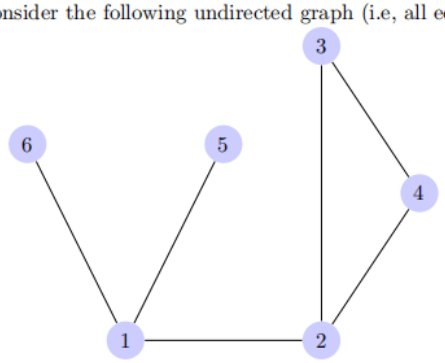
Arvind Sai DOODA **A20553046**

CS584 Machine Learning
Assignment 5

Problem 1 (25 Points): Label Propagation

Consider the following undirected graph (i.e, all edges have an edge weight 1) :



1. **3 Points.** Consider a binary classification problem. Write down the initial label vector P_0 for this graph where v_6 has observed label 1 and v_4 and v_5 have observed label 2.
2. **4 Points.** Perform 1 iteration of the label spreading algorithm with the decay parameter $\alpha = 0.8$ and determine the node labels for the unlabeled nodes v_1, v_2, and v_3, i.e., compute P_1 and provide the labels l_1, l_2, and l_3 after 1 iteration.
3. **4 Points.** Perform 2 iterations of the label spreading algorithm with the decay parameter $\alpha = 0.8$ and determine the node labels for the unlabeled nodes v_1, v_2, and v_3, i.e., compute P_2 and provide the labels l_1, l_2, and l_3 after 2 iterations.
4. **5 Points.** Perform infinite iterations of the label spreading algorithm with the decay parameter $\alpha = 0.8$ and determine the node labels for the unlabeled nodes v_1, v_2, and v_3, i.e., compute P_∞ and provide the labels l_1, l_2, and l_3 after infinite iterations.
5. **9 Points.** Determine the node labels for the unlabeled nodes v_1, v_2, and v_3 via the energy minimization algorithm.

1.

In [18]:
```
# 1. 3 Points. Consider a binary classification problem. Write down the initial label vector P0 for this graph
# where v6 has observed label 1 and v4 and v5 have observed label 2. (Done by Arvind -----)

import numpy as np
print("Arvind is solely responsible ")
print("libraries imported successfully")

#libraries were imprted succesfully
from scipy.linalg import fractional_matrix_power

# Initial label vector P0 values
P_0 = np.matrix([0, 0, 0, -1, -1, 1]).T
print("1. Initial label vector P0:\n", P_0)
```

```
Arvind is solely responsible
libraries imported successfully
1. Initial label vector P0:
 [[ 0]
 [ 0]
 [ 0]
 [-1]
 [-1]
 [ 1]]
```

2.

In [12]:
```
# 2. 4 Points. Perform 1 iteration of the label spreading algorithm with the decay parameter α = 0.8 and
# determine the node labels for the unlabeled nodes v1, v2, and v3, i.e., compute P1 and provide the labels
# l1, l2, and l3 after 1 iteration

# Similarity matrix S
S = np.array([[0, 1, 0, 0, 1, 1],
              [1, 0, 1, 1, 0, 0],
              [0, 1, 0, 1, 0, 0],
              [0, 1, 1, 0, 0, 0],
              [1, 0, 0, 0, 0, 0],
              [1, 0, 0, 0, 0, 0]])

# Normalize similarity matrix
# Similarity matrix S for this graph (with all edges = 1)
def sim_norm(S):
    D_ = np.diag(np.sum(S, axis=1))
    D_neg_half = fractional_matrix_power(D_, -0.5)
    S_norm = np.matmul(np.matmul(D_neg_half, S), D_neg_half)
    return np.round(S_norm, 3)

S_norm = sim_norm(S)

print("\nSimilarity matrix S:\n", S)
print("\nNormalized similarity matrix S_norm:\n", S_norm)

# Label spreading - 1 iteration, α = 0.8
alpha = 0.8
P_1 = (1 - alpha) * P_0 + alpha * np.matmul(S_norm, P_0)
[l0, l1, l2] = np.concatenate(P_1[:3])

print("\nLabel spreading - 1 iteration, α = 0.8:")
print("P_1 =\n", np.round(P_1, 3))
print("\nl0 = {}, l1 = {}, l2 = {}".format(l0, l1, l2))
```

Output :

```
Similarity matrix S:
 [[0 1 0 0 1 1]
 [1 0 1 1 0 0]
 [0 1 0 1 0 0]
 [0 1 1 0 0 0]
 [1 0 0 0 0 0]
 [1 0 0 0 0 0]]

Normalized similarity matrix S_norm:
 [[0.    0.333 0.    0.    0.577 0.577]
 [0.333 0.    0.408 0.408 0.    0.   ]
 [0.    0.408 0.    0.5   0.    0.   ]
 [0.    0.408 0.5   0.    0.    0.   ]
 [0.577 0.    0.    0.    0.    0.   ]
 [0.577 0.    0.    0.    0.    0.   ]]

Label spreading - 1 iteration, α = 0.8:
P_1 =
 [[ 0.   ]
 [-0.326]
 [-0.4  ]
 [-0.2  ]
 [-0.2  ]
 [ 0.2  ]]

l0 = [[0.]], l1 = [[-0.3264]], l2 = [[-0.4]]
```

3.

In [13]:
```
# 3. 4 Points. Perform 2 iterations of the label spreading algorithm with the decay parameter α = 0.8 and
# determine the node labels for the unlabeled nodes v1, v2, and v3, i.e., compute P2 and provide the labels
# l1, l2, and l3 after 2 iterations.

# Label spreading - 2 iterations, α = 0.8
P_2 = (1 - alpha) * P_1 + alpha * np.matmul(S_norm, P_1)
[l0, l1, l2] = np.concatenate(P_2[:3])

print("\n3. Label spreading - 2 iterations, α = 0.8:")
print("P_2 =\n", np.round(P_2, 3))
print("\nl0 = {}, l1 = {}, l2 = {}".format(l0, l1, l2))
```

```
3. Label spreading - 2 iterations, α = 0.8:
P_2 =
 [[-0.087]
 [-0.261]
 [-0.267]
 [-0.307]
 [-0.04 ]
 [ 0.04 ]]

l0 = [[-0.08695296]], l1 = [[-0.26112]], l2 = [[-0.26653696]]
```

4.

In [14]:
```
# 4. 5 Points. Perform infinite iterations of the label spreading algorithm with the decay parameter α = 0.8
# and determine the node labels for the unlabeled nodes v1, v2, and v3, i.e., compute P∞ and provide the
# labels l1, l2, and l3 after infinite iterations.

# Infinite label spreading, α = 0.8
I = np.identity(S_norm.shape[0])
I_aS_neg1 = fractional_matrix_power(I - alpha * S_norm, -1.0)
P_inf = np.round((1 - alpha) * np.matmul(I_aS_neg1, P_0), 3)
[l0, l1, l2] = np.concatenate(P_inf[:3])

print("\n4. Infinite label spreading, α = 0.8:")
print("P_inf =\n", P_inf)
print("\nl0 = {}, l1 = {}, l2 = {}".format(l0, l1, l2))
```

```
4. Infinite label spreading, α = 0.8:
P_inf =
 [[-0.097]
 [-0.209]
 [-0.209]
 [-0.352]
 [-0.245]
 [ 0.155]]

l0 = -0.097, l1 = -0.209, l2 = -0.209
```

5.

