self.data.y = torch.tensor(new_labels)

# Usage
custom_karate_data = CustomKarateClub()
print(custom_karate_data.data.y)  # Check the new labels
```

```
tensor([0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
        1, 1, 0, 0, 1, 0, 0, 1, 0, 0])
/usr/local/lib/python3.10/dist-packages/torch_geometric/data/in_memory_dataset.py:300: UserWarning: It is not recommended to directly access the internal storage format `data` of an
  warnings.warn(msg)
```

```
dataset = CustomKarateClub()
```

Start coding or generate with AI.

```
# Print first element
print(f'Graph: {dataset[0]}')
```

```
Graph: Data(x=[34, 34], edge_index=[2, 156], y=[34], train_mask=[34])
```

```
data = dataset[0]

print(f'x = {data.x.shape}')
print(data.x)
```

```
x = torch.Size([34, 34])
tensor([[1., 0., 0.,  ..., 0., 0., 0.],
        [0., 1., 0.,  ..., 0., 0., 0.],
        [0., 0., 1.,  ..., 0., 0., 0.],
        ...,
        [0., 0., 0.,  ..., 1., 0., 0.],
        [0., 0., 0.,  ..., 0., 1., 0.],
        [0., 0., 0.,  ..., 0., 0., 1.]])
```

```
print(f'edge_index = {data.edge_index.shape}')
print(data.edge_index)
```

```
edge_index = torch.Size([2, 156])
tensor([[ 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  1,  1,
          1,  1,  1,  1,  1,  1,  1,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  3,
          3,  3,  3,  3,  3,  4,  4,  4,  5,  5,  5,  5,  6,  6,  6,  6,  7,  7,
          7,  7,  8,  8,  8,  8,  8,  9,  9, 10, 10, 10, 11, 12, 12, 13, 13, 13,
         13, 13, 14, 14, 15, 15, 16, 16, 17, 17, 18, 18, 19, 19, 19, 20, 20, 21,
         21, 22, 22, 23, 23, 23, 23, 23, 24, 24, 24, 25, 25, 25, 26, 26, 27, 27,
         27, 27, 28, 28, 28, 29, 29, 29, 29, 30, 30, 30, 30, 31, 31, 31, 31, 31,
         31, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 33, 33, 33, 33, 33,
         33, 33, 33, 33, 33, 33, 33, 33, 33, 33, 33, 33],
        [ 1,  2,  3,  4,  5,  6,  7,  8, 10, 11, 12, 13, 17, 19, 21, 31,  0,  2,
          3,  7, 13, 17, 19, 21, 30,  0,  1,  3,  7,  8,  9, 13, 27, 28, 32,  0,
          1,  2,  7, 12, 13,  0,  6, 10,  0,  6, 10, 16,  0,  4,  5, 16,  0,  1,
          2,  3,  0,  2, 30, 32, 33,  2, 33,  0,  4,  5,  0,  0,  3,  0,  1,  2,
          3, 33, 32, 33, 32, 33,  5,  6,  0,  1, 32, 33,  0,  1, 33, 32, 33,  0,
          1, 32, 33, 25, 27, 29, 32, 33, 25, 27, 31, 23, 24, 31, 29, 33,  2, 23,
         24, 33,  2, 31, 33, 23, 26, 32, 33,  1,  8, 32, 33,  0, 24, 25, 28, 32,
         33,  2,  8, 14, 15, 18, 20, 22, 23, 29, 30, 31, 33,  8,  9, 13, 14, 15,
         18, 19, 20, 22, 23, 26, 27, 28, 29, 30, 31, 32]])
```

```
from torch_geometric.utils import to_dense_adj

A = to_dense_adj(data.edge_index)[0].numpy().astype(int)
print(f'A = {A.shape}')
print(A)
```

```
A = (34, 34)
[[0 1 1 ... 1 0 0]
 [1 0 1 ... 0 0 0]
 [1 1 0 ... 0 1 0]
 ...
 [1 0 0 ... 0 1 1]
 [0 0 1 ... 1 0 1]
 [0 0 0 ... 1 1 0]]
```

