Colab1

September 22, 2021

1 CS224W - Colab 1

In this Colab, we will write a full pipeline for **learning node embeddings**. We will go through the following 3 steps.

To start, we will load a classic graph in network science, the Karate Club Network. We will explore multiple graph statistics for that graph.

We will then work together to transform the graph structure into a PyTorch tensor, so that we can perform machine learning over the graph.

Finally, we will finish the first learning algorithm on graphs: a node embedding model. For simplicity, our model here is simpler than DeepWalk / node2vec algorithms taught in the lecture. But it's still rewarding and challenging, as we will write it from scratch via PyTorch.

Now let's get started!

Note: Make sure to **sequentially run all the cells**, so that the intermediate variables / packages will carry over to the next cell

2 1 Graph Basics

To start, we will load a classic graph in network science, the Karate Club Network. We will explore multiple graph statistics for that graph.

2.1 Setup

We will heavily use NetworkX in this Colab.

[1]:
```
import networkx as nx
```

2.2 Zachary's karate club network

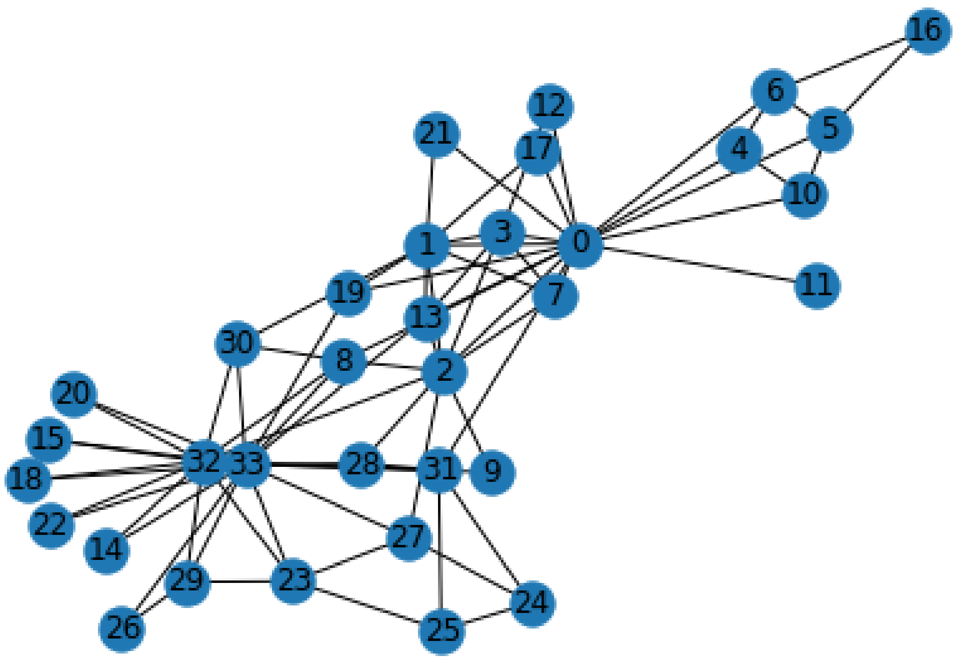
The Karate Club Network is a graph describes a social network of 34 members of a karate club and documents links between members who interacted outside the club.

[2]:
```
G = nx.karate_club_graph()

# G is an undirected graph
type(G)
```

[2]: networkx.classes.graph.Graph

[3]:
```
# Visualize the graph
nx.draw(G, with_labels = True)
```



2.3 Question 1: What is the average degree of the karate club network? (5 Points)

[4]:
```
def average_degree(num_edges, num_nodes):
  # TODO: Implement this function that takes number of edges
  # and number of nodes, and returns the average node degree of
  # the graph. Round the result to nearest integer (for example
  # 3.3 will be rounded to 3 and 3.7 will be rounded to 4)

  avg_degree = 0

  ############# Your code here ############

  avg_degree = round(num_edges/num_nodes)

  #########################################

  return avg_degree
```

```python
num_edges = G.number_of_edges()
num_nodes = G.number_of_nodes()
avg_degree = average_degree(num_edges, num_nodes)
print("Average degree of karate club network is {}".format(avg_degree))
```

```
Average degree of karate club network is 2
```

2.4 Question 2: What is the average clustering coefficient of the karate club network? (5 Points)

[5]:
```python
def average_clustering_coefficient(G):
  # TODO: Implement this function that takes a nx.Graph
  # and returns the average clustering coefficient. Round
  # the result to 2 decimal places (for example 3.333 will
  # be rounded to 3.33 and 3.7571 will be rounded to 3.76)

  avg_cluster_coef = 0

  ############# Your code here ############
  ## Note:
  ## 1: Please use the appropriate NetworkX clustering function

  avg_cluster_coef = round(nx.average_clustering(G),2)

  #########################################

  return avg_cluster_coef

avg_cluster_coef = average_clustering_coefficient(G)
print("Average clustering coefficient of karate club network is {}".
 ↪format(avg_cluster_coef))
```

```
Average clustering coefficient of karate club network is 0.57
```

2.5 Question 3: What is the PageRank value for node 0 (node with id 0) after one PageRank iteration? (5 Points)

Please complete the code block by implementing the PageRank equation: $r_j = \sum_{i \to j} \beta \frac{r_i}{d_i} + (1-\beta)\frac{1}{N}$

[6]:
```python
def one_iter_pagerank(G, beta, r0, node_id):
  # TODO: Implement this function that takes a nx.Graph, beta, r0 and node id.
  # The return value r1 is one interation PageRank value for the input node.
  # Please round r1 to 2 decimal places.

  r1 = 0
```

```
  ############# Your code here ############
  ## Note:
  ## 1: You should not use nx.pagerank

  for neighbor in G.neighbors(node_id):
    r1 += beta*r0/G.degree[neighbor]
  r1 += (1-beta)/G.number_of_nodes()
  r1 = round(r1,2)

  #########################################

  return r1

beta = 0.8
r0 = 1 / G.number_of_nodes()
node = 0
r1 = one_iter_pagerank(G, beta, r0, node)
print("The PageRank value for node 0 after one iteration is {}".format(r1))
```

```
The PageRank value for node 0 after one iteration is 0.13
```

2.6 Question 4: What is the (raw) closeness centrality for the karate club network node 5? (5 Points)

The equation for closeness centrality is $c(v) = \frac{1}{\sum_{u \neq v} \text{shortest path length between } u \text{ and } v}$

[7]:
```
def closeness_centrality(G, node=5):
  # TODO: Implement the function that calculates closeness centrality
  # for a node in karate club network. G is the input karate club
  # network and node is the node id in the graph. Please round the
  # closeness centrality result to 2 decimal places.

  closeness = 0

  ## Note:
  ## 1: You can use networkx closeness centrality function.
  ## 2: Notice that networkx closeness centrality returns the normalized
  ## closeness directly, which is different from the raw (unnormalized)
  ## one that we learned in the lecture.

  closeness = round(nx.closeness_centrality(G,node), 2)

  #########################################

  return closeness

node = 5
```

```
closeness = closeness_centrality(G, node=node)
print("The karate club network has closeness centrality {}".format(closeness))
```

```
The karate club network has closeness centrality 0.38
```

3 2 Graph to Tensor

We will then work together to transform the graph G into a PyTorch tensor, so that we can perform machine learning over the graph.

3.1 Setup

Check if PyTorch is properly installed

[8]:
```
import torch
print(torch.__version__)
```

```
1.9.0+cu102
```

3.2 PyTorch tensor basics

We can generate PyTorch tensor with all zeros, ones or random values.