In [20]:
```
# 5. 9 Points. Determine the node labels for the unlabeled nodes v1, v2, and v3 via the energy minimization
# algorithm.

# Energy minimization
D = np.diag(np.sum(S, axis=1))
L = D - S
L_uu = L[:3, :3]
L_ul = L[:3, 3:]
Y_l = P_0[3:]
F_u = np.round(np.matmul(-fractional_matrix_power(L_uu, -1.0), np.matmul(L_ul, Y_l)), 3)
[l0, l1, l2] = np.round(F_u, 3).tolist()[:3]

print("\n5. Energy minimization:")
print("F_u =\n", F_u)
print("\nl0 = {}, l1 = {}, l2 = {}".format(l0, l1, l2))

print("     --------first problem completed Successfully-------")
print("                                        -       by Arvind")
```

```
5. Energy minimization:
F_u =
 [[-0.231]
 [-0.692]
 [-0.846]]

l0 = [-0.231], l1 = [-0.692], l2 = [-0.846]
     --------first problem completed Successfully-------
                                       -       by Arvind
```

—---------- **Problem 1 completed** —----------

Problem 2 (25 Points): Implementing Label Spreading

The following sample code is used to generate the two circles.

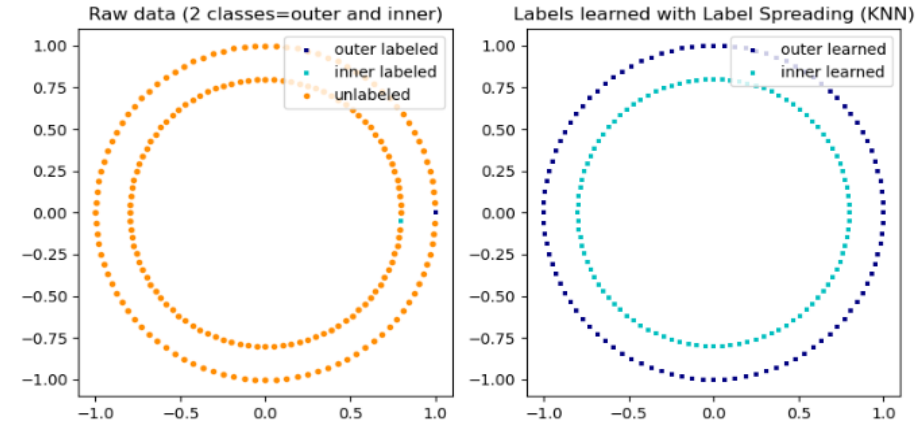
```
import numpy as np
from sklearn.datasets import make_circles
n_samples = 200
X, y = make_circles(n_samples=n_samples, shuffle=False)
outer, inner = 0, 1
labels = np.full(n_samples, -1.0)
labels[0] = outer
labels[-1] = inner
```

Code snippet and output were attached below

In [1]:
```
#   Problem 2 (25 Points): Implementing Label Spreading
#   In this problem, you will implement Label Spreading to classify data points from two circles (See left figure
#   in Figure 1). As both label groups lie inside their own distinct shape, we can see that Label Spreading can
#   propagate labels correctly around the circle (See right figure in Figure 1).

                                #  ARVIND (soley responsible)----------😊
# first we need to import the libraries
import numpy as np
from sklearn import datasets
from sklearn.semi_supervised import LabelSpreading
from sklearn.datasets import make_circles
import matplotlib.pyplot as plt

# Create the dataset
n_samples = 200
X, y = make_circles(n_samples=n_samples, shuffle=False)
outer, inner = 0, 1
labels = np.full(n_samples, -1.)
labels[0] = outer
labels[-1] = inner

# Create and fit the model
label_spread = LabelSpreading(kernel='knn', alpha=0.8)
label_spread.fit(X, labels)

# Predict the labels lamo
output_labels = label_spread.transduction_

# we should Plot the data now
plt.figure(figsize=(8.5, 4))
plt.subplot(1, 2, 1)
plt.scatter(X[labels == outer, 0], X[labels == outer, 1], color='navy',
            marker='s', lw=0, label="outer labeled", s=10)
plt.scatter(X[labels == inner, 0], X[labels == inner, 1], color='c',
            marker='s', lw=0, label='inner labeled', s=10)
plt.scatter(X[labels == -1, 0], X[labels == -1, 1], color='darkorange',
            marker='.', label='unlabeled')
plt.legend(scatterpoints=1, shadow=False, loc='upper right')
plt.title("Raw data (2 classes=outer and inner)")

plt.subplot(1, 2, 2)
plt.scatter(X[output_labels == outer, 0], X[output_labels == outer, 1], color='navy',
            marker='s', lw=0, s=10, label="outer learned")
plt.scatter(X[output_labels == inner, 0], X[output_labels == inner, 1], color='c',
            marker='s', lw=0, s=10, label="inner learned")
plt.legend(scatterpoints=1, shadow=False, loc='upper right')
plt.title("Labels learned with Label Spreading (KNN)")

plt.subplots_adjust(left=0.07, bottom=0.07, right=0.93, top=0.92)
plt.show()

# it prints the two circles    --------- finally done ----------------------
```

Output :

```
# it prints the two circles    --------- finally done ----------------------
```



Problem 3 (25 Points): Implementing Label Propagation with Self-Training

In this problem, we will incorporate self-training into label propagation (using energy minimization) to classify handwritten digits.

We start by learning a label propagation model with only 10 labeled points, then we select the top 5 most confident points to label. Next, we train with 15 labeled points (original 10 + 5 new ones). We repeat this process 4 times to have a model trained with 30 labeled examples. Please report *accuracy* and *confusion matrix* after learning each model.

The sample code to load the digit dataset is as follows:

```
import numpy as np
from sklearn import datasets

digits = datasets.load_digits()
rng = np.random.RandomState(0)
indices = np.arange(len(digits.data))
rng.shuffle(indices)

X = digits.data[indices[:330]]
y = digits.target[indices[:330]]
images = digits.images[indices[:330]]

n_total_samples = len(y)
n_labeled_points = 10
```

Code :

In [3]:

```
# Problem 3 (25 Points): Implementing Label Propagation with Self-Training
# In this problem, we will incorporate self-training into label propagation (using energy minimization) to classify
# handwritten digits. We start by learning a label propagation model with only 10 labeled points, then we select the top 5 most
# confident points to label. Next, we train with 15 labeled points (original 10 + 5 new ones). We repeat this
# process 4 times to have a model trained with 30 labeled examples. Please report accuracy and confusion matrix after learning each model.

# The above comment is the question ---------------------------------------- (Arvind 😊)

# we should import the libraries which we needed
import numpy as np
from sklearn import datasets
from sklearn.semi_supervised import LabelPropagation
from sklearn.metrics import accuracy_score, confusion_matrix

# Load digit dataset
digits = datasets.load_digits()
rng = np.random.RandomState(0)
indices = np.arange(len(digits.data))
rng.shuffle(indices)

X = digits.data[indices[:330]]
y = digits.target[indices[:330]]
images = digits.images[indices[:330]]
n_total_samples = len(y)
n_labeled_points = 10

label_prop_model = LabelPropagation(kernel='knn', n_neighbors=7, max_iter=1000)

for iteration in range(4):
    # Fit model with labeled data
    label_prop_model.fit(X[:n_labeled_points], y[:n_labeled_points])

    # Make predictions on all data (including unlabeled)
    y_pred = label_prop_model.predict(X)

    # Evaluate model performance
    accuracy = accuracy_score(y, y_pred)
    confusion_mat = confusion_matrix(y, y_pred)

    print(f"Iteration {iteration + 1} - Labeled Points: {n_labeled_points}")
    print(f" The Accuracy: {accuracy:.3f}")
    print("The Confusion Matrix:")
    print(confusion_mat)

    # To  Select the top 5 most confident points to label the below line of code is used
    confidence_scores = label_prop_model.predict_proba(X)
    top_confident_indices = np.argsort(np.max(confidence_scores, axis=1))[-5:]

    # now to Add the confident points to the labeled set
    X_labeled = X[:n_labeled_points]
    y_labeled = y[:n_labeled_points]
    X_labeled = np.concatenate((X_labeled, X[top_confident_indices]))
    y_labeled = np.concatenate((y_labeled, y_pred[top_confident_indices]))

    # To Increase  labeled points for the next iteration
    n_labeled_points += 5

    print("\n")

 # finally we can see the output
```

