Graph Convolutional Network for Image Classification

Author: Kena Hemnani
Date: 16-10-2022

Introduction:

Graphs are almost everywhere. There is a lot of real life data that cannot be stored in a grid or formulated as a sequence example Social Network, Road maps, Molecule structure. Such data can be formulated as graphs.

Graph Neural Networks can be formulated for :
1)Learning the embeddings of each node. Example, a Node classification task.

In link , GCN is formulated as a node classification task to label the unlabelled data and performs well on sparsely labeled data for a variety of datasets like Citation Networks and ImageNet dataset.

2)Learning the embeddings of entire graphs. Example, Graph classification task.

Image Classification can also be formulated as a Graph classification task. Other examples are classifying whether protein is an enzyme or not in bioinformatics, Action prediction based on the skeleton of the body obtained from key points of the body.

3)Link prediction i.e, predicting the relationship between two nodes in a graph.

In Social Network analysis to infer the social interactions or to suggest friends etc. this is used.

Graph Convolutional Network:

In CNNs, Input is convolved with a set of weights i.e, kernels. Here convolution refers to the weighted sum of k x k pixels (k= kernel size). Hence, for getting output at the green pixel in Figure-2 (left side), information from the neighbourhood pixels is gathered and the weighted sum is computed. Similarly, Graph Convolutional Network(Interchangeably called GNN) aggregates information from the nodes connected to a given node.
Hence, Graph Convolutional Network is the generalized form of Convolutional Neural Network.

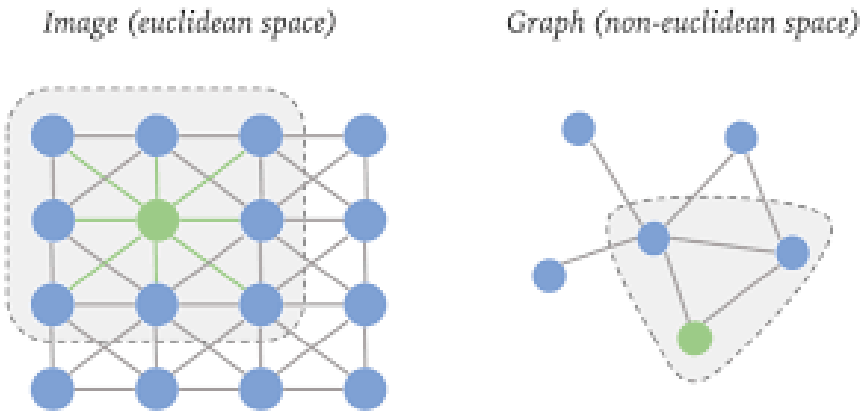


Figure-2 Image Ref:link

Let us say our graph has some initial feature vector, then the goal is to learn the embeddings of each node of a graph in lower dimensional space such that the information about how each node belongs to the context of the graph is captured.

GCN aggregates the information from the neighbor nodes and learns the embeddings of each node.

Consider a simple undirected graph with 3 nodes as shown in Figure-3:

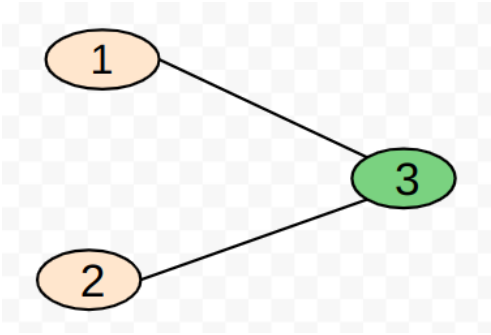


Figure-3

Consider a simple GCN with 2 hidden layer as shown in Figure-4

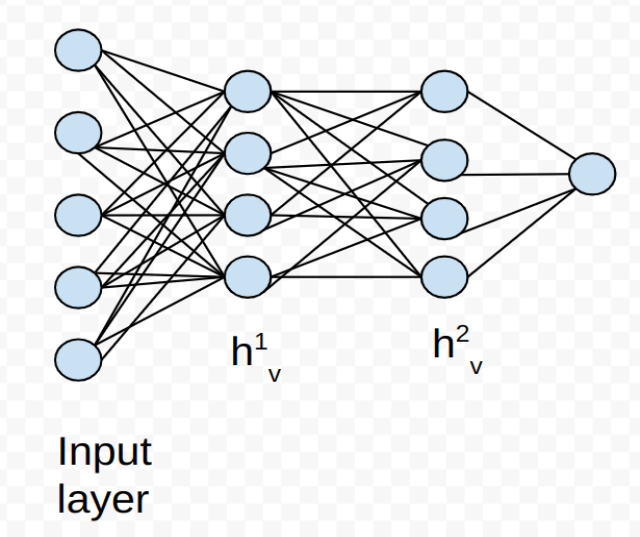


Figure-4

In Figure-4, h^1_v = output of hidden layer-1 for node v

h^2_v = output of hidden layer-2 for node v

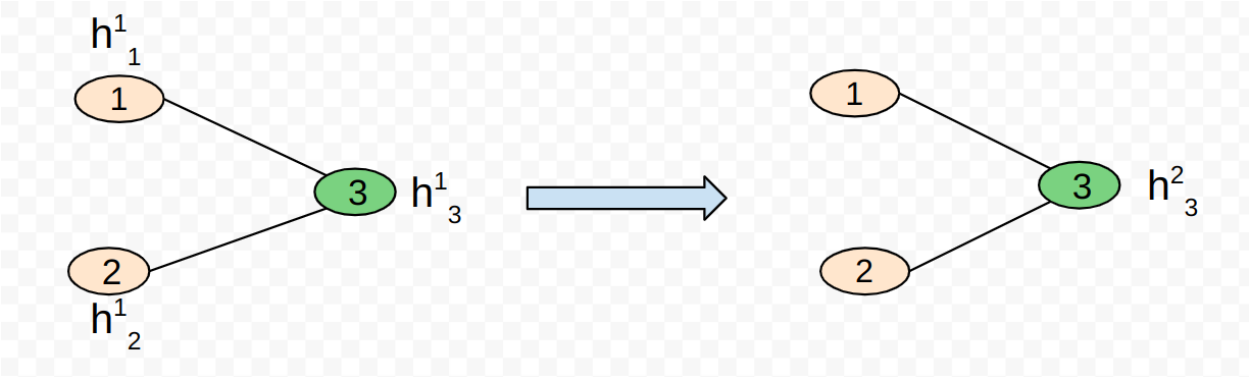


Figure-5

Let us assume that we already know the hidden layer output at layer-1 for all nodes and we want to find the output at hidden layer-2 for node 3 as shown in Figure-5 then,

$$h^2_3 = \sigma\left(\frac{1}{n_3+1} W^1 (h^1_1 + h^1_2 + h^1_3)\right)$$

where, n_3 = number of neighbours of node-3

W^1 = Weight parameter matrix between hidden layer-1 and 2

Hence, generalize layer-wise propogation rule is:

$$h_v^{l+1} = \sigma(\frac{1}{n_v + 1} W^l (h_v^l + \sum_{u \in N_v} h_u^l))$$

N_v = neighbour set of node-v
n_v = number of neighbours of node-v = $|N_v|$

Matrix formulation of the above update rule is:

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

$\tilde{A} = A + I_n$ is the adjacency matrix of the undirected graph G with added self-connections.
I_n is the identity matrix
$\tilde{D}$ is a diagonal degree matrix of graph G with added self-connections.

Implementation details:

Part-1 GCN:

Here, GCN Network has been formulated as a Node classification task for MNIST Handwritten dataset.
In the dataset there are 60000 training images and 10000 testing images belonging to one of the 10 classes.
Each image of 28 x 28 is flattened. Then 10 Nearest neighbours of each is computed using cosine similarity.