[9]:
```
# Generate 3 x 4 tensor with all ones
ones = torch.ones(3, 4)
print(ones)

# Generate 3 x 4 tensor with all zeros
zeros = torch.zeros(3, 4)
print(zeros)

# Generate 3 x 4 tensor with random values on the interval [0, 1)
random_tensor = torch.rand(3, 4)
print(random_tensor)

# Get the shape of the tensor
print(ones.shape)
```

```
tensor([[1., 1., 1., 1.],
        [1., 1., 1., 1.],
        [1., 1., 1., 1.]])
tensor([[0., 0., 0., 0.],
        [0., 0., 0., 0.],
        [0., 0., 0., 0.]])
tensor([[0.5903, 0.1552, 0.1427, 0.5153],
        [0.7826, 0.3062, 0.0978, 0.4393],
        [0.1320, 0.6661, 0.8469, 0.6831]])
torch.Size([3, 4])
```

PyTorch tensor contains elements for a single data type, the `dtype`.

[10]:
```
# Create a 3 x 4 tensor with all 32-bit floating point zeros
zeros = torch.zeros(3, 4, dtype=torch.float32)
print(zeros.dtype)

# Change the tensor dtype to 64-bit integer
zeros = zeros.type(torch.long)
print(zeros.dtype)
```

```
torch.float32
torch.int64
```

3.3 Question 5: Getting the edge list of the karate club network and transform it into `torch.LongTensor`. What is the `torch.sum` value of `pos_edge_index` tensor? (10 Points)

[11]:
```
def graph_to_edge_list(G):
  # TODO: Implement the function that returns the edge list of
  # an nx.Graph. The returned edge_list should be a list of tuples
  # where each tuple is a tuple representing an edge connected
  # by two nodes.

  edge_list = []

  ############# Your code here ############

  edge_list = list(G.edges())

  #########################################

  return edge_list

def edge_list_to_tensor(edge_list):
  # TODO: Implement the function that transforms the edge_list to
  # tensor. The input edge_list is a list of tuples and the resulting
  # tensor should have the shape [2 x len(edge_list)].

  edge_index = torch.LongTensor([])

  ############# Your code here ############

  edge_index = torch.LongTensor(edge_list).T

  #########################################

  return edge_index
```

```
pos_edge_list = graph_to_edge_list(G)
pos_edge_index = edge_list_to_tensor(pos_edge_list)
print("The pos_edge_index tensor has shape {}".format(pos_edge_index.shape))
print("The pos_edge_index tensor has sum value {}".format(torch.
 ↪sum(pos_edge_index)))
```

```
The pos_edge_index tensor has shape torch.Size([2, 78])
The pos_edge_index tensor has sum value 2535
```

3.4 Question 6: Please implement following function that samples negative edges. Then you will answer which edges (edge_1 to edge_5) can be negative ones in the karate club network? (10 Points)

[12]:
```
import random

def sample_negative_edges(G, num_neg_samples):
  # TODO: Implement the function that returns a list of negative edges.
  # The number of sampled negative edges is num_neg_samples. You do not
  # need to consider the corner case when the number of possible negative edges
  # is less than num_neg_samples. It should be ok as long as your␣
 ↪implementation
  # works on the karate club network. In this implementation, self loop should
  # not be considered as either a positive or negative edge. Also, notice that
  # the karate club network is an undirected graph, if (0, 1) is a positive
  # edge, do you think (1, 0) can be a negative one?

  neg_edge_list = []

  ############# Your code here ############

  pos_edge_list = graph_to_edge_list(G)
  for i in G.nodes():
    for j in G.nodes():
      if i >= j or (i,j) in pos_edge_list:
        continue
      neg_edge_list.append((i,j))
  neg_edge_list = random.sample(neg_edge_list,num_neg_samples)

  #########################################

  return neg_edge_list

# Sample 78 negative edges
neg_edge_list = sample_negative_edges(G, len(pos_edge_list))

# Transform the negative edge list to tensor
```

```
neg_edge_index = edge_list_to_tensor(neg_edge_list)
print("The neg_edge_index tensor has shape {}".format(neg_edge_index.shape))

# Which of following edges can be negative ones?
edge_1 = (7, 1)
edge_2 = (1, 33)
edge_3 = (33, 22)
edge_4 = (0, 4)
edge_5 = (4, 2)

############# Your code here ############
## Note:
## 1: For each of the 5 edges, print whether it can be negative edge

def neg_edge(edge):
  if edge in pos_edge_list or (edge[1],edge[0]) in pos_edge_list:
    print("no")
  else:
    print("yes")
neg_edge(edge_1)
neg_edge(edge_2)
neg_edge(edge_3)
neg_edge(edge_4)
neg_edge(edge_5)

#########################################
```

```
The neg_edge_index tensor has shape torch.Size([2, 78])
no
yes
no
no
yes
```

4 3 Node Emebedding Learning

Finally, we will finish the first learning algorithm on graphs: a node embedding model.

4.1 Setup

[13]:
```
import torch
import torch.nn as nn
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA

print(torch.__version__)
```

1.9.0+cu102

To write our own node embedding learning methods, we'll heavily use the `nn.Embedding` module in PyTorch. Let's see how to use `nn.Embedding`:

[14]:
```
# Initialize an embedding layer
# Suppose we want to have embedding for 4 items (e.g., nodes)
# Each item is represented with 8 dimensional vector

emb_sample = nn.Embedding(num_embeddings=4, embedding_dim=8)
print('Sample embedding layer: {}'.format(emb_sample))
```

```
Sample embedding layer: Embedding(4, 8)
```

We can select items from the embedding matrix, by using Tensor indices

[15]:
```
# Select an embedding in emb_sample
id = torch.LongTensor([1])
print(emb_sample(id))

# Select multiple embeddings
ids = torch.LongTensor([1, 3])
print(emb_sample(ids))

# Get the shape of the embedding weight matrix
shape = emb_sample.weight.data.shape
print(shape)

# Overwrite the weight to tensor with all ones
emb_sample.weight.data = torch.ones(shape)

# Let's check if the emb is indeed initilized
ids = torch.LongTensor([0, 3])
print(emb_sample(ids))
```

```
tensor([[-0.4852,  0.0553, -0.7192, -0.5815, -0.1160, -0.7977,  1.0060,
1.1413]],
       grad_fn=<EmbeddingBackward>)
tensor([[-0.4852,  0.0553, -0.7192, -0.5815, -0.1160, -0.7977,  1.0060,
1.1413],
        [-1.1993,  0.4037, -0.0466,  1.6998, -0.6454,  0.1823,  0.1226,
-0.1198]],
       grad_fn=<EmbeddingBackward>)
torch.Size([4, 8])
tensor([[1., 1., 1., 1., 1., 1., 1., 1.],
        [1., 1., 1., 1., 1., 1., 1., 1.]], grad_fn=<EmbeddingBackward>)
```

Now, it's your time to create node embedding matrix for the graph we have! - We want to have **16 dimensional** vector for each node in the karate club network. - We want to initalize the matrix under **uniform distribution**, in the range of $[0, 1)$. We suggest you using `torch.rand`.