```
print(f'y = {data.y.shape}')
print(data.y)
```

```
y = torch.Size([34])
tensor([0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
        1, 1, 0, 0, 1, 0, 0, 1, 0, 0])
```

```
print(f'train_mask = {data.train_mask.shape}')
print(data.train_mask)
```

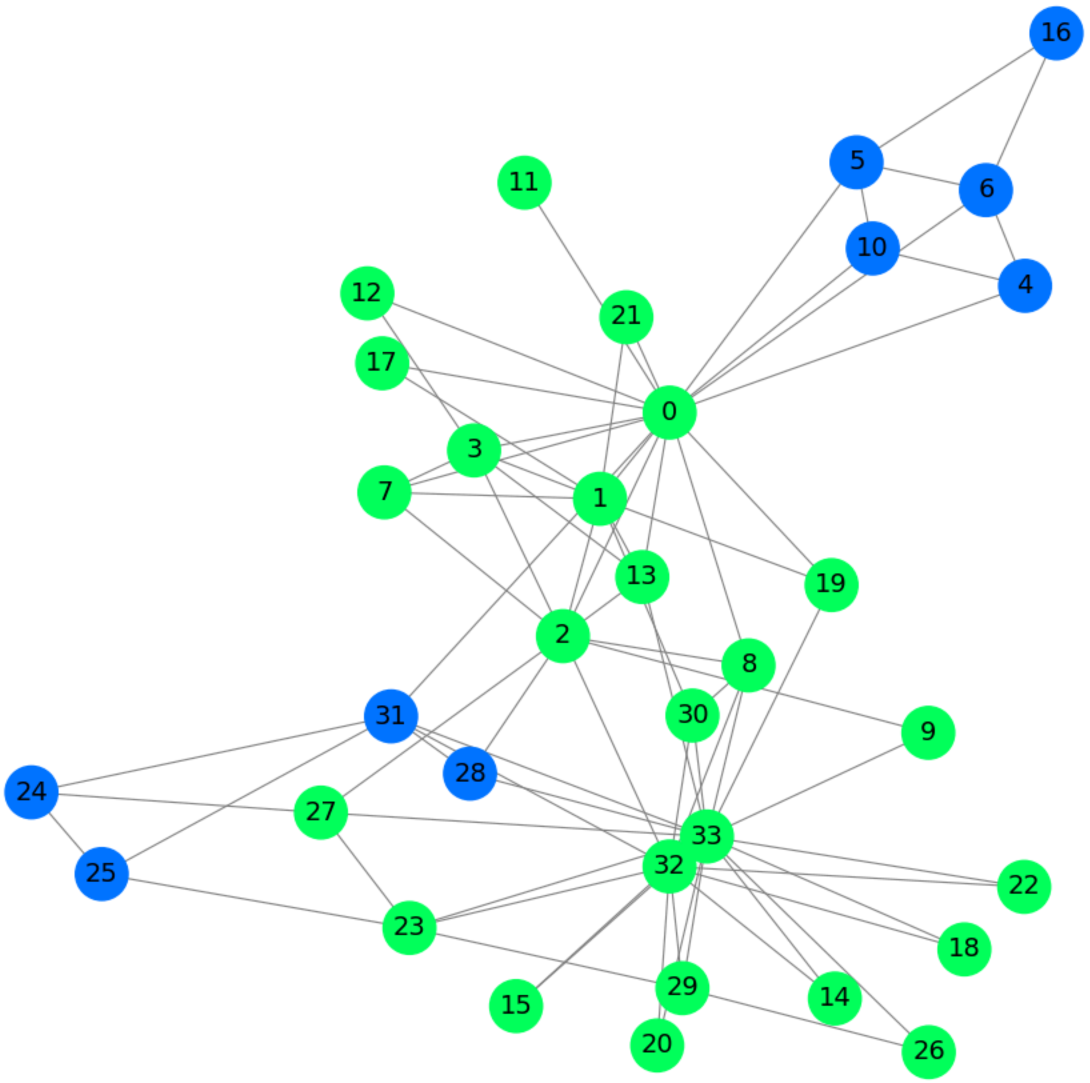
```
train_mask = torch.Size([34])
tensor([ True, False, False, False,  True, False, False, False,  True, False,
        False, False, False, False, False, False, False, False, False, False,
        False, False, False, False,  True, False, False, False, False, False,
        False, False, False, False])
```