Output :

```
Iteration 1 - Labeled Points: 10
 The Accuracy: 0.242
The Confusion Matrix:
[[ 0  0  1  0  0  0 23  0  0  0]
 [ 0  1 17  0  0  0  3  0  9  0]
 [ 0  0 32  0  0  0  1  0  0  0]
 [ 0  1 27  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0 25  0  2  0]
 [ 0  1 13  0  0  0 19  0  3  0]
 [ 0  0  2  0  0  0 40  0  0  0]
 [ 0  3 13  0  0  0  3  0 18  0]
 [ 0  0 18  0  0  0 10  0  7  0]
 [ 0  0 28  0  0  0  4  0  6  0]]


Iteration 2 - Labeled Points: 15
 The Accuracy: 0.264
The Confusion Matrix:
[[ 0  0  0  0  0  0 24  0  0  0]
 [ 0  0  4  0  0  0 19  0  7  0]
 [ 0  0 30  0  0  0  1  0  2  0]
 [ 0  0 27  0  0  0  0  0  1  0]
 [ 0  0  0  0  0  0 26  0  1  0]
 [ 0  0  1  0  0  0 19  0 16  0]
 [ 0  0  0  0  0  0 42  0  0  0]
 [ 0  2  7  0  0  0  5  0 23  0]
 [ 0  0  5  0  0  0 15  0 15  0]
 [ 0  1 11  0  0  0 13  0 13  0]]


Iteration 3 - Labeled Points: 20
 The Accuracy: 0.336
The Confusion Matrix:
[[ 0  0  0  0  0  1 20  0  3  0]
 [ 1 15  3  0  0  0  8  0  3  0]
 [ 0  2 27  0  0  0  0  0  4  0]
 [ 0  1 26  0  0  0  0  0  1  0]
 [ 0  0  0  0  0  1 26  0  0  0]
 [ 0  1  1  0  0  5 10  0 19  0]
 [ 0  0  0  0  0  0 42  0  0  0]
 [ 1  3  5  0  0  8  0  0 20  0]
 [ 0  1  3  0  0  0  9  0 22  0]
 [ 1  2  9  0  0  2  4  0 20  0]]


Iteration 4 - Labeled Points: 25
 The Accuracy: 0.412
The Confusion Matrix:
[[ 0  0  0  0  0  0 12  0 12  0]
 [ 0 10  3  0  0  0  8  3  6  0]
 [ 0  2 17  0  0  0  1  1 12  0]
 [ 0  0  5  0  0  0  0  0 23  0]
 [ 1  0  0  0  0  0 23  3  0  0]
 [ 0  0  0  0  0  9  5  1 21  0]
 [ 0  0  0  0  0  0 42  0  0  0]
 [ 0  0  1  0  0  1  0 26  9  0]
 [ 0  0  0  0  0  0  1  2 32  0]
 [ 0  1  1  0  0  0  1  5 30  0]]
```

---------- **problem 3 completed** ----------

Problem 4 (25 Points): Implementing Graph Convolutional Networks

Graph convolutional network (GCN) is a well-known graph neural network for semi-supervised classification. More details please refer to https://arxiv.org/pdf/1609.02907.pdf.

The source code is in github (https://github.com/tkipf/pygcn). Please test GCN on the Cora dataset for node classification. Specifically, show GCN's node classification accuracy on Cora with the number of labeled nodes to be 60, 120, 180, 240, 300, respectively.

Code :

In [26]:

```
# Please test GCN on the Cora dataset for
# node classification. Specifically, show GCN's node classification accuracy on Cora with the number of labeled
# nodes to be 60, 120, 180, 240, 300, respectively  (done ---------- by Arvind )

from __future__ import division
from __future__ import print_function

import time
import argparse
import numpy as np
import os  # Added import for os module

import torch
import torch.nn.functional as F
import torch.optim as optim

from pygcn.utils import load_data, accuracy
from pygcn.models import GCN

# libraries imported

# Set the training settings
class Args:
    def __init__(self):
        self.no_cuda = False  # Set to True if you want to disable CUDA
        self.fastmode = False
        self.seed = 42
        self.epochs = 200
        self.lr = 0.01
        self.weight_decay = 5e-4
        self.hidden = 16
        self.dropout = 0.5

args = Args()
args.cuda = not args.no_cuda and torch.cuda.is_available()

np.random.seed(args.seed)
torch.manual_seed(args.seed)
if args.cuda:
    torch.cuda.manual_seed(args.seed)

# Specify the data of yours (here i gave mine)------------------------
data_dir = r"C:\Users\arvin\Downloads\pygcn-master\data\cora/"

# Load teh data ----
adj, features, labels, _, idx_val, idx_test = load_data(data_dir)

# Model and optimizer
model = GCN(nfeat=features.shape[1],
            nhid=args.hidden,
            nclass=labels.max().item() + 1,
            dropout=args.dropout)
optimizer = optim.Adam(model.parameters(),
                       lr=args.lr, weight_decay=args.weight_decay)
```

```
optimizer = optim.Adam(model.parameters(),
                       lr=args.lr, weight_decay=args.weight_decay)

if args.cuda:
    model.cuda()
    features = features.cuda()
    adj = adj.cuda()
    labels = labels.cuda()
    idx_val = idx_val.cuda()
    idx_test = idx_test.cuda()

# Define a list of labeled node counts here i gave all nodes that we want to test .
labeled_nodes_counts = [60, 120, 180, 240, 300]

for labeled_nodes_count in labeled_nodes_counts:
    # Randomly choose labeled nodes
    idx_train = np.random.choice(idx_val, labeled_nodes_count, replace=False)

    def train(epoch):
        t = time.time()
        model.train()
        optimizer.zero_grad()
        output = model(features, adj)
        loss_train = F.nll_loss(output[idx_train], labels[idx_train])
        acc_train = accuracy(output[idx_train], labels[idx_train])
        loss_train.backward()
        optimizer.step()

        if not args.fastmode:
            # Evaluate validation set performance separately,
            # deactivates dropout during the validation run.
            model.eval()
            output = model(features, adj)

        loss_val = F.nll_loss(output[idx_val], labels[idx_val])
        acc_val = accuracy(output[idx_val], labels[idx_val])
        print('Epoch: {:04d}'.format(epoch + 1),
              'loss_train: {:.4f}'.format(loss_train.item()),
              'acc_train: {:.4f}'.format(acc_train.item()),
              'loss_val: {:.4f}'.format(loss_val.item()),
              'acc_val: {:.4f}'.format(acc_val.item()),
              'time: {:.4f}s'.format(time.time() - t))

    def test():
        model.eval()
        output = model(features, adj)
        acc_test = accuracy(output[idx_test], labels[idx_test])
        print("Test set accuracy for {} labeled nodes: {:.4f}".format(labeled_nodes_count, acc_test.item()))

    # Train model
    t_total = time.time()
    for epoch in range(args.epochs):
        train(epoch)
    print(f"Optimization Finished for {labeled_nodes_count} labeled nodes!")
    print("Total time elapsed: {:.4f}s".format(time.time() - t_total))

    # Testing the model .
    test()
```

```
Loading cora dataset...
Epoch: 0001 loss_train: 1.9602 acc_train: 0.1500 loss_val: 1.9550 acc_val: 0.1167 time: 0.2196s
```