[16]:
```
# Please do not change / reset the random seed
torch.manual_seed(1)

def create_node_emb(num_node=34, embedding_dim=16):
  # TODO: Implement this function that will create the node embedding matrix.
  # A torch.nn.Embedding layer will be returned. You do not need to change
  # the values of num_node and embedding_dim. The weight matrix of returned
  # layer should be initialized under uniform distribution.

  emb = None

  ############# Your code here ############

  emb = nn.Embedding(num_embeddings=num_nodes, embedding_dim=embedding_dim)
  emb.weight.data = torch.rand(num_nodes,embedding_dim)

  #########################################

  return emb

emb = create_node_emb()
ids = torch.LongTensor([0, 3])

# Print the embedding layer
print("Embedding: {}".format(emb))

# An example that gets the embeddings for node 0 and 3
print(emb(ids))
```

```
Embedding: Embedding(34, 16)
tensor([[0.2114, 0.7335, 0.1433, 0.9647, 0.2933, 0.7951, 0.5170, 0.2801, 0.8339,
         0.1185, 0.2355, 0.5599, 0.8966, 0.2858, 0.1955, 0.1808],
        [0.7486, 0.6546, 0.3843, 0.9820, 0.6012, 0.3710, 0.4929, 0.9915, 0.8358,
         0.4629, 0.9902, 0.7196, 0.2338, 0.0450, 0.7906, 0.9689]],
       grad_fn=<EmbeddingBackward>)
```

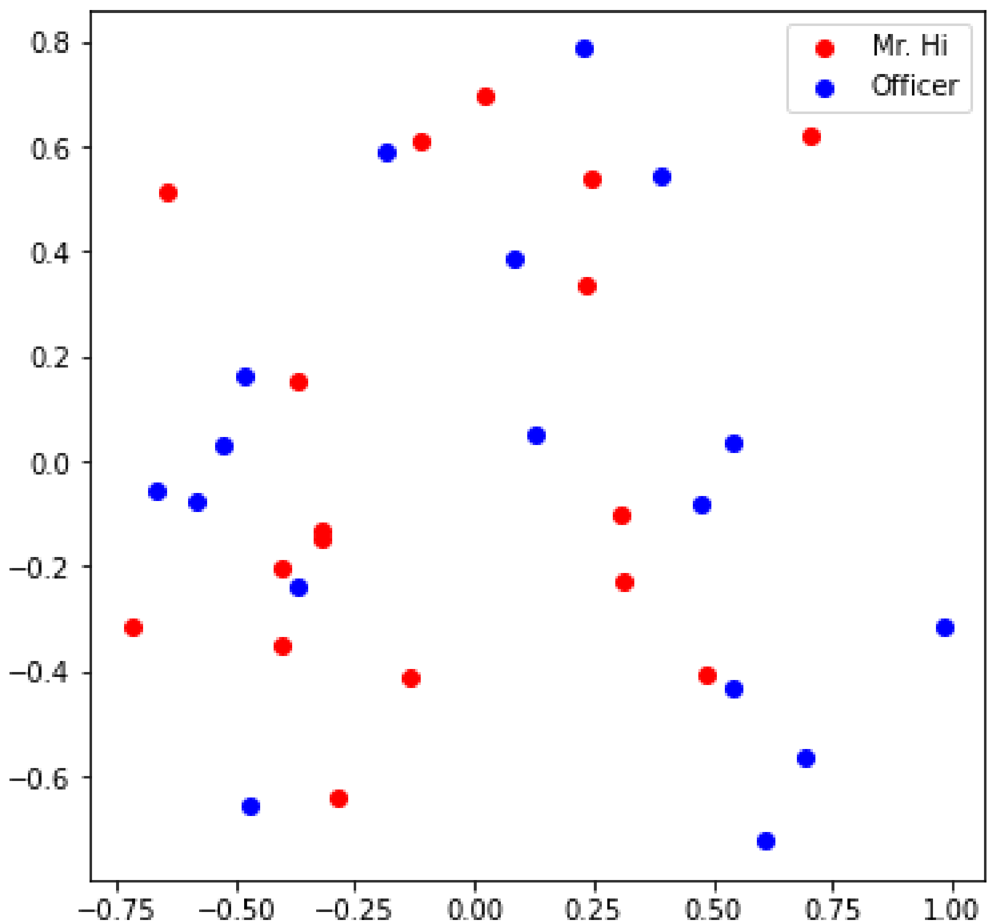
4.2 Visualize the initial node embeddings

One good way to understand an embedding matrix, is to visualize it in a 2D space. Here, we have implemented an embedding visualization function for you. We first do PCA to reduce the dimensionality of embeddings to a 2D space. Then visualize each point, colored by the community it belongs to.

[17]:
```
def visualize_emb(emb):
  X = emb.weight.data.numpy()
  pca = PCA(n_components=2)
  components = pca.fit_transform(X)
  plt.figure(figsize=(6, 6))
```

```
  club1_x = []
  club1_y = []
  club2_x = []
  club2_y = []
  for node in G.nodes(data=True):
    if node[1]['club'] == 'Mr. Hi':
      club1_x.append(components[node[0]][0])
      club1_y.append(components[node[0]][1])
    else:
      club2_x.append(components[node[0]][0])
      club2_y.append(components[node[0]][1])
  plt.scatter(club1_x, club1_y, color="red", label="Mr. Hi")
  plt.scatter(club2_x, club2_y, color="blue", label="Officer")
  plt.legend()
  plt.show()

# Visualize the initial random embeddding
visualize_emb(emb)
```



4.3 Question 7: Training the embedding! What is the best performance you can get? Please report both the best loss and accuracy on Gradescope. (20 Points)