```
print(f'Edges are directed: {data.is_directed()}')
print(f'Graph has isolated nodes: {data.has_isolated_nodes()}')
print(f'Graph has loops: {data.has_self_loops()}')
```

```
Edges are directed: False
Graph has isolated nodes: False
Graph has loops: False
```

```
from torch_geometric.utils import to_networkx

G = to_networkx(data, to_undirected=True)
plt.figure(figsize=(12,12))
plt.axis('off')
nx.draw_networkx(G,
                pos=nx.spring_layout(G, seed=0),
                with_labels=True,
                node_size=800,
                node_color=data.y,
                cmap="hsv",
                vmin=-2,
                vmax=3,
                width=0.8,
                edge_color="grey",
                font_size=14
                )
plt.show()
```



```
from torch.nn import Linear
from torch_geometric.nn import GCNConv
```

```
class GCN(torch.nn.Module):
    def __init__(self):
        super().__init__()
        self.gcn = GCNConv(dataset.num_features, 3)
        self.out = Linear(3, dataset.num_classes)

    def forward(self, x, edge_index):
        h = self.gcn(x, edge_index).relu()
        z = self.out(h)
        return h, z

model = GCN()
print(model)
```

```
GCN(
  (gcn): GCNConv(34, 3)
  (out): Linear(in_features=3, out_features=2, bias=True)
)
```

```
criterion = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.02)

# Calculate accuracy
def accuracy(pred_y, y):
    return (pred_y == y).sum() / len(y)

# Data for animations
embeddings = []
losses = []
accuracies = []
outputs = []

# Training loop
for epoch in range(201):
    # Clear gradients
    optimizer.zero_grad()

    # Forward pass
    h, z = model(data.x, data.edge_index)

    # Calculate loss function
    loss = criterion(z, data.y)

    # Calculate accuracy
    acc = accuracy(z.argmax(dim=1), data.y)

    # Compute gradients
    loss.backward()

    # Tune parameters
    optimizer.step()

    # Store data for animations
    embeddings.append(h)
    losses.append(loss)
    accuracies.append(acc)
    outputs.append(z.argmax(dim=1))

    # Print metrics every 10 epochs
    if epoch % 10 == 0:
        print(f'Epoch {epoch:>3} | Loss: {loss:.2f} | Acc: {acc*100:.2f}%')
```