Output For node 60 labeled node :

```
Epoch: 0168 loss_train: 0.4605 acc_train: 0.9167 loss_val: 0.8539 acc_val: 0.7267 time: 0.0464s
Epoch: 0169 loss_train: 0.4792 acc_train: 0.9167 loss_val: 0.8503 acc_val: 0.7233 time: 0.0218s
Epoch: 0170 loss_train: 0.4802 acc_train: 0.9167 loss_val: 0.8474 acc_val: 0.7233 time: 0.0469s
Epoch: 0171 loss_train: 0.4208 acc_train: 0.9333 loss_val: 0.8442 acc_val: 0.7233 time: 0.0160s
Epoch: 0172 loss_train: 0.4391 acc_train: 0.9167 loss_val: 0.8414 acc_val: 0.7300 time: 0.0313s
Epoch: 0173 loss_train: 0.4062 acc_train: 0.9500 loss_val: 0.8386 acc_val: 0.7300 time: 0.0313s
Epoch: 0174 loss_train: 0.4189 acc_train: 0.9333 loss_val: 0.8361 acc_val: 0.7367 time: 0.0316s
Epoch: 0175 loss_train: 0.4607 acc_train: 0.9167 loss_val: 0.8334 acc_val: 0.7367 time: 0.0450s
Epoch: 0176 loss_train: 0.4540 acc_train: 0.9000 loss_val: 0.8313 acc_val: 0.7333 time: 0.0229s
Epoch: 0177 loss_train: 0.4187 acc_train: 0.9167 loss_val: 0.8280 acc_val: 0.7267 time: 0.0314s
Epoch: 0178 loss_train: 0.4740 acc_train: 0.9167 loss_val: 0.8245 acc_val: 0.7300 time: 0.0313s
Epoch: 0179 loss_train: 0.4486 acc_train: 0.9333 loss_val: 0.8201 acc_val: 0.7267 time: 0.0312s
Epoch: 0180 loss_train: 0.4207 acc_train: 0.8833 loss_val: 0.8162 acc_val: 0.7233 time: 0.0361s
Epoch: 0181 loss_train: 0.4685 acc_train: 0.8833 loss_val: 0.8126 acc_val: 0.7267 time: 0.0267s
Epoch: 0182 loss_train: 0.4014 acc_train: 0.9500 loss_val: 0.8102 acc_val: 0.7267 time: 0.0378s
Epoch: 0183 loss_train: 0.5147 acc_train: 0.9000 loss_val: 0.8088 acc_val: 0.7267 time: 0.0355s
Epoch: 0184 loss_train: 0.4512 acc_train: 0.9000 loss_val: 0.8081 acc_val: 0.7300 time: 0.0256s
Epoch: 0185 loss_train: 0.4119 acc_train: 0.9500 loss_val: 0.8088 acc_val: 0.7367 time: 0.0313s
Epoch: 0186 loss_train: 0.3860 acc_train: 0.9500 loss_val: 0.8100 acc_val: 0.7333 time: 0.0312s
Epoch: 0187 loss_train: 0.4733 acc_train: 0.9000 loss_val: 0.8108 acc_val: 0.7333 time: 0.0160s
Epoch: 0188 loss_train: 0.4271 acc_train: 0.9500 loss_val: 0.8098 acc_val: 0.7367 time: 0.0313s
Epoch: 0189 loss_train: 0.3742 acc_train: 0.9333 loss_val: 0.8055 acc_val: 0.7400 time: 0.0404s
Epoch: 0190 loss_train: 0.4351 acc_train: 0.9167 loss_val: 0.8011 acc_val: 0.7367 time: 0.0268s
Epoch: 0191 loss_train: 0.4023 acc_train: 0.9333 loss_val: 0.7966 acc_val: 0.7433 time: 0.0157s
Epoch: 0192 loss_train: 0.4165 acc_train: 0.9333 loss_val: 0.7928 acc_val: 0.7367 time: 0.0312s
Epoch: 0193 loss_train: 0.3610 acc_train: 0.9500 loss_val: 0.7899 acc_val: 0.7367 time: 0.0312s
Epoch: 0194 loss_train: 0.4449 acc_train: 0.9500 loss_val: 0.7878 acc_val: 0.7333 time: 0.0317s
Epoch: 0195 loss_train: 0.4381 acc_train: 0.8833 loss_val: 0.7861 acc_val: 0.7333 time: 0.0313s
Epoch: 0196 loss_train: 0.3759 acc_train: 0.9833 loss_val: 0.7846 acc_val: 0.7267 time: 0.0156s
Epoch: 0197 loss_train: 0.4415 acc_train: 0.9500 loss_val: 0.7815 acc_val: 0.7300 time: 0.0410s
Epoch: 0198 loss_train: 0.4322 acc_train: 0.9000 loss_val: 0.7777 acc_val: 0.7333 time: 0.0425s
Epoch: 0199 loss_train: 0.4151 acc_train: 0.9667 loss_val: 0.7761 acc_val: 0.7467 time: 0.0469s
Epoch: 0200 loss_train: 0.3849 acc_train: 0.9833 loss_val: 0.7758 acc_val: 0.7500 time: 0.0317s
Optimization Finished for 60 labeled nodes!
Total time elapsed: 9.3180s
Test set accuracy for 60 labeled nodes: 0.7510
Epoch: 0001 loss_train: 0.8843 acc_train: 0.7250 loss_val: 0.7637 acc_val: 0.7600 time: 0.0312s
Epoch: 0002 loss_train: 0.9355 acc_train: 0.7333 loss_val: 0.7453 acc_val: 0.7767 time: 0.0323s
Epoch: 0003 loss_train: 0.8827 acc_train: 0.7417 loss_val: 0.7279 acc_val: 0.7800 time: 0.0558s
Epoch: 0004 loss_train: 0.8222 acc_train: 0.7500 loss_val: 0.7177 acc_val: 0.8100 time: 0.0364s
Epoch: 0005 loss_train: 0.7924 acc_train: 0.8083 loss_val: 0.7164 acc_val: 0.8133 time: 0.0328s
Epoch: 0006 loss_train: 0.7506 acc_train: 0.8250 loss_val: 0.7178 acc_val: 0.8133 time: 0.0301s
Epoch: 0007 loss_train: 0.7046 acc_train: 0.8083 loss_val: 0.7192 acc_val: 0.8167 time: 0.0453s
Epoch: 0008 loss_train: 0.7114 acc_train: 0.7750 loss_val: 0.7135 acc_val: 0.8200 time: 0.0333s
Epoch: 0009 loss_train: 0.7705 acc_train: 0.8000 loss_val: 0.7001 acc_val: 0.8300 time: 0.0520s
Epoch: 0010 loss_train: 0.7116 acc_train: 0.8583 loss_val: 0.6806 acc_val: 0.8400 time: 0.0301s
Epoch: 0011 loss_train: 0.7161 acc_train: 0.8250 loss_val: 0.6592 acc_val: 0.8433 time: 0.0469s
Epoch: 0012 loss_train: 0.7225 acc_train: 0.8250 loss_val: 0.6392 acc_val: 0.8633 time: 0.0473s
Epoch: 0013 loss_train: 0.5815 acc_train: 0.8583 loss_val: 0.6226 acc_val: 0.8667 time: 0.0469s
Epoch: 0014 loss_train: 0.6360 acc_train: 0.8667 loss_val: 0.6092 acc_val: 0.8700 time: 0.0563s
Epoch: 0015 loss_train: 0.6208 acc_train: 0.8667 loss_val: 0.5986 acc_val: 0.8733 time: 0.0427s
Epoch: 0016 loss_train: 0.5414 acc_train: 0.8833 loss_val: 0.5895 acc_val: 0.8767 time: 0.0313s
```