[18]:
```
from torch.optim import SGD


def accuracy(pred, label):
  # TODO: Implement the accuracy function. This function takes the
  # pred tensor (the resulting tensor after sigmoid) and the label
  # tensor (torch.LongTensor). Predicted value greater than 0.5 will
  # be classified as label 1. Else it will be classified as label 0.
  # The returned accuracy should be rounded to 4 decimal places.
  # For example, accuracy 0.82956 will be rounded to 0.8296.

  accu = 0.0

  ############# Your code here ############
  pred = pred.ge(0.5)
  accu = torch.sum(pred==label)/pred.shape[0]
  accu = accu.item()

  #########################################

  return round(accu,4)

def train(emb, loss_fn, sigmoid, train_label, train_edge):
  # TODO: Train the embedding layer here. You can also change epochs and
  # learning rate. In general, you need to implement:
  # (1) Get the embeddings of the nodes in train_edge
  # (2) Dot product the embeddings between each node pair
  # (3) Feed the dot product result into sigmoid
  # (4) Feed the sigmoid output into the loss_fn
  # (5) Print both loss and accuracy of each epoch
  # (as a sanity check, the loss should decrease during training)

  epochs = 500
  learning_rate = 0.1

  optimizer = SGD(emb.parameters(), lr=learning_rate, momentum=0.9)

  for i in range(epochs):

    ############# Your code here ############
    optimizer.zero_grad()
```

```python
    # Get the embedings of the nodes in train_edge
    node_emb0 = emb(train_edge[0])
    node_emb1 = emb(train_edge[1])

    # Dot product the embeddings of the nodes in train_edge
    dotp = torch.sum(node_emb0*node_emb1,1)

    # Feed dot product into sigmoid
    sig = sigmoid(dotp)

    # Feed sig into lossfn
    loss = loss_fn(sig,train_label)

    loss.backward()
    optimizer.step()

    # Print results
    print("Epoch ",i,"Loss ",loss.item(),"Accu ", accuracy(sig,train_label))


    #########################################

loss_fn = nn.BCELoss()
sigmoid = nn.Sigmoid()

# Generate the positive and negative labels
pos_label = torch.ones(pos_edge_index.shape[1], )
neg_label = torch.zeros(neg_edge_index.shape[1], )

# Concat positive and negative labels into one tensor
train_label = torch.cat([pos_label, neg_label], dim=0)

# Concat positive and negative edges into one tensor
# Since the network is very small, we do not split the edges into val/test sets
train_edge = torch.cat([pos_edge_index, neg_edge_index], dim=1)

train(emb, loss_fn, sigmoid, train_label, train_edge)
```

```
Epoch  0 Loss  2.0520997047424316 Accu  0.5
Epoch  1 Loss  2.0382442474365234 Accu  0.5
Epoch  2 Loss  2.0121307373046875 Accu  0.5
Epoch  3 Loss  1.9753713607788086 Accu  0.5
Epoch  4 Loss  1.9295581579208374 Accu  0.5
Epoch  5 Loss  1.8762335777282715 Accu  0.5
Epoch  6 Loss  1.8168731927871704 Accu  0.5
Epoch  7 Loss  1.7528722286224365 Accu  0.5
```

```
Epoch  8 Loss  1.6855337619781494 Accu  0.5
Epoch  9 Loss  1.6160619258880615 Accu  0.5
Epoch  10 Loss  1.5455574989318848 Accu  0.5
Epoch  11 Loss  1.4750118255615234 Accu  0.5
Epoch  12 Loss  1.4053034782409668 Accu  0.5
Epoch  13 Loss  1.3371955156326294 Accu  0.5
Epoch  14 Loss  1.271332025527954 Accu  0.5
Epoch  15 Loss  1.2082380056381226 Accu  0.5
Epoch  16 Loss  1.1483200788497925 Accu  0.5
Epoch  17 Loss  1.0918701887130737 Accu  0.5
Epoch  18 Loss  1.0390719175338745 Accu  0.5
Epoch  19 Loss  0.990010678768158 Accu  0.5
Epoch  20 Loss  0.9446856379508972 Accu  0.5
Epoch  21 Loss  0.9030229449272156 Accu  0.5064
Epoch  22 Loss  0.8648912310600281 Accu  0.5064
Epoch  23 Loss  0.8301149010658264 Accu  0.5064
Epoch  24 Loss  0.798488199710846 Accu  0.5192
Epoch  25 Loss  0.7697864174842834 Accu  0.5449
Epoch  26 Loss  0.7437759041786194 Accu  0.5577
Epoch  27 Loss  0.7202226519584656 Accu  0.5641
Epoch  28 Loss  0.6988973617553711 Accu  0.5705
Epoch  29 Loss  0.679580807685852 Accu  0.5833
Epoch  30 Loss  0.6620659828186035 Accu  0.5897
Epoch  31 Loss  0.6461609601974487 Accu  0.5962
Epoch  32 Loss  0.6316890120506287 Accu  0.609
Epoch  33 Loss  0.6184890270233154 Accu  0.6218
Epoch  34 Loss  0.6064154505729675 Accu  0.641
Epoch  35 Loss  0.5953372120857239 Accu  0.6603
Epoch  36 Loss  0.5851373076438904 Accu  0.6667
Epoch  37 Loss  0.5757114887237549 Accu  0.6667
Epoch  38 Loss  0.566967248916626 Accu  0.6667
Epoch  39 Loss  0.5588229298591614 Accu  0.6603
Epoch  40 Loss  0.5512060523033142 Accu  0.6731
Epoch  41 Loss  0.5440533757209778 Accu  0.6795
Epoch  42 Loss  0.5373089909553528 Accu  0.6859
Epoch  43 Loss  0.5309237837791443 Accu  0.7051
Epoch  44 Loss  0.5248546600341797 Accu  0.7179
Epoch  45 Loss  0.5190641283988953 Accu  0.7179
Epoch  46 Loss  0.5135190486907959 Accu  0.75
Epoch  47 Loss  0.5081906914710999 Accu  0.7564
Epoch  48 Loss  0.503053605556488 Accu  0.7628
Epoch  49 Loss  0.49808579683303833 Accu  0.7949
Epoch  50 Loss  0.4932679831981659 Accu  0.8141
Epoch  51 Loss  0.4885829985141754 Accu  0.8269
Epoch  52 Loss  0.48401620984077454 Accu  0.8333
Epoch  53 Loss  0.479554682970047 Accu  0.8333
Epoch  54 Loss  0.4751869738101959 Accu  0.8397
Epoch  55 Loss  0.4709031581878662 Accu  0.8462
```

```
Epoch  56 Loss  0.4666946232318878 Accu  0.8462
Epoch  57 Loss  0.46255356073379517 Accu  0.8526
Epoch  58 Loss  0.45847341418266296 Accu  0.8526
Epoch  59 Loss  0.45444825291633606 Accu  0.8654
Epoch  60 Loss  0.45047301054000854 Accu  0.8782
Epoch  61 Loss  0.44654303789138794 Accu  0.8782
Epoch  62 Loss  0.44265449047088623 Accu  0.8782
Epoch  63 Loss  0.4388037919998169 Accu  0.8782
Epoch  64 Loss  0.4349880814552307 Accu  0.8846
Epoch  65 Loss  0.4312044382095337 Accu  0.8846
Epoch  66 Loss  0.427450567483902 Accu  0.8846
Epoch  67 Loss  0.42372459173202515 Accu  0.8974
Epoch  68 Loss  0.42002448439598083 Accu  0.8974
Epoch  69 Loss  0.41634881496429443 Accu  0.9038
Epoch  70 Loss  0.4126962125301361 Accu  0.9038
Epoch  71 Loss  0.4090653955936432 Accu  0.9103