```
Epoch   0 | Loss: 0.69 | Acc: 73.53%
Epoch  10 | Loss: 0.51 | Acc: 73.53%
Epoch  20 | Loss: 0.39 | Acc: 73.53%
Epoch  30 | Loss: 0.29 | Acc: 73.53%
Epoch  40 | Loss: 0.21 | Acc: 94.12%
Epoch  50 | Loss: 0.15 | Acc: 100.00%
Epoch  60 | Loss: 0.11 | Acc: 100.00%
Epoch  70 | Loss: 0.08 | Acc: 100.00%
Epoch  80 | Loss: 0.07 | Acc: 100.00%
Epoch  90 | Loss: 0.05 | Acc: 100.00%
Epoch 100 | Loss: 0.04 | Acc: 100.00%
Epoch 110 | Loss: 0.04 | Acc: 100.00%
Epoch 120 | Loss: 0.03 | Acc: 100.00%
Epoch 130 | Loss: 0.03 | Acc: 100.00%
Epoch 140 | Loss: 0.02 | Acc: 100.00%
Epoch 150 | Loss: 0.02 | Acc: 100.00%
Epoch 160 | Loss: 0.02 | Acc: 100.00%
Epoch 170 | Loss: 0.02 | Acc: 100.00%
Epoch 180 | Loss: 0.02 | Acc: 100.00%
Epoch 190 | Loss: 0.02 | Acc: 100.00%
Epoch 200 | Loss: 0.01 | Acc: 100.00%
```

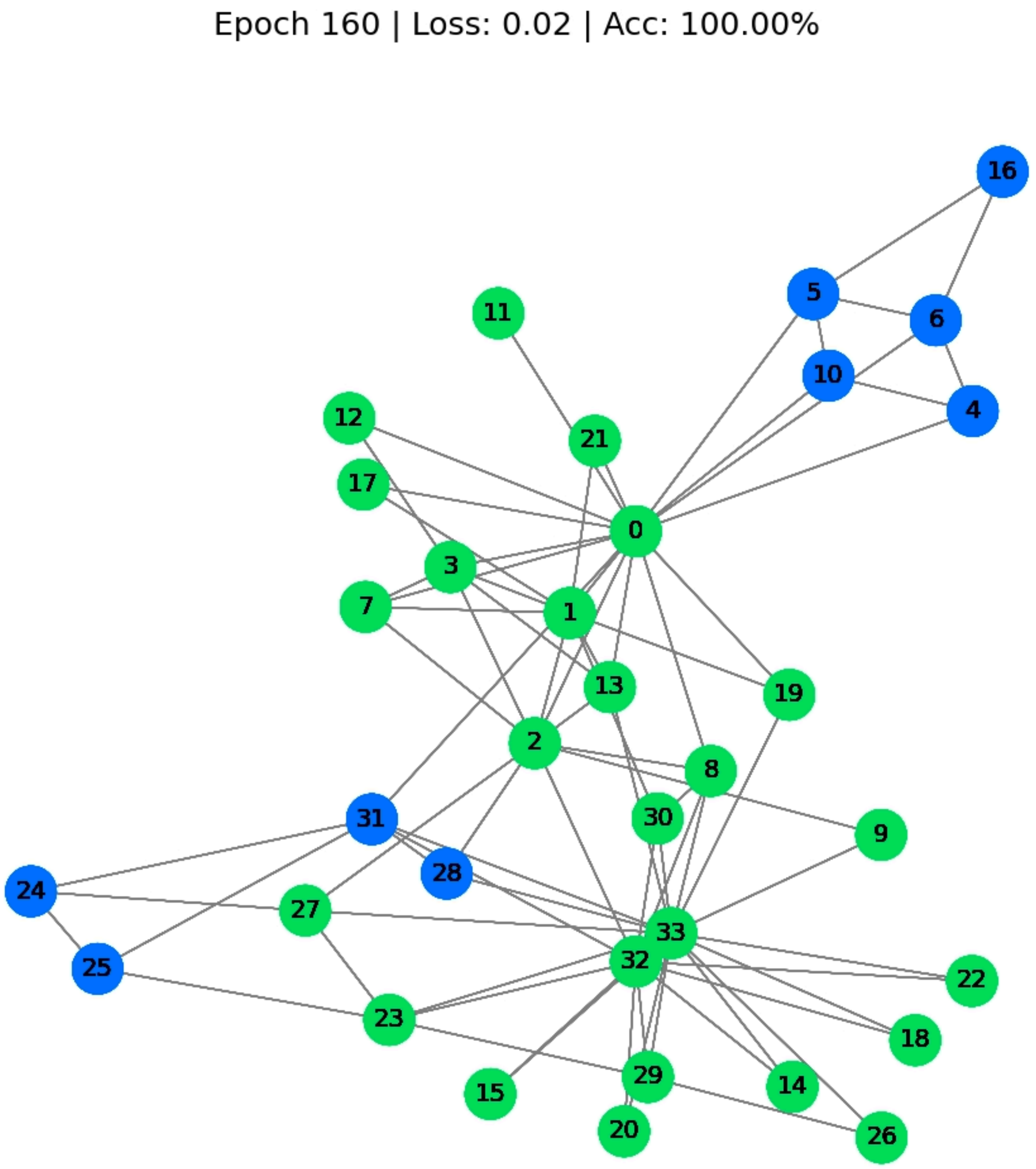
```
%%capture
from IPython.display import HTML
from matplotlib import animation
plt.rcParams["animation.bitrate"] = 3000

def animate(i):
    G = to_networkx(data, to_undirected=True)
    nx.draw_networkx(G,
                    pos=nx.spring_layout(G, seed=0),
                    with_labels=True,
                    node_size=800,
                    node_color=outputs[i],
                    cmap="hsv",
                    vmin=-2,
                    vmax=3,
                    width=0.8,
                    edge_color="grey",
                    font_size=14
                    )
    plt.title(f'Epoch {i} | Loss: {losses[i]:.2f} | Acc: {accuracies[i]*100:.2f}%',
              fontsize=18, pad=20)

fig = plt.figure(figsize=(12, 12))
plt.axis('off')

anim = animation.FuncAnimation(fig, animate, \
            np.arange(0, 200, 10), interval=500, repeat=True)
html = HTML(anim.to_html5_video())
```