Output for 120 labeled node :

```
Epoch: 0175 loss_train: 0.3272 acc_train: 0.9417 loss_val: 0.5650 acc_val: 0.8567 time: 0.0313s
Epoch: 0176 loss_train: 0.3055 acc_train: 0.9583 loss_val: 0.5677 acc_val: 0.8533 time: 0.0311s
Epoch: 0177 loss_train: 0.2968 acc_train: 0.9583 loss_val: 0.5695 acc_val: 0.8567 time: 0.0313s
Epoch: 0178 loss_train: 0.3109 acc_train: 0.9250 loss_val: 0.5684 acc_val: 0.8567 time: 0.0312s
Epoch: 0179 loss_train: 0.2786 acc_train: 0.9833 loss_val: 0.5666 acc_val: 0.8567 time: 0.0378s
Epoch: 0180 loss_train: 0.3065 acc_train: 0.9500 loss_val: 0.5642 acc_val: 0.8567 time: 0.0436s
Epoch: 0181 loss_train: 0.2874 acc_train: 0.9750 loss_val: 0.5624 acc_val: 0.8533 time: 0.0323s
Epoch: 0182 loss_train: 0.2770 acc_train: 0.9583 loss_val: 0.5602 acc_val: 0.8533 time: 0.0473s
Epoch: 0183 loss_train: 0.2992 acc_train: 0.9667 loss_val: 0.5593 acc_val: 0.8567 time: 0.0312s
Epoch: 0184 loss_train: 0.2756 acc_train: 0.9583 loss_val: 0.5578 acc_val: 0.8600 time: 0.0473s
Epoch: 0185 loss_train: 0.2876 acc_train: 0.9583 loss_val: 0.5577 acc_val: 0.8567 time: 0.0313s
Epoch: 0186 loss_train: 0.2945 acc_train: 0.9750 loss_val: 0.5595 acc_val: 0.8533 time: 0.0365s
Epoch: 0187 loss_train: 0.3062 acc_train: 0.9750 loss_val: 0.5616 acc_val: 0.8533 time: 0.0318s
Epoch: 0188 loss_train: 0.3082 acc_train: 0.9583 loss_val: 0.5643 acc_val: 0.8533 time: 0.0313s
Epoch: 0189 loss_train: 0.2908 acc_train: 0.9583 loss_val: 0.5671 acc_val: 0.8500 time: 0.0312s
Epoch: 0190 loss_train: 0.2694 acc_train: 0.9667 loss_val: 0.5708 acc_val: 0.8467 time: 0.0473s
Epoch: 0191 loss_train: 0.2659 acc_train: 0.9500 loss_val: 0.5732 acc_val: 0.8400 time: 0.0313s
Epoch: 0192 loss_train: 0.3195 acc_train: 0.9667 loss_val: 0.5729 acc_val: 0.8400 time: 0.0455s
Epoch: 0193 loss_train: 0.2784 acc_train: 0.9500 loss_val: 0.5693 acc_val: 0.8433 time: 0.0382s
Epoch: 0194 loss_train: 0.2670 acc_train: 0.9833 loss_val: 0.5651 acc_val: 0.8500 time: 0.0313s
Epoch: 0195 loss_train: 0.2597 acc_train: 0.9667 loss_val: 0.5592 acc_val: 0.8533 time: 0.0312s
Epoch: 0196 loss_train: 0.2983 acc_train: 0.9750 loss_val: 0.5569 acc_val: 0.8600 time: 0.0317s
Epoch: 0197 loss_train: 0.2865 acc_train: 0.9583 loss_val: 0.5558 acc_val: 0.8600 time: 0.0312s
Epoch: 0198 loss_train: 0.2899 acc_train: 0.9167 loss_val: 0.5555 acc_val: 0.8633 time: 0.0312s
Epoch: 0199 loss_train: 0.2793 acc_train: 0.9667 loss_val: 0.5574 acc_val: 0.8600 time: 0.0428s
Epoch: 0200 loss_train: 0.2756 acc_train: 0.9583 loss_val: 0.5585 acc_val: 0.8600 time: 0.0403s
Optimization Finished for 120 labeled nodes!
Total time elapsed: 8.9448s
Test set accuracy for 120 labeled nodes: 0.7790
Epoch: 0001 loss_train: 0.6716 acc_train: 0.8111 loss_val: 0.5403 acc_val: 0.8700 time: 0.0473s
Epoch: 0002 loss_train: 0.6394 acc_train: 0.8111 loss_val: 0.5173 acc_val: 0.8800 time: 0.0313s
Epoch: 0003 loss_train: 0.5552 acc_train: 0.8278 loss_val: 0.4983 acc_val: 0.8833 time: 0.0313s
Epoch: 0004 loss_train: 0.5482 acc_train: 0.8444 loss_val: 0.4853 acc_val: 0.9000 time: 0.0473s
Epoch: 0005 loss_train: 0.5517 acc_train: 0.8389 loss_val: 0.4731 acc_val: 0.9100 time: 0.0437s
Epoch: 0006 loss_train: 0.5515 acc_train: 0.8722 loss_val: 0.4591 acc_val: 0.9133 time: 0.0328s
Epoch: 0007 loss_train: 0.5520 acc_train: 0.8667 loss_val: 0.4445 acc_val: 0.9267 time: 0.0475s
Epoch: 0008 loss_train: 0.4655 acc_train: 0.9167 loss_val: 0.4310 acc_val: 0.9300 time: 0.0306s
Epoch: 0009 loss_train: 0.4907 acc_train: 0.9000 loss_val: 0.4212 acc_val: 0.9267 time: 0.0317s
Epoch: 0010 loss_train: 0.4765 acc_train: 0.9167 loss_val: 0.4138 acc_val: 0.9167 time: 0.0469s
Epoch: 0011 loss_train: 0.4241 acc_train: 0.9000 loss_val: 0.4073 acc_val: 0.9133 time: 0.0304s
Epoch: 0012 loss_train: 0.4663 acc_train: 0.9056 loss_val: 0.3984 acc_val: 0.9133 time: 0.0334s
Epoch: 0013 loss_train: 0.4010 acc_train: 0.9167 loss_val: 0.3896 acc_val: 0.9200 time: 0.0313s
Epoch: 0014 loss_train: 0.4310 acc_train: 0.8778 loss_val: 0.3819 acc_val: 0.9200 time: 0.0312s
Epoch: 0015 loss_train: 0.4571 acc_train: 0.9111 loss_val: 0.3773 acc_val: 0.9333 time: 0.0317s
Epoch: 0016 loss_train: 0.3834 acc_train: 0.9333 loss_val: 0.3742 acc_val: 0.9400 time: 0.0469s
Epoch: 0017 loss_train: 0.3958 acc_train: 0.9333 loss_val: 0.3712 acc_val: 0.9367 time: 0.0463s
Epoch: 0018 loss_train: 0.3910 acc_train: 0.9389 loss_val: 0.3678 acc_val: 0.9367 time: 0.0409s
Epoch: 0019 loss_train: 0.3673 acc_train: 0.9556 loss_val: 0.3639 acc_val: 0.9333 time: 0.0313s
Epoch: 0020 loss_train: 0.3503 acc_train: 0.9444 loss_val: 0.3605 acc_val: 0.9300 time: 0.0473s
Epoch: 0021 loss_train: 0.3454 acc_train: 0.9611 loss_val: 0.3578 acc_val: 0.9233 time: 0.0313s
Epoch: 0022 loss_train: 0.3070 acc_train: 0.9444 loss_val: 0.3558 acc_val: 0.9233 time: 0.0312s
Epoch: 0023 loss_train: 0.3557 acc_train: 0.9500 loss_val: 0.3545 acc_val: 0.9200 time: 0.0390s
Epoch: 0024 loss_train: 0.3456 acc_train: 0.9389 loss_val: 0.3530 acc_val: 0.9233 time: 0.0312s
Epoch: 0025 loss_train: 0.3544 acc_train: 0.9333 loss_val: 0.3518 acc_val: 0.9200 time: 0.0167s
```