Epoch  72 Loss  0.4054553806781769 Accu  0.9103
Epoch  73 Loss  0.40186530351638794 Accu  0.9103
Epoch  74 Loss  0.3982943892478943 Accu  0.9231
Epoch  75 Loss  0.3947419226169586 Accu  0.9231
Epoch  76 Loss  0.3912074863910675 Accu  0.9231
Epoch  77 Loss  0.38769054412841797 Accu  0.9231
Epoch  78 Loss  0.38419073820114136 Accu  0.9295
Epoch  79 Loss  0.3807078003883362 Accu  0.9359
Epoch  80 Loss  0.37724146246910095 Accu  0.9359
Epoch  81 Loss  0.37379157543182373 Accu  0.9359
Epoch  82 Loss  0.37035802006721497 Accu  0.9359
Epoch  83 Loss  0.36694082617759705 Accu  0.9359
Epoch  84 Loss  0.3635398745536804 Accu  0.9359
Epoch  85 Loss  0.36015522480010986 Accu  0.9359
Epoch  86 Loss  0.35678690671920776 Accu  0.9423
Epoch  87 Loss  0.35343503952026367 Accu  0.9423
Epoch  88 Loss  0.3500998020172119 Accu  0.9423
Epoch  89 Loss  0.34678128361701965 Accu  0.9423
Epoch  90 Loss  0.3434796631336212 Accu  0.9423
Epoch  91 Loss  0.3401951789855957 Accu  0.9487
Epoch  92 Loss  0.33692803978919983 Accu  0.9551
Epoch  93 Loss  0.3336784839630127 Accu  0.9615
Epoch  94 Loss  0.33044669032096863 Accu  0.9615
Epoch  95 Loss  0.3272330164909363 Accu  0.9679
Epoch  96 Loss  0.32403764128685 Accu  0.9679
Epoch  97 Loss  0.32086095213890076 Accu  0.9679
Epoch  98 Loss  0.31770309805870056 Accu  0.9679
Epoch  99 Loss  0.3145644962787628 Accu  0.9679
Epoch  100 Loss  0.3114454448223114 Accu  0.9679
Epoch  101 Loss  0.30834606289863586 Accu  0.9679
Epoch  102 Loss  0.3052668571472168 Accu  0.9679
Epoch  103 Loss  0.30220794677734375 Accu  0.9679
```

```
Epoch  104 Loss  0.29916974902153015 Accu  0.9679
Epoch  105 Loss  0.2961525022983551 Accu  0.9679
Epoch  106 Loss  0.2931564450263977 Accu  0.9679
Epoch  107 Loss  0.2901819348335266 Accu  0.9679
Epoch  108 Loss  0.2872292101383209 Accu  0.9679
Epoch  109 Loss  0.2842985689640045 Accu  0.9679
Epoch  110 Loss  0.28139013051986694 Accu  0.9679
Epoch  111 Loss  0.27850425243377686 Accu  0.9679
Epoch  112 Loss  0.27564120292663574 Accu  0.9679
Epoch  113 Loss  0.27280113101005554 Accu  0.9679
Epoch  114 Loss  0.26998427510261536 Accu  0.9679
Epoch  115 Loss  0.2671908438205719 Accu  0.9744
Epoch  116 Loss  0.2644209861755371 Accu  0.9744
Epoch  117 Loss  0.2616749405860901 Accu  0.9808
Epoch  118 Loss  0.2589527666568756 Accu  0.9808
Epoch  119 Loss  0.25625479221343994 Accu  0.9808
Epoch  120 Loss  0.2535809874534607 Accu  0.9808
Epoch  121 Loss  0.2509315311908722 Accu  0.9808
Epoch  122 Loss  0.24830657243728638 Accu  0.9808
Epoch  123 Loss  0.24570611119270325 Accu  0.9808
Epoch  124 Loss  0.2431303858757019 Accu  0.9808
Epoch  125 Loss  0.24057930707931519 Accu  0.9808
Epoch  126 Loss  0.2380530834197998 Accu  0.9872
Epoch  127 Loss  0.2355516403913498 Accu  0.9872
Epoch  128 Loss  0.23307503759860992 Accu  0.9872
Epoch  129 Loss  0.23062331974506378 Accu  0.9872
Epoch  130 Loss  0.22819648683071136 Accu  0.9872
Epoch  131 Loss  0.22579452395439148 Accu  0.9872
Epoch  132 Loss  0.22341738641262054 Accu  0.9872
Epoch  133 Loss  0.22106510400772095 Accu  0.9872
Epoch  134 Loss  0.2187376320362091 Accu  0.9872
Epoch  135 Loss  0.21643482148647308 Accu  0.9872
Epoch  136 Loss  0.21415671706199646 Accu  0.9872
Epoch  137 Loss  0.21190319955348969 Accu  0.9936
Epoch  138 Loss  0.2096741944551468 Accu  0.9936
Epoch  139 Loss  0.20746959745883942 Accu  0.9936
Epoch  140 Loss  0.2052893489599228 Accu  0.9936
Epoch  141 Loss  0.20313327014446259 Accu  0.9936
Epoch  142 Loss  0.20100131630897522 Accu  0.9936
Epoch  143 Loss  0.19889327883720398 Accu  0.9936
Epoch  144 Loss  0.19680911302566528 Accu  0.9936
Epoch  145 Loss  0.1947486251592636 Accu  0.9936
Epoch  146 Loss  0.19271166622638702 Accu  0.9936
Epoch  147 Loss  0.19069808721542358 Accu  0.9936
Epoch  148 Loss  0.18870776891708374 Accu  0.9936
Epoch  149 Loss  0.18674048781394958 Accu  0.9936
Epoch  150 Loss  0.18479610979557037 Accu  0.9936
Epoch  151 Loss  0.18287447094917297 Accu  0.9936
```

```
Epoch  152 Loss  0.18097534775733948 Accu  0.9936
Epoch  153 Loss  0.17909860610961914 Accu  0.9936
Epoch  154 Loss  0.17724402248859406 Accu  0.9936
Epoch  155 Loss  0.1754114329814911 Accu  0.9936
Epoch  156 Loss  0.17360062897205353 Accu  0.9936
Epoch  157 Loss  0.17181141674518585 Accu  0.9936
Epoch  158 Loss  0.17004358768463135 Accu  0.9936
Epoch  159 Loss  0.1682969629764557 Accu  0.9936
Epoch  160 Loss  0.16657131910324097 Accu  0.9936
Epoch  161 Loss  0.16486649215221405 Accu  0.9936
Epoch  162 Loss  0.16318224370479584 Accu  0.9936
Epoch  163 Loss  0.16151836514472961 Accu  0.9936
Epoch  164 Loss  0.15987463295459747 Accu  0.9936
Epoch  165 Loss  0.15825089812278748 Accu  0.9936
Epoch  166 Loss  0.15664689242839813 Accu  0.9936
Epoch  167 Loss  0.15506243705749512 Accu  0.9936
Epoch  168 Loss  0.15349730849266052 Accu  0.9936
Epoch  169 Loss  0.15195131301879883 Accu  0.9936
Epoch  170 Loss  0.15042419731616974 Accu  0.9936
Epoch  171 Loss  0.14891579747200012 Accu  0.9936
Epoch  172 Loss  0.14742591977119446 Accu  0.9936
Epoch  173 Loss  0.14595429599285126 Accu  0.9936
Epoch  174 Loss  0.1445007473230362 Accu  0.9936
Epoch  175 Loss  0.14306508004665375 Accu  0.9936
Epoch  176 Loss  0.14164705574512482 Accu  0.9936
Epoch  177 Loss  0.14024649560451508 Accu  0.9936
Epoch  178 Loss  0.13886317610740662 Accu  0.9936
Epoch  179 Loss  0.1374969184398651 Accu  0.9936
Epoch  180 Loss  0.13614746928215027 Accu  0.9936
Epoch  181 Loss  0.13481470942497253 Accu  0.9936
Epoch  182 Loss  0.13349835574626923 Accu  0.9936
Epoch  183 Loss  0.1321982443332672 Accu  0.9936
Epoch  184 Loss  0.13091418147087097 Accu  0.9936
Epoch  185 Loss  0.1296459585428238 Accu  0.9936
Epoch  186 Loss  0.12839341163635254 Accu  0.9936
Epoch  187 Loss  0.12715628743171692 Accu  0.9936
Epoch  188 Loss  0.125934436917305 Accu  0.9936
Epoch  189 Loss  0.12472768872976303 Accu  0.9936
Epoch  190 Loss  0.12353578954935074 Accu  0.9936