```
display(html)
```


Epoch 160 | Loss: 0.02 | Acc: 100.00%

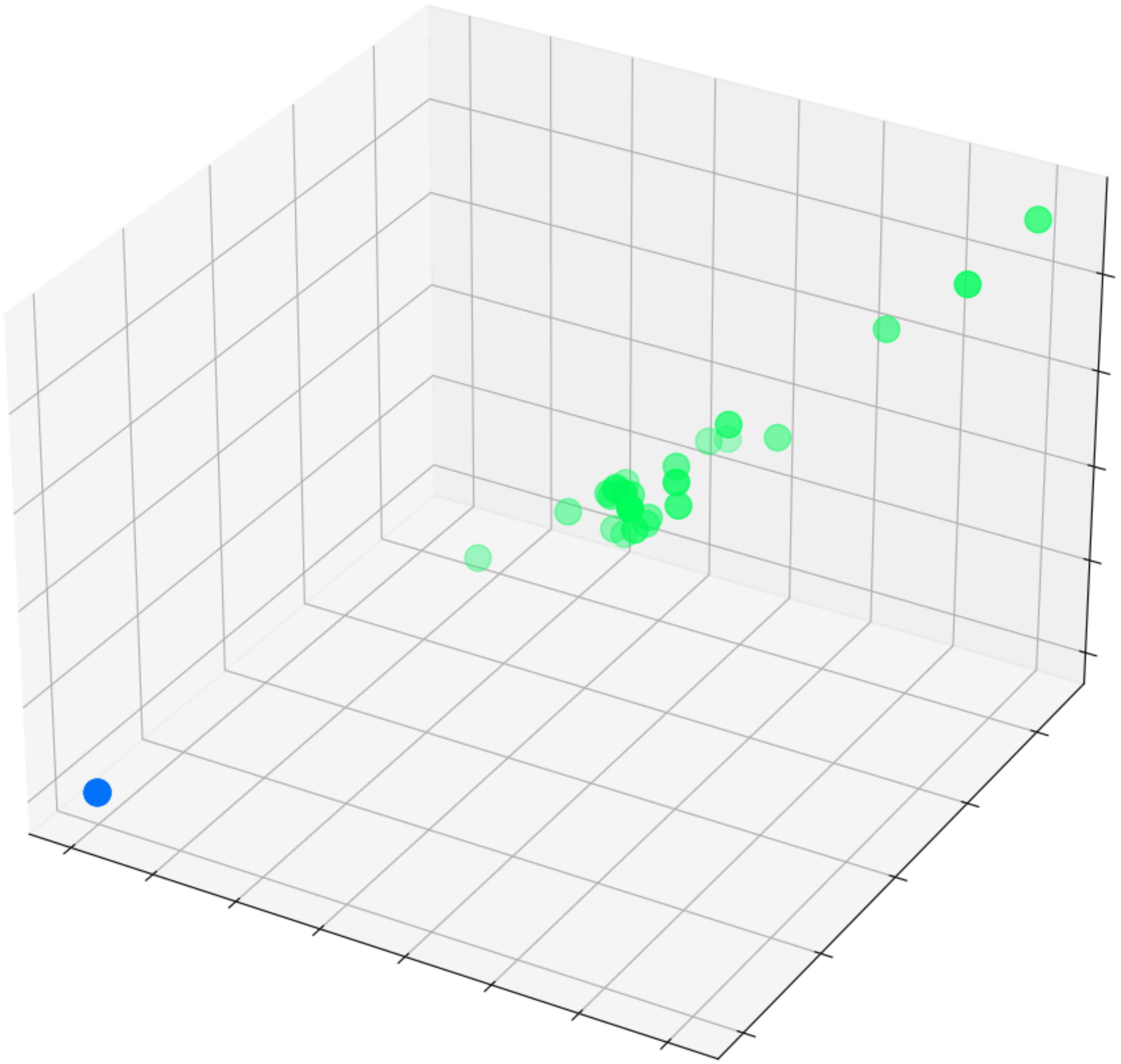
```
# Print embeddings
print(f'Final embeddings = {h.shape}')
print(h)
```