Output for 180 labeled node :

```
Epoch: 0174 loss_train: 0.2293 acc_train: 0.9722 loss_val: 0.3489 acc_val: 0.9067 time: 0.0601s
Epoch: 0175 loss_train: 0.2377 acc_train: 0.9722 loss_val: 0.3507 acc_val: 0.9067 time: 0.0466s
Epoch: 0176 loss_train: 0.2612 acc_train: 0.9500 loss_val: 0.3519 acc_val: 0.9067 time: 0.0473s
Epoch: 0177 loss_train: 0.2576 acc_train: 0.9389 loss_val: 0.3523 acc_val: 0.9033 time: 0.0469s
Epoch: 0178 loss_train: 0.2747 acc_train: 0.9556 loss_val: 0.3517 acc_val: 0.9067 time: 0.0468s
Epoch: 0179 loss_train: 0.2439 acc_train: 0.9556 loss_val: 0.3507 acc_val: 0.9067 time: 0.0587s
Epoch: 0180 loss_train: 0.2794 acc_train: 0.9556 loss_val: 0.3500 acc_val: 0.9100 time: 0.0313s
Epoch: 0181 loss_train: 0.2887 acc_train: 0.9500 loss_val: 0.3497 acc_val: 0.9167 time: 0.0625s
Epoch: 0182 loss_train: 0.2561 acc_train: 0.9667 loss_val: 0.3490 acc_val: 0.9167 time: 0.0468s
Epoch: 0183 loss_train: 0.2795 acc_train: 0.9500 loss_val: 0.3486 acc_val: 0.9233 time: 0.0469s
Epoch: 0184 loss_train: 0.2432 acc_train: 0.9778 loss_val: 0.3485 acc_val: 0.9233 time: 0.0543s
Epoch: 0185 loss_train: 0.2532 acc_train: 0.9667 loss_val: 0.3492 acc_val: 0.9167 time: 0.0625s
Epoch: 0186 loss_train: 0.2704 acc_train: 0.9667 loss_val: 0.3498 acc_val: 0.9167 time: 0.0317s
Epoch: 0187 loss_train: 0.2765 acc_train: 0.9500 loss_val: 0.3504 acc_val: 0.9133 time: 0.0469s
Epoch: 0188 loss_train: 0.2707 acc_train: 0.9500 loss_val: 0.3519 acc_val: 0.9100 time: 0.0629s
Epoch: 0189 loss_train: 0.2557 acc_train: 0.9778 loss_val: 0.3525 acc_val: 0.9100 time: 0.0377s
Epoch: 0190 loss_train: 0.2667 acc_train: 0.9833 loss_val: 0.3522 acc_val: 0.9100 time: 0.0473s
Epoch: 0191 loss_train: 0.2870 acc_train: 0.9667 loss_val: 0.3515 acc_val: 0.9100 time: 0.0625s
Epoch: 0192 loss_train: 0.2507 acc_train: 0.9667 loss_val: 0.3510 acc_val: 0.9100 time: 0.0473s
Epoch: 0193 loss_train: 0.2702 acc_train: 0.9389 loss_val: 0.3497 acc_val: 0.9100 time: 0.0564s
Epoch: 0194 loss_train: 0.2568 acc_train: 0.9667 loss_val: 0.3487 acc_val: 0.9100 time: 0.0513s
Epoch: 0195 loss_train: 0.2238 acc_train: 0.9833 loss_val: 0.3480 acc_val: 0.9133 time: 0.0469s
Epoch: 0196 loss_train: 0.2868 acc_train: 0.9556 loss_val: 0.3471 acc_val: 0.9167 time: 0.0624s
Epoch: 0197 loss_train: 0.2642 acc_train: 0.9500 loss_val: 0.3465 acc_val: 0.9167 time: 0.0543s
Epoch: 0198 loss_train: 0.2328 acc_train: 0.9667 loss_val: 0.3465 acc_val: 0.9200 time: 0.0545s
Epoch: 0199 loss_train: 0.2569 acc_train: 0.9556 loss_val: 0.3471 acc_val: 0.9133 time: 0.0660s
Epoch: 0200 loss_train: 0.2748 acc_train: 0.9556 loss_val: 0.3479 acc_val: 0.9167 time: 0.0490s
Optimization Finished for 180 labeled nodes!
Total time elapsed: 7.8153s
Test set accuracy for 180 labeled nodes: 0.7930
Epoch: 0001 loss_train: 0.4646 acc_train: 0.8750 loss_val: 0.3419 acc_val: 0.9233 time: 0.0567s
Epoch: 0002 loss_train: 0.3996 acc_train: 0.9042 loss_val: 0.3336 acc_val: 0.9267 time: 0.0499s
Epoch: 0003 loss_train: 0.4422 acc_train: 0.8708 loss_val: 0.3258 acc_val: 0.9333 time: 0.0313s
Epoch: 0004 loss_train: 0.3747 acc_train: 0.9167 loss_val: 0.3190 acc_val: 0.9333 time: 0.0316s
Epoch: 0005 loss_train: 0.4104 acc_train: 0.8917 loss_val: 0.3147 acc_val: 0.9400 time: 0.0333s
Epoch: 0006 loss_train: 0.3915 acc_train: 0.9083 loss_val: 0.3107 acc_val: 0.9500 time: 0.0297s
Epoch: 0007 loss_train: 0.3512 acc_train: 0.9292 loss_val: 0.3059 acc_val: 0.9500 time: 0.0324s
Epoch: 0008 loss_train: 0.3547 acc_train: 0.9125 loss_val: 0.2998 acc_val: 0.9467 time: 0.0351s
Epoch: 0009 loss_train: 0.3343 acc_train: 0.9333 loss_val: 0.2927 acc_val: 0.9500 time: 0.0317s
Epoch: 0010 loss_train: 0.3459 acc_train: 0.9500 loss_val: 0.2848 acc_val: 0.9533 time: 0.0469s
Epoch: 0011 loss_train: 0.4001 acc_train: 0.9125 loss_val: 0.2784 acc_val: 0.9467 time: 0.0378s
Epoch: 0012 loss_train: 0.3500 acc_train: 0.9333 loss_val: 0.2733 acc_val: 0.9500 time: 0.0622s
Epoch: 0013 loss_train: 0.3196 acc_train: 0.9542 loss_val: 0.2680 acc_val: 0.9500 time: 0.0652s
Epoch: 0014 loss_train: 0.3381 acc_train: 0.9417 loss_val: 0.2626 acc_val: 0.9533 time: 0.0597s
Epoch: 0015 loss_train: 0.2886 acc_train: 0.9458 loss_val: 0.2590 acc_val: 0.9667 time: 0.0648s
Epoch: 0016 loss_train: 0.3058 acc_train: 0.9583 loss_val: 0.2568 acc_val: 0.9633 time: 0.0477s
Epoch: 0017 loss_train: 0.3175 acc_train: 0.9500 loss_val: 0.2555 acc_val: 0.9633 time: 0.0380s
Epoch: 0018 loss_train: 0.3151 acc_train: 0.9417 loss_val: 0.2540 acc_val: 0.9667 time: 0.0326s
Epoch: 0019 loss_train: 0.3146 acc_train: 0.9500 loss_val: 0.2524 acc_val: 0.9667 time: 0.0317s
Epoch: 0020 loss_train: 0.3098 acc_train: 0.9542 loss_val: 0.2520 acc_val: 0.9600 time: 0.0469s
Epoch: 0021 loss_train: 0.2667 acc_train: 0.9458 loss_val: 0.2531 acc_val: 0.9567 time: 0.0363s
Epoch: 0022 loss_train: 0.3298 acc_train: 0.9500 loss_val: 0.2548 acc_val: 0.9567 time: 0.0352s
Epoch: 0023 loss_train: 0.2871 acc_train: 0.9417 loss_val: 0.2555 acc_val: 0.9500 time: 0.0264s
Epoch: 0024 loss_train: 0.2898 acc_train: 0.9458 loss_val: 0.2557 acc_val: 0.9533 time: 0.0473s
Epoch: 0025 loss_train: 0.2859 acc_train: 0.9583 loss_val: 0.2552 acc_val: 0.9600 time: 0.0313s
Epoch: 0026 loss_train: 0.2837 acc_train: 0.9417 loss_val: 0.2551 acc_val: 0.9633 time: 0.0312s
```