Epoch  191 Loss  0.12235862761735916 Accu  0.9936
Epoch  192 Loss  0.12119598686695099 Accu  0.9936
Epoch  193 Loss  0.1200476661324501 Accu  1.0
Epoch  194 Loss  0.11891347914934158 Accu  1.0
Epoch  195 Loss  0.11779329180717468 Accu  1.0
Epoch  196 Loss  0.1166868805885315 Accu  1.0
Epoch  197 Loss  0.11559410393238068 Accu  1.0
Epoch  198 Loss  0.11451476812362671 Accu  1.0
Epoch  199 Loss  0.11344867944717407 Accu  1.0
```

```
Epoch  200 Loss  0.11239570379257202 Accu  1.0
Epoch  201 Loss  0.11135566979646683 Accu  1.0
Epoch  202 Loss  0.11032838374376297 Accu  1.0
Epoch  203 Loss  0.10931369662284851 Accu  1.0
Epoch  204 Loss  0.10831145942211151 Accu  1.0
Epoch  205 Loss  0.10732148587703705 Accu  1.0
Epoch  206 Loss  0.1063435971736908 Accu  1.0
Epoch  207 Loss  0.105377696454525 Accu  1.0
Epoch  208 Loss  0.10442358255386353 Accu  1.0
Epoch  209 Loss  0.10348110646009445 Accu  1.0
Epoch  210 Loss  0.10255011171102524 Accu  1.0
Epoch  211 Loss  0.10163049399852753 Accu  1.0
Epoch  212 Loss  0.10072202235460281 Accu  1.0
Epoch  213 Loss  0.09982461482286453 Accu  1.0
Epoch  214 Loss  0.09893808513879776 Accu  1.0
Epoch  215 Loss  0.09806234389543533 Accu  1.0
Epoch  216 Loss  0.09719718992710114 Accu  1.0
Epoch  217 Loss  0.09634249657392502 Accu  1.0
Epoch  218 Loss  0.09549814462661743 Accu  1.0
Epoch  219 Loss  0.09466397762298584 Accu  1.0
Epoch  220 Loss  0.09383987635374069 Accu  1.0
Epoch  221 Loss  0.09302569925785065 Accu  1.0
Epoch  222 Loss  0.09222131222486496 Accu  1.0
Epoch  223 Loss  0.09142657369375229 Accu  1.0
Epoch  224 Loss  0.09064137935638428 Accu  1.0
Epoch  225 Loss  0.08986559510231018 Accu  1.0
Epoch  226 Loss  0.08909907937049866 Accu  1.0
Epoch  227 Loss  0.08834172785282135 Accu  1.0
Epoch  228 Loss  0.08759340643882751 Accu  1.0
Epoch  229 Loss  0.0868539959192276 Accu  1.0
Epoch  230 Loss  0.08612337708473206 Accu  1.0
Epoch  231 Loss  0.08540144562721252 Accu  1.0
Epoch  232 Loss  0.08468805253505707 Accu  1.0
Epoch  233 Loss  0.08398311585187912 Accu  1.0
Epoch  234 Loss  0.08328651636838913 Accu  1.0
Epoch  235 Loss  0.08259812742471695 Accu  1.0
Epoch  236 Loss  0.08191784471273422 Accu  1.0
Epoch  237 Loss  0.081245556473732 Accu  1.0
Epoch  238 Loss  0.0805811882019043 Accu  1.0
Epoch  239 Loss  0.07992459088563919 Accu  1.0
Epoch  240 Loss  0.07927568256855011 Accu  1.0
Epoch  241 Loss  0.0786343440413475 Accu  1.0
Epoch  242 Loss  0.07800048589706421 Accu  1.0
Epoch  243 Loss  0.07737400382757187 Accu  1.0
Epoch  244 Loss  0.07675480842590332 Accu  1.0
Epoch  245 Loss  0.07614278793334961 Accu  1.0
Epoch  246 Loss  0.07553786039352417 Accu  1.0
Epoch  247 Loss  0.07493990659713745 Accu  1.0
```

```
Epoch  248 Loss  0.07434885948896408 Accu  1.0
Epoch  249 Loss  0.0737646147608757 Accu  1.0
Epoch  250 Loss  0.07318708300590515 Accu  1.0
Epoch  251 Loss  0.07261615246534348 Accu  1.0
Epoch  252 Loss  0.07205177843570709 Accu  1.0
Epoch  253 Loss  0.07149384170770645 Accu  1.0
Epoch  254 Loss  0.07094225287437439 Accu  1.0
Epoch  255 Loss  0.07039695233106613 Accu  1.0
Epoch  256 Loss  0.06985782831907272 Accu  1.0
Epoch  257 Loss  0.06932481378316879 Accu  1.0
Epoch  258 Loss  0.06879783421754837 Accu  1.0
Epoch  259 Loss  0.0682767853140831 Accu  1.0
Epoch  260 Loss  0.0677616074681282 Accu  1.0
Epoch  261 Loss  0.06725221872329712 Accu  1.0
Epoch  262 Loss  0.06674852222204208 Accu  1.0
Epoch  263 Loss  0.06625046581029892 Accu  1.0
Epoch  264 Loss  0.06575797498226166 Accu  1.0
Epoch  265 Loss  0.06527096033096313 Accu  1.0
Epoch  266 Loss  0.06478935480117798 Accu  1.0
Epoch  267 Loss  0.06431309133768082 Accu  1.0
Epoch  268 Loss  0.06384208053350449 Accu  1.0
Epoch  269 Loss  0.0633762925863266 Accu  1.0
Epoch  270 Loss  0.0629156082868576 Accu  1.0
Epoch  271 Loss  0.06245999410748482 Accu  1.0
Epoch  272 Loss  0.062009360641241074 Accu  1.0
Epoch  273 Loss  0.061563652008771896 Accu  1.0
Epoch  274 Loss  0.06112281605601311 Accu  1.0
Epoch  275 Loss  0.06068676710128784 Accu  1.0
Epoch  276 Loss  0.06025545299053192 Accu  1.0
Epoch  277 Loss  0.05982881411910057 Accu  1.0
Epoch  278 Loss  0.05940677598118782 Accu  1.0
Epoch  279 Loss  0.058989278972148895 Accu  1.0
Epoch  280 Loss  0.05857628956437111 Accu  1.0
Epoch  281 Loss  0.058167714625597 Accu  1.0
Epoch  282 Loss  0.05776350200176239 Accu  1.0
Epoch  283 Loss  0.057363614439964294 Accu  1.0
Epoch  284 Loss  0.05696798115968704 Accu  1.0
Epoch  285 Loss  0.05657654628157616 Accu  1.0
Epoch  286 Loss  0.05618925765156746 Accu  1.0
Epoch  287 Loss  0.05580604821443558 Accu  1.0
Epoch  288 Loss  0.055426884442567825 Accu  1.0
Epoch  289 Loss  0.05505170673131943 Accu  1.0
Epoch  290 Loss  0.05468045920133591 Accu  1.0
Epoch  291 Loss  0.05431309714913368 Accu  1.0
Epoch  292 Loss  0.053949564695358276 Accu  1.0
Epoch  293 Loss  0.05358980968594551 Accu  1.0
Epoch  294 Loss  0.053233783692121506 Accu  1.0
Epoch  295 Loss  0.05288143828511238 Accu  1.0
```

```
Epoch  296 Loss  0.052532728761434555 Accu  1.0
Epoch  297 Loss  0.05218762159347534 Accu  1.0
Epoch  298 Loss  0.05184605345129967 Accu  1.0
Epoch  299 Loss  0.051507968455553055 Accu  1.0
Epoch  300 Loss  0.05117334797978401 Accu  1.0
Epoch  301 Loss  0.050842128694057465 Accu  1.0
Epoch  302 Loss  0.050514284521341324 Accu  1.0
Epoch  303 Loss  0.050189752131700516 Accu  1.0
Epoch  304 Loss  0.049868494272232056 Accu  1.0
Epoch  305 Loss  0.049550481140613556 Accu  1.0
Epoch  306 Loss  0.04923565685749054 Accu  1.0
Epoch  307 Loss  0.04892399162054062 Accu  1.0
Epoch  308 Loss  0.04861542955040932 Accu  1.0
Epoch  309 Loss  0.04830995574593544 Accu  1.0
Epoch  310 Loss  0.04800751432776451 Accu  1.0
Epoch  311 Loss  0.04770806431770325 Accu  1.0
Epoch  312 Loss  0.047411590814590454 Accu  1.0
Epoch  313 Loss  0.04711802676320076 Accu  1.0
Epoch  314 Loss  0.046827349811792374 Accu  1.0
Epoch  315 Loss  0.04653951898217201 Accu  1.0