```
Final embeddings = torch.Size([34, 3])
tensor([[1.1333e+00, 1.4626e+00, 1.3707e+00],
        [1.4843e+00, 1.8333e+00, 1.8665e+00],
        [1.1667e+00, 1.5140e+00, 1.3628e+00],
        [1.3009e+00, 1.5372e+00, 1.4209e+00],
        [0.0000e+00, 0.0000e+00, 0.0000e+00],
        [0.0000e+00, 0.0000e+00, 0.0000e+00],
        [0.0000e+00, 0.0000e+00, 0.0000e+00],
        [1.0691e+00, 1.2007e+00, 1.0936e+00],
        [9.7018e-01, 1.1704e+00, 1.2228e+00],
        [1.0104e+00, 1.1877e+00, 1.0188e+00],
        [0.0000e+00, 0.0000e+00, 0.0000e+00],
        [9.8309e-01, 1.1927e+00, 1.2436e+00],
        [1.0718e+00, 1.1198e+00, 1.1209e+00],
        [1.0727e+00, 1.2098e+00, 1.1216e+00],
        [9.8075e-01, 1.1929e+00, 1.2325e+00],
        [1.0226e+00, 1.2038e+00, 1.2188e+00],
        [0.0000e+00, 0.0000e+00, 0.0000e+00],
        [9.7687e-01, 1.1752e+00, 1.2088e+00],
        [1.0870e+00, 1.0552e+00, 1.2852e+00],
        [9.8910e-01, 1.1672e+00, 1.0503e+00],
        [1.0405e+00, 1.1162e+00, 1.2975e+00],
        [1.1571e+00, 1.2123e+00, 1.2237e+00],
        [9.7958e-01, 1.2630e+00, 1.2176e+00],
        [8.7760e-01, 1.1146e+00, 1.1200e+00],
        [0.0000e+00, 0.0000e+00, 3.8445e-04],
        [0.0000e+00, 0.0000e+00, 2.1070e-05],
        [1.1460e+00, 1.2225e+00, 1.3317e+00],
        [6.9420e-01, 9.3486e-01, 9.1289e-01],
        [1.8612e-04, 0.0000e+00, 0.0000e+00],
        [1.2121e+00, 1.4128e+00, 1.5301e+00],
        [1.1057e+00, 1.3121e+00, 1.3307e+00],
        [0.0000e+00, 0.0000e+00, 0.0000e+00],
        [1.6658e+00, 1.9465e+00, 2.1058e+00],
        [1.7630e+00, 2.1907e+00, 2.3209e+00]], grad_fn=<ReluBackward0>)
```

```
# Get first embedding at epoch = 0
embed = h.detach().cpu().numpy()

fig = plt.figure(figsize=(12, 12))
ax = fig.add_subplot(projection='3d')
ax.patch.set_alpha(0)
plt.tick_params(left=False,
                bottom=False,
                labelleft=False,
                labelbottom=False)
ax.scatter(embed[:, 0], embed[:, 1], embed[:, 2],
           s=200, c=data.y, cmap="hsv", vmin=-2, vmax=3)
```