Output for 240 labeled node :

```
Epoch: 0174 loss_train: 0.2731 acc_train: 0.9708 loss_val: 0.2413 acc_val: 0.9667 time: 0.0350s
Epoch: 0175 loss_train: 0.2849 acc_train: 0.9500 loss_val: 0.2416 acc_val: 0.9667 time: 0.0157s
Epoch: 0176 loss_train: 0.2644 acc_train: 0.9708 loss_val: 0.2420 acc_val: 0.9667 time: 0.0325s
Epoch: 0177 loss_train: 0.2669 acc_train: 0.9708 loss_val: 0.2421 acc_val: 0.9633 time: 0.0145s
Epoch: 0178 loss_train: 0.2631 acc_train: 0.9625 loss_val: 0.2416 acc_val: 0.9633 time: 0.0420s
Epoch: 0179 loss_train: 0.2840 acc_train: 0.9500 loss_val: 0.2416 acc_val: 0.9633 time: 0.0278s
Epoch: 0180 loss_train: 0.2778 acc_train: 0.9417 loss_val: 0.2427 acc_val: 0.9667 time: 0.0288s
Epoch: 0181 loss_train: 0.2788 acc_train: 0.9583 loss_val: 0.2443 acc_val: 0.9667 time: 0.0293s
Epoch: 0182 loss_train: 0.2693 acc_train: 0.9583 loss_val: 0.2451 acc_val: 0.9700 time: 0.0327s
Epoch: 0183 loss_train: 0.2849 acc_train: 0.9625 loss_val: 0.2447 acc_val: 0.9700 time: 0.0293s
Epoch: 0184 loss_train: 0.2569 acc_train: 0.9625 loss_val: 0.2433 acc_val: 0.9700 time: 0.0341s
Epoch: 0185 loss_train: 0.2814 acc_train: 0.9500 loss_val: 0.2421 acc_val: 0.9700 time: 0.0342s
Epoch: 0186 loss_train: 0.2737 acc_train: 0.9708 loss_val: 0.2411 acc_val: 0.9667 time: 0.0295s
Epoch: 0187 loss_train: 0.2763 acc_train: 0.9625 loss_val: 0.2410 acc_val: 0.9667 time: 0.0363s
Epoch: 0188 loss_train: 0.2619 acc_train: 0.9667 loss_val: 0.2419 acc_val: 0.9633 time: 0.0302s
Epoch: 0189 loss_train: 0.2731 acc_train: 0.9542 loss_val: 0.2429 acc_val: 0.9633 time: 0.0302s
Epoch: 0190 loss_train: 0.2702 acc_train: 0.9417 loss_val: 0.2434 acc_val: 0.9633 time: 0.0237s
Epoch: 0191 loss_train: 0.2605 acc_train: 0.9417 loss_val: 0.2428 acc_val: 0.9633 time: 0.0282s
Epoch: 0192 loss_train: 0.2875 acc_train: 0.9542 loss_val: 0.2414 acc_val: 0.9667 time: 0.0273s
Epoch: 0193 loss_train: 0.2492 acc_train: 0.9792 loss_val: 0.2418 acc_val: 0.9667 time: 0.0290s
Epoch: 0194 loss_train: 0.2709 acc_train: 0.9625 loss_val: 0.2431 acc_val: 0.9667 time: 0.0314s
Epoch: 0195 loss_train: 0.2746 acc_train: 0.9625 loss_val: 0.2434 acc_val: 0.9700 time: 0.0278s
Epoch: 0196 loss_train: 0.2446 acc_train: 0.9792 loss_val: 0.2431 acc_val: 0.9700 time: 0.0339s
Epoch: 0197 loss_train: 0.2507 acc_train: 0.9583 loss_val: 0.2434 acc_val: 0.9667 time: 0.0248s
Epoch: 0198 loss_train: 0.2760 acc_train: 0.9542 loss_val: 0.2440 acc_val: 0.9667 time: 0.0241s
Epoch: 0199 loss_train: 0.2504 acc_train: 0.9667 loss_val: 0.2437 acc_val: 0.9667 time: 0.0209s
Epoch: 0200 loss_train: 0.2402 acc_train: 0.9792 loss_val: 0.2429 acc_val: 0.9667 time: 0.0156s
Optimization Finished for 240 labeled nodes!
Total time elapsed: 6.5618s
Test set accuracy for 240 labeled nodes: 0.8220
Epoch: 0001 loss_train: 0.3145 acc_train: 0.9400 loss_val: 0.2408 acc_val: 0.9667 time: 0.0156s
Epoch: 0002 loss_train: 0.3147 acc_train: 0.9467 loss_val: 0.2378 acc_val: 0.9700 time: 0.0381s
Epoch: 0003 loss_train: 0.3026 acc_train: 0.9467 loss_val: 0.2355 acc_val: 0.9700 time: 0.0275s
Epoch: 0004 loss_train: 0.3003 acc_train: 0.9400 loss_val: 0.2337 acc_val: 0.9700 time: 0.0270s
Epoch: 0005 loss_train: 0.2937 acc_train: 0.9333 loss_val: 0.2310 acc_val: 0.9667 time: 0.0293s
Epoch: 0006 loss_train: 0.3032 acc_train: 0.9500 loss_val: 0.2275 acc_val: 0.9667 time: 0.0152s
Epoch: 0007 loss_train: 0.3166 acc_train: 0.9500 loss_val: 0.2230 acc_val: 0.9667 time: 0.0313s
Epoch: 0008 loss_train: 0.3281 acc_train: 0.9367 loss_val: 0.2196 acc_val: 0.9633 time: 0.0169s
Epoch: 0009 loss_train: 0.3028 acc_train: 0.9333 loss_val: 0.2170 acc_val: 0.9700 time: 0.0301s
Epoch: 0010 loss_train: 0.2937 acc_train: 0.9467 loss_val: 0.2149 acc_val: 0.9700 time: 0.0160s
Epoch: 0011 loss_train: 0.2931 acc_train: 0.9367 loss_val: 0.2131 acc_val: 0.9733 time: 0.0313s
Epoch: 0012 loss_train: 0.2810 acc_train: 0.9333 loss_val: 0.2120 acc_val: 0.9767 time: 0.0156s
Epoch: 0013 loss_train: 0.3054 acc_train: 0.9433 loss_val: 0.2117 acc_val: 0.9733 time: 0.0386s
Epoch: 0014 loss_train: 0.2943 acc_train: 0.9400 loss_val: 0.2107 acc_val: 0.9733 time: 0.0136s
Epoch: 0015 loss_train: 0.2955 acc_train: 0.9367 loss_val: 0.2098 acc_val: 0.9733 time: 0.0156s
Epoch: 0016 loss_train: 0.2700 acc_train: 0.9400 loss_val: 0.2088 acc_val: 0.9767 time: 0.0312s
Epoch: 0017 loss_train: 0.2744 acc_train: 0.9600 loss_val: 0.2077 acc_val: 0.9700 time: 0.0156s
Epoch: 0018 loss_train: 0.2891 acc_train: 0.9633 loss_val: 0.2065 acc_val: 0.9700 time: 0.0317s
Epoch: 0019 loss_train: 0.2723 acc_train: 0.9467 loss_val: 0.2054 acc_val: 0.9700 time: 0.0313s
Epoch: 0020 loss_train: 0.2774 acc_train: 0.9433 loss_val: 0.2046 acc_val: 0.9700 time: 0.0156s
Epoch: 0021 loss_train: 0.2755 acc_train: 0.9467 loss_val: 0.2042 acc_val: 0.9667 time: 0.0313s
Epoch: 0022 loss_train: 0.2705 acc_train: 0.9533 loss_val: 0.2041 acc_val: 0.9633 time: 0.0262s
Epoch: 0023 loss_train: 0.2779 acc_train: 0.9500 loss_val: 0.2038 acc_val: 0.9667 time: 0.0177s
Epoch: 0024 loss_train: 0.2678 acc_train: 0.9467 loss_val: 0.2035 acc_val: 0.9667 time: 0.0156s
Epoch: 0025 loss_train: 0.2608 acc_train: 0.9567 loss_val: 0.2030 acc_val: 0.9733 time: 0.0312s
```