Epoch  316 Loss  0.04625450819730759 Accu  1.0
Epoch  317 Loss  0.04597226530313492 Accu  1.0
Epoch  318 Loss  0.04569277912378311 Accu  1.0
Epoch  319 Loss  0.04541600123047829 Accu  1.0
Epoch  320 Loss  0.04514189437031746 Accu  1.0
Epoch  321 Loss  0.04487042874097824 Accu  1.0
Epoch  322 Loss  0.044601574540138245 Accu  1.0
Epoch  323 Loss  0.044335294514894485 Accu  1.0
Epoch  324 Loss  0.04407157376408577 Accu  1.0
Epoch  325 Loss  0.04381035268306732 Accu  1.0
Epoch  326 Loss  0.04355161637067795 Accu  1.0
Epoch  327 Loss  0.043295327574014664 Accu  1.0
Epoch  328 Loss  0.04304147884249687 Accu  1.0
Epoch  329 Loss  0.042790014296770096 Accu  1.0
Epoch  330 Loss  0.04254091531038284 Accu  1.0
Epoch  331 Loss  0.042294152081012726 Accu  1.0
Epoch  332 Loss  0.04204969108104706 Accu  1.0
Epoch  333 Loss  0.041807517409324646 Accu  1.0
Epoch  334 Loss  0.041567590087652206 Accu  1.0
Epoch  335 Loss  0.04132987931370735 Accu  1.0
Epoch  336 Loss  0.04109436646103859 Accu  1.0
Epoch  337 Loss  0.040861036628484726 Accu  1.0
Epoch  338 Loss  0.04062983766198158 Accu  1.0
Epoch  339 Loss  0.04040076211094856 Accu  1.0
Epoch  340 Loss  0.040173791348934174 Accu  1.0
Epoch  341 Loss  0.039948880672454834 Accu  1.0
Epoch  342 Loss  0.03972601145505905 Accu  1.0
Epoch  343 Loss  0.03950515761971474 Accu  1.0
```

```
Epoch  344 Loss  0.03928631544113159 Accu  1.0
Epoch  345 Loss  0.03906942531466484 Accu  1.0
Epoch  346 Loss  0.038854487240314484 Accu  1.0
Epoch  347 Loss  0.03864148631691933 Accu  1.0
Epoch  348 Loss  0.038430385291576385 Accu  1.0
Epoch  349 Loss  0.03822116181254387 Accu  1.0
Epoch  350 Loss  0.038013797253370285 Accu  1.0
Epoch  351 Loss  0.03780826926231384 Accu  1.0
Epoch  352 Loss  0.03760456293821335 Accu  1.0
Epoch  353 Loss  0.03740263730287552 Accu  1.0
Epoch  354 Loss  0.037202492356300354 Accu  1.0
Epoch  355 Loss  0.03700410574674606 Accu  1.0
Epoch  356 Loss  0.03680744394659996 Accu  1.0
Epoch  357 Loss  0.03661249950528145 Accu  1.0
Epoch  358 Loss  0.03641924634575844 Accu  1.0
Epoch  359 Loss  0.036227673292160034 Accu  1.0
Epoch  360 Loss  0.03603774309158325 Accu  1.0
Epoch  361 Loss  0.035849448293447495 Accu  1.0
Epoch  362 Loss  0.035662777721881866 Accu  1.0
Epoch  363 Loss  0.03547770157456398 Accu  1.0
Epoch  364 Loss  0.03529421240091324 Accu  1.0
Epoch  365 Loss  0.035112276673316956 Accu  1.0
Epoch  366 Loss  0.03493189066648483 Accu  1.0
Epoch  367 Loss  0.03475303575396538 Accu  1.0
Epoch  368 Loss  0.0345756858587265 Accu  1.0
Epoch  369 Loss  0.03439982607960701 Accu  1.0
Epoch  370 Loss  0.03422544524073601 Accu  1.0
Epoch  371 Loss  0.034052524715662 Accu  1.0
Epoch  372 Loss  0.033881042152643204 Accu  1.0
Epoch  373 Loss  0.03371099755167961 Accu  1.0
Epoch  374 Loss  0.03354236111044884 Accu  1.0
Epoch  375 Loss  0.03337512165307999 Accu  1.0
Epoch  376 Loss  0.033209264278411865 Accu  1.0
Epoch  377 Loss  0.033044759184122086 Accu  1.0
Epoch  378 Loss  0.032881610095500946 Accu  1.0
Epoch  379 Loss  0.03271980211138725 Accu  1.0
Epoch  380 Loss  0.032559316605329514 Accu  1.0
Epoch  381 Loss  0.03240013122558594 Accu  1.0
Epoch  382 Loss  0.03224225342273712 Accu  1.0
Epoch  383 Loss  0.03208563104271889 Accu  1.0
Epoch  384 Loss  0.03193028271198273 Accu  1.0
Epoch  385 Loss  0.03177618980407715 Accu  1.0
Epoch  386 Loss  0.03162332624197006 Accu  1.0
Epoch  387 Loss  0.03147169575095177 Accu  1.0
Epoch  388 Loss  0.03132126107811928 Accu  1.0
Epoch  389 Loss  0.03117203339934349 Accu  1.0
Epoch  390 Loss  0.031023994088172913 Accu  1.0
Epoch  391 Loss  0.0308771263808012 Accu  1.0
```

```
Epoch  392 Loss  0.030731409788131714 Accu  1.0
Epoch  393 Loss  0.030586842447519302 Accu  1.0
Epoch  394 Loss  0.03044341690838337 Accu  1.0
Epoch  395 Loss  0.030301107093691826 Accu  1.0
Epoch  396 Loss  0.03015991672873497 Accu  1.0
Epoch  397 Loss  0.030019821599125862 Accu  1.0
Epoch  398 Loss  0.029880817979574203 Accu  1.0
Epoch  399 Loss  0.029742885380983353 Accu  1.0
Epoch  400 Loss  0.02960602380335331 Accu  1.0
Epoch  401 Loss  0.02947021834552288 Accu  1.0
Epoch  402 Loss  0.029335442930459976 Accu  1.0
Epoch  403 Loss  0.02920171245932579 Accu  1.0
Epoch  404 Loss  0.029069004580378532 Accu  1.0
Epoch  405 Loss  0.02893730066716671 Accu  1.0
Epoch  406 Loss  0.02880660444498062 Accu  1.0
Epoch  407 Loss  0.028676895424723625 Accu  1.0
Epoch  408 Loss  0.028548164293169975 Accu  1.0
Epoch  409 Loss  0.02842041105031967 Accu  1.0
Epoch  410 Loss  0.028293613344430923 Accu  1.0
Epoch  411 Loss  0.028167767450213432 Accu  1.0
Epoch  412 Loss  0.028042856603860855 Accu  1.0
Epoch  413 Loss  0.02791888266801834 Accu  1.0
Epoch  414 Loss  0.027795828878879547 Accu  1.0
Epoch  415 Loss  0.027673687785863876 Accu  1.0
Epoch  416 Loss  0.027552451938390732 Accu  1.0
Epoch  417 Loss  0.02743210829794407 Accu  1.0
Epoch  418 Loss  0.02731264941394329 Accu  1.0
Epoch  419 Loss  0.02719406969845295 Accu  1.0
Epoch  420 Loss  0.02707635797560215 Accu  1.0
Epoch  421 Loss  0.026959510520100594 Accu  1.0
Epoch  422 Loss  0.02684350498020649 Accu  1.0
Epoch  423 Loss  0.026728352531790733 Accu  1.0
Epoch  424 Loss  0.026614027097821236 Accu  1.0
Epoch  425 Loss  0.026500524953007698 Accu  1.0
Epoch  426 Loss  0.026387840509414673 Accu  1.0
Epoch  427 Loss  0.02627597376704216 Accu  1.0
Epoch  428 Loss  0.02616489864885807 Accu  1.0
Epoch  429 Loss  0.026054630056023598 Accu  1.0
Epoch  430 Loss  0.025945130735635757 Accu  1.0
Epoch  431 Loss  0.02583642490208149 Accu  1.0
Epoch  432 Loss  0.025728488340973854 Accu  1.0
Epoch  433 Loss  0.025621306151151657 Accu  1.0
Epoch  434 Loss  0.025514887645840645 Accu  1.0
Epoch  435 Loss  0.025409216061234474 Accu  1.0
Epoch  436 Loss  0.025304287672042847 Accu  1.0
Epoch  437 Loss  0.025200089439749718 Accu  1.0
Epoch  438 Loss  0.025096625089645386 Accu  1.0
Epoch  439 Loss  0.024993881583213806 Accu  1.0
```