```
plt.show()
```



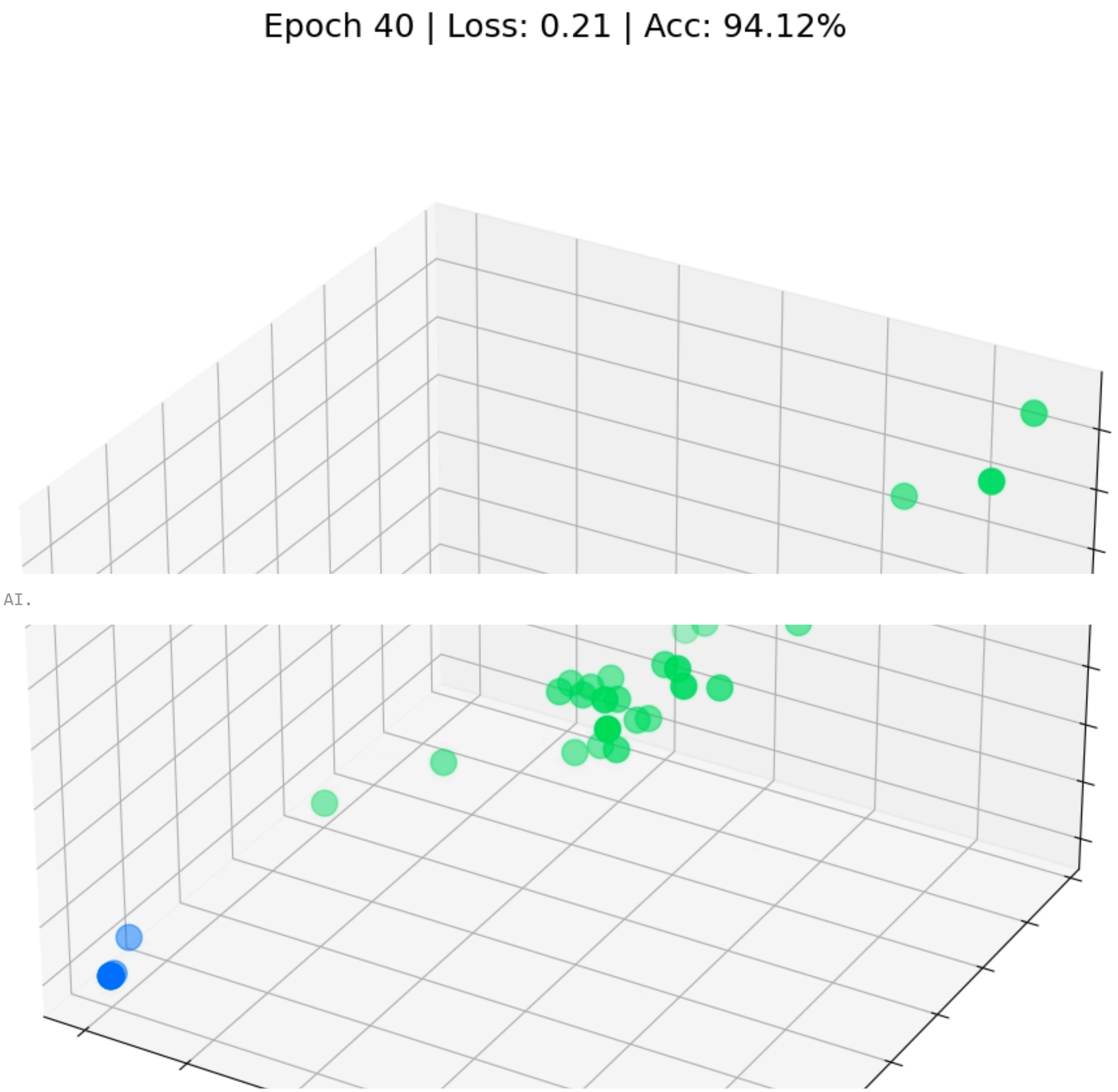
```
%%capture

def animate(i):
    embed = embeddings[i].detach().cpu().numpy()
    ax.clear()
    ax.scatter(embed[:, 0], embed[:, 1], embed[:, 2],
           s=200, c=data.y, cmap="hsv", vmin=-2, vmax=3)
    plt.title(f'Epoch {i} | Loss: {losses[i]:.2f} | Acc: {accuracies[i]*100:.2f}%',
              fontsize=18, pad=40)

fig = plt.figure(figsize=(12, 12))
plt.axis('off')
ax = fig.add_subplot(projection='3d')
plt.tick_params(left=False,
                bottom=False,
                labelleft=False,
                labelbottom=False)

anim = animation.FuncAnimation(fig, animate, \
              np.arange(0, 200, 10), interval=800, repeat=True)
html = HTML(anim.to_html5_video())


display(html)
```



Start coding or generate with AI.