Output for 300 labeled node :

```
Epoch: 0155 loss_train: 0.2716 acc_train: 0.9600 loss_val: 0.2022 acc_val: 0.9733 time: 0.0316s
Epoch: 0156 loss_train: 0.2746 acc_train: 0.9300 loss_val: 0.2028 acc_val: 0.9767 time: 0.0313s
Epoch: 0157 loss_train: 0.2799 acc_train: 0.9533 loss_val: 0.2034 acc_val: 0.9767 time: 0.0156s
Epoch: 0158 loss_train: 0.2946 acc_train: 0.9500 loss_val: 0.2035 acc_val: 0.9767 time: 0.0440s
Epoch: 0159 loss_train: 0.2858 acc_train: 0.9400 loss_val: 0.2030 acc_val: 0.9767 time: 0.0248s
Epoch: 0160 loss_train: 0.2532 acc_train: 0.9600 loss_val: 0.2027 acc_val: 0.9767 time: 0.0313s
Epoch: 0161 loss_train: 0.2755 acc_train: 0.9467 loss_val: 0.2029 acc_val: 0.9767 time: 0.0312s
Epoch: 0162 loss_train: 0.2854 acc_train: 0.9533 loss_val: 0.2035 acc_val: 0.9767 time: 0.0473s
Epoch: 0163 loss_train: 0.2851 acc_train: 0.9500 loss_val: 0.2032 acc_val: 0.9733 time: 0.0313s
Epoch: 0164 loss_train: 0.2777 acc_train: 0.9533 loss_val: 0.2023 acc_val: 0.9700 time: 0.0312s
Epoch: 0165 loss_train: 0.2781 acc_train: 0.9533 loss_val: 0.2016 acc_val: 0.9733 time: 0.0426s
Epoch: 0166 loss_train: 0.2660 acc_train: 0.9433 loss_val: 0.2016 acc_val: 0.9767 time: 0.0115s
Epoch: 0167 loss_train: 0.2928 acc_train: 0.9367 loss_val: 0.2014 acc_val: 0.9800 time: 0.0469s
Epoch: 0168 loss_train: 0.2667 acc_train: 0.9500 loss_val: 0.2012 acc_val: 0.9800 time: 0.0317s
Epoch: 0169 loss_train: 0.2676 acc_train: 0.9533 loss_val: 0.2013 acc_val: 0.9833 time: 0.0312s
Epoch: 0170 loss_train: 0.2776 acc_train: 0.9533 loss_val: 0.2026 acc_val: 0.9867 time: 0.0312s
Epoch: 0171 loss_train: 0.2774 acc_train: 0.9533 loss_val: 0.2040 acc_val: 0.9833 time: 0.0317s
Epoch: 0172 loss_train: 0.2666 acc_train: 0.9667 loss_val: 0.2044 acc_val: 0.9767 time: 0.0497s
Epoch: 0173 loss_train: 0.2899 acc_train: 0.9400 loss_val: 0.2039 acc_val: 0.9767 time: 0.0425s
Epoch: 0174 loss_train: 0.3005 acc_train: 0.9633 loss_val: 0.2028 acc_val: 0.9800 time: 0.0313s
Epoch: 0175 loss_train: 0.2754 acc_train: 0.9467 loss_val: 0.2023 acc_val: 0.9800 time: 0.0469s
Epoch: 0176 loss_train: 0.2610 acc_train: 0.9567 loss_val: 0.2023 acc_val: 0.9767 time: 0.0317s
Epoch: 0177 loss_train: 0.2831 acc_train: 0.9533 loss_val: 0.2028 acc_val: 0.9700 time: 0.0469s
Epoch: 0178 loss_train: 0.2685 acc_train: 0.9567 loss_val: 0.2030 acc_val: 0.9733 time: 0.0409s
Epoch: 0179 loss_train: 0.2930 acc_train: 0.9267 loss_val: 0.2028 acc_val: 0.9733 time: 0.0272s
Epoch: 0180 loss_train: 0.2614 acc_train: 0.9567 loss_val: 0.2025 acc_val: 0.9767 time: 0.0313s
Epoch: 0181 loss_train: 0.2881 acc_train: 0.9333 loss_val: 0.2024 acc_val: 0.9800 time: 0.0316s
Epoch: 0182 loss_train: 0.2753 acc_train: 0.9467 loss_val: 0.2027 acc_val: 0.9833 time: 0.0313s
Epoch: 0183 loss_train: 0.2572 acc_train: 0.9667 loss_val: 0.2029 acc_val: 0.9833 time: 0.0312s
Epoch: 0184 loss_train: 0.2569 acc_train: 0.9633 loss_val: 0.2031 acc_val: 0.9833 time: 0.0473s
Epoch: 0185 loss_train: 0.2984 acc_train: 0.9500 loss_val: 0.2031 acc_val: 0.9800 time: 0.0371s
Epoch: 0186 loss_train: 0.2790 acc_train: 0.9533 loss_val: 0.2030 acc_val: 0.9800 time: 0.0306s
Epoch: 0187 loss_train: 0.2774 acc_train: 0.9633 loss_val: 0.2027 acc_val: 0.9733 time: 0.0316s
Epoch: 0188 loss_train: 0.2825 acc_train: 0.9533 loss_val: 0.2030 acc_val: 0.9733 time: 0.0156s
Epoch: 0189 loss_train: 0.3023 acc_train: 0.9467 loss_val: 0.2035 acc_val: 0.9700 time: 0.0313s
Epoch: 0190 loss_train: 0.2919 acc_train: 0.9500 loss_val: 0.2037 acc_val: 0.9700 time: 0.0433s
Epoch: 0191 loss_train: 0.2615 acc_train: 0.9533 loss_val: 0.2033 acc_val: 0.9733 time: 0.0353s
Epoch: 0192 loss_train: 0.2684 acc_train: 0.9667 loss_val: 0.2026 acc_val: 0.9800 time: 0.0501s
Epoch: 0193 loss_train: 0.2721 acc_train: 0.9467 loss_val: 0.2022 acc_val: 0.9800 time: 0.0327s
Epoch: 0194 loss_train: 0.2464 acc_train: 0.9600 loss_val: 0.2024 acc_val: 0.9767 time: 0.0313s
Epoch: 0195 loss_train: 0.3115 acc_train: 0.9500 loss_val: 0.2026 acc_val: 0.9767 time: 0.0312s
Epoch: 0196 loss_train: 0.2695 acc_train: 0.9533 loss_val: 0.2025 acc_val: 0.9767 time: 0.0316s
Epoch: 0197 loss_train: 0.2414 acc_train: 0.9433 loss_val: 0.2025 acc_val: 0.9700 time: 0.0313s
Epoch: 0198 loss_train: 0.2699 acc_train: 0.9567 loss_val: 0.2025 acc_val: 0.9700 time: 0.0469s
Epoch: 0199 loss_train: 0.2724 acc_train: 0.9700 loss_val: 0.2030 acc_val: 0.9700 time: 0.0381s
Epoch: 0200 loss_train: 0.2914 acc_train: 0.9367 loss_val: 0.2029 acc_val: 0.9733 time: 0.0323s
Optimization Finished for 300 labeled nodes!
Total time elapsed: 6.1014s
Test set accuracy for 300 labeled nodes: 0.8340
```

In [ ]:

---------- **Problems finished** ----------

Code files where attached separately .