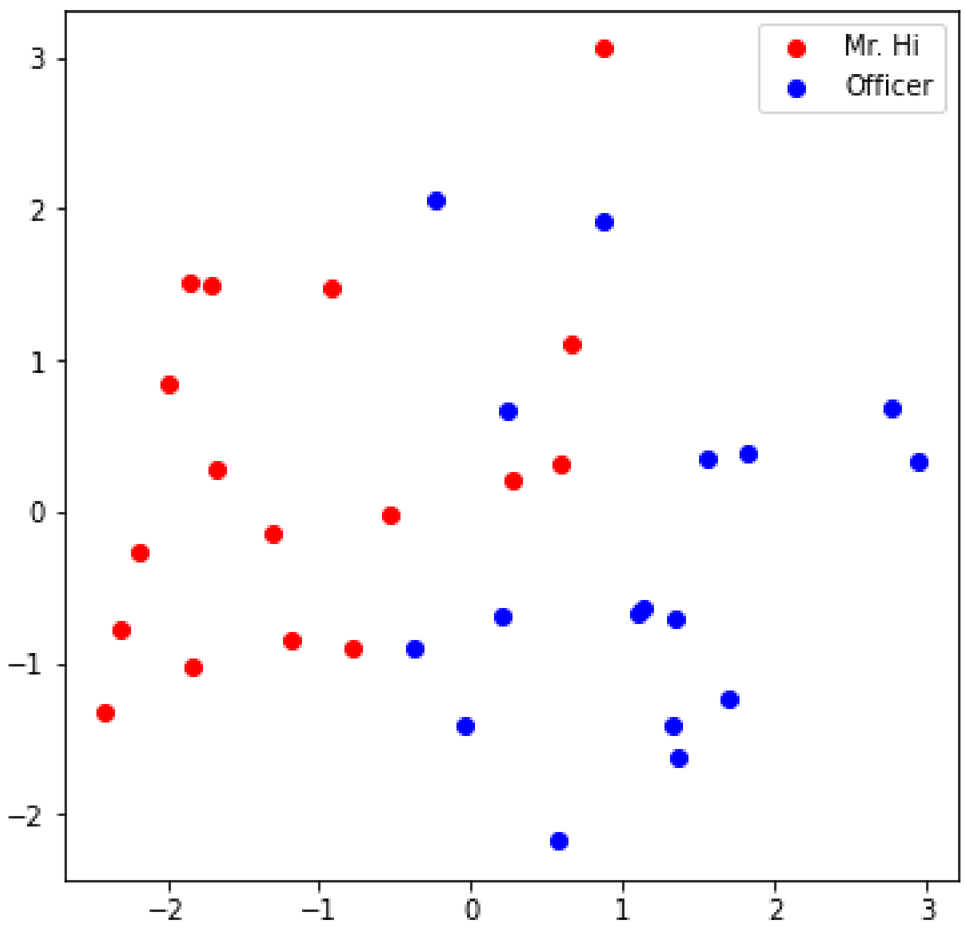
```
Epoch  440 Loss  0.024891844019293785 Accu  1.0
Epoch  441 Loss  0.02479051798582077 Accu  1.0
Epoch  442 Loss  0.024689890444278717 Accu  1.0
Epoch  443 Loss  0.024589957669377327 Accu  1.0
Epoch  444 Loss  0.024490708485245705 Accu  1.0
Epoch  445 Loss  0.024392144754529 Accu  1.0
Epoch  446 Loss  0.024294253438711166 Accu  1.0
Epoch  447 Loss  0.024197028949856758 Accu  1.0
Epoch  448 Loss  0.024100469425320625 Accu  1.0
Epoch  449 Loss  0.024004559963941574 Accu  1.0
Epoch  450 Loss  0.023909302428364754 Accu  1.0
Epoch  451 Loss  0.023814698681235313 Accu  1.0
Epoch  452 Loss  0.023720718920230865 Accu  1.0
Epoch  453 Loss  0.023627372458577156 Accu  1.0
Epoch  454 Loss  0.023534651845693588 Accu  1.0
Epoch  455 Loss  0.023442555218935013 Accu  1.0
Epoch  456 Loss  0.023351071402430534 Accu  1.0
Epoch  457 Loss  0.023260196670889854 Accu  1.0
Epoch  458 Loss  0.023169921711087227 Accu  1.0
Epoch  459 Loss  0.02308024652302265 Accu  1.0
Epoch  460 Loss  0.02299116551876068 Accu  1.0
Epoch  461 Loss  0.02290266565978527 Accu  1.0
Epoch  462 Loss  0.02281475067138672 Accu  1.0
Epoch  463 Loss  0.02272740937769413 Accu  1.0
Epoch  464 Loss  0.022640638053417206 Accu  1.0
Epoch  465 Loss  0.022554442286491394 Accu  1.0
Epoch  466 Loss  0.022468801587820053 Accu  1.0
Epoch  467 Loss  0.022383710369467735 Accu  1.0
Epoch  468 Loss  0.022299179807305336 Accu  1.0
Epoch  469 Loss  0.022215191274881363 Accu  1.0
Epoch  470 Loss  0.02213173918426037 Accu  1.0
Epoch  471 Loss  0.0220488291233778 Accu  1.0
Epoch  472 Loss  0.02196645550429821 Accu  1.0
Epoch  473 Loss  0.021884603425860405 Accu  1.0
Epoch  474 Loss  0.021803269162774086 Accu  1.0
Epoch  475 Loss  0.021722448989748955 Accu  1.0
Epoch  476 Loss  0.021642155945301056 Accu  1.0
Epoch  477 Loss  0.021562358364462852 Accu  1.0
Epoch  478 Loss  0.021483076736330986 Accu  1.0
Epoch  479 Loss  0.021404286846518517 Accu  1.0
Epoch  480 Loss  0.021325992420315742 Accu  1.0
Epoch  481 Loss  0.021248191595077515 Accu  1.0
Epoch  482 Loss  0.021170878782868385 Accu  1.0
Epoch  483 Loss  0.021094050258398056 Accu  1.0
Epoch  484 Loss  0.021017689257860184 Accu  1.0
Epoch  485 Loss  0.020941810682415962 Accu  1.0
Epoch  486 Loss  0.020866407081484795 Accu  1.0
Epoch  487 Loss  0.020791465416550636 Accu  1.0
```

```
Epoch  488 Loss  0.020716974511742592 Accu  1.0
Epoch  489 Loss  0.020642954856157303 Accu  1.0
Epoch  490 Loss  0.020569385960698128 Accu  1.0
Epoch  491 Loss  0.020496264100074768 Accu  1.0
Epoch  492 Loss  0.02042359486222267 Accu  1.0
Epoch  493 Loss  0.02035137265920639 Accu  1.0
Epoch  494 Loss  0.02027958072721958 Accu  1.0
Epoch  495 Loss  0.020208220928907394 Accu  1.0
Epoch  496 Loss  0.020137306302785873 Accu  1.0
Epoch  497 Loss  0.02006680890917778 Accu  1.0
Epoch  498 Loss  0.01999674178659916 Accu  1.0
Epoch  499 Loss  0.019927095621824265 Accu  1.0
```

4.4 Visualize the final node embeddings

Visualize your final embedding here! You can visually compare the figure with the previous embedding figure. After training, you should oberserve that the two classes are more evidently separated. This is a great sanitity check for your implementation as well.

[19]:
```
# Visualize the final learned embedding
visualize_emb(emb)
```



5 Submission

In order to get credit, you must go submit your answers on Gradescope.