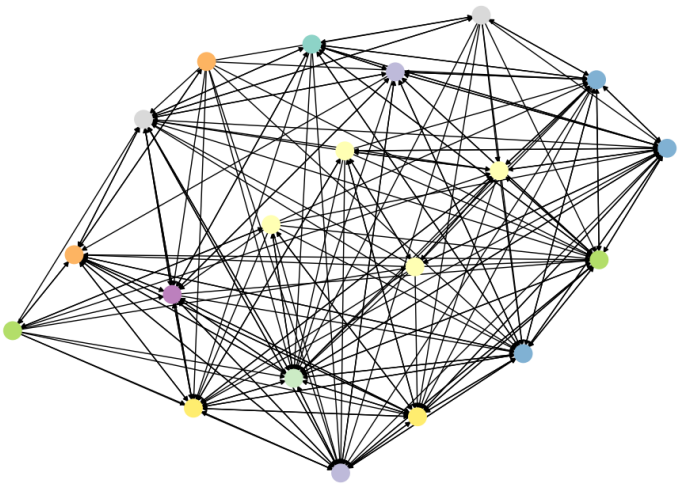
Let us assume a directed graph G(V,E),
$V = \{v_1, v_2, ..., v_i, ..., v_n\}$ = set of nodes in G. Here, it is set of images
$E = \{e_1, e_2, e_3, ..., e_n\}$ = set of edges. Here, $e_i = (v_i, v_j)$ if v_j is one of k nearest neighbors of v_i.

For this implementation k=10.

Data:
Our dataset is the entire graph formulated as explained above.

Data(edge_index=[2, 700000], num_nodes=70000, x=[70000, 784], y=[70000], num_classes=10, train_mask=[70000], test_mask=[70000])

Following is the visualization of the subset of this large graphs containing 20 nodes.



Model:

A Graph Convolutional Network with 4 GCN layers is implemented as shown below:

```
GCN(
  (conv1): GCNConv(784, 300)
  (RELU1): Sequential(
    (0): BatchNorm1d(300, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (1): LeakyReLU(negative_slope=0.1)
  )
  (conv2): GCNConv(300, 200)
  (RELU2): Sequential(
    (0): BatchNorm1d(200, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (1): LeakyReLU(negative_slope=0.1)
  )
  (conv3): GCNConv(200, 64)
  (RELU3): Sequential(
    (0): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (1): LeakyReLU(negative_slope=0.1)
  )
  (conv4): GCNConv(64, 10)
  (batchnorm4): BatchNorm1d(10, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
)
```

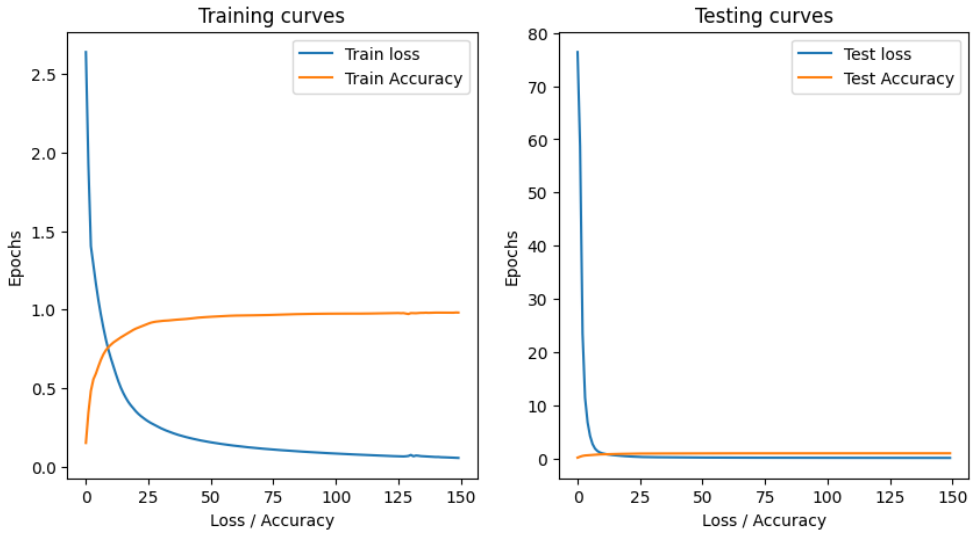
Hyperparameters:

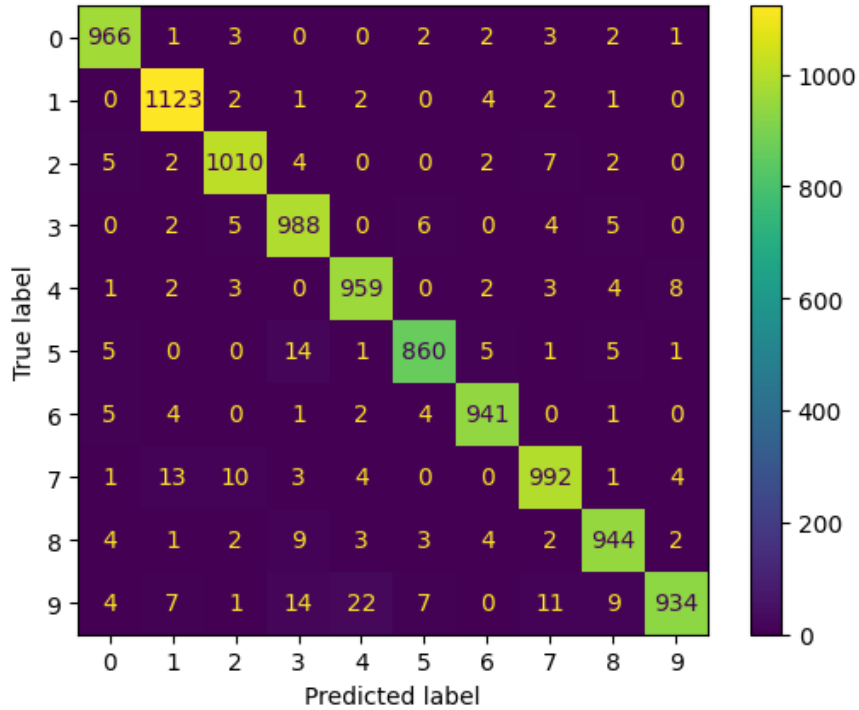
Model is trained for following hyperparameters using Adam Optimizer:

LEARNING_RATE = 1e-1
EPOCHS = 150

Results:

Training accuracy: 0.9813833333333334
Test Accuracy:0.9717


Training curves
Train loss
Train Accuracy
Epochs
Loss / Accuracy
Testing curves
Test loss
Test Accuracy
Epochs
Loss / Accuracy


True label
Predicted label
0 966 1 3 0 0 2 2 3 2 1
1 0 1123 2 1 2 0 4 2 1 0
2 5 2 1010 4 0 0 2 7 2 0
3 0 2 5 988 0 6 0 4 5 0
4 1 2 3 0 959 0 2 3 4 8
5 5 0 0 14 1 860 5 1 5 1
6 5 4 0 1 2 4 941 0 1 0
7 1 13 10 3 4 0 0 992 1 4
8 4 1 2 9 3 3 4 2 944 2
9 4 7 1 14 22 7 0 11 9 934
0 1 2 3 4 5 6 7 8 9
1000
800
600
400
200
0

Part-2 CNN:

Model:

A Convolutional Neural Network with 4 CNN layers is implemented as shown below:

```
CNN(
  (CNN1): Sequential(
    (0): Conv2d(1, 10, kernel_size=(3, 3), stride=(1, 1), padding=same)
    (1): BatchNorm2d(10, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): LeakyReLU(negative_slope=0.1)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (CNN2): Sequential(
    (0): Conv2d(10, 30, kernel_size=(3, 3), stride=(1, 1), padding=same)
    (1): BatchNorm2d(30, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): LeakyReLU(negative_slope=0.1)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (CNN3): Sequential(
    (0): Conv2d(30, 50, kernel_size=(3, 3), stride=(1, 1), padding=same)
    (1): BatchNorm2d(50, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): LeakyReLU(negative_slope=0.1)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (CNN4): Sequential(
    (0): Conv2d(50, 100, kernel_size=(3, 3), stride=(1, 1), padding=same)
    (1): BatchNorm2d(100, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): LeakyReLU(negative_slope=0.1)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (Linear): Linear(in_features=100, out_features=10, bias=True)
)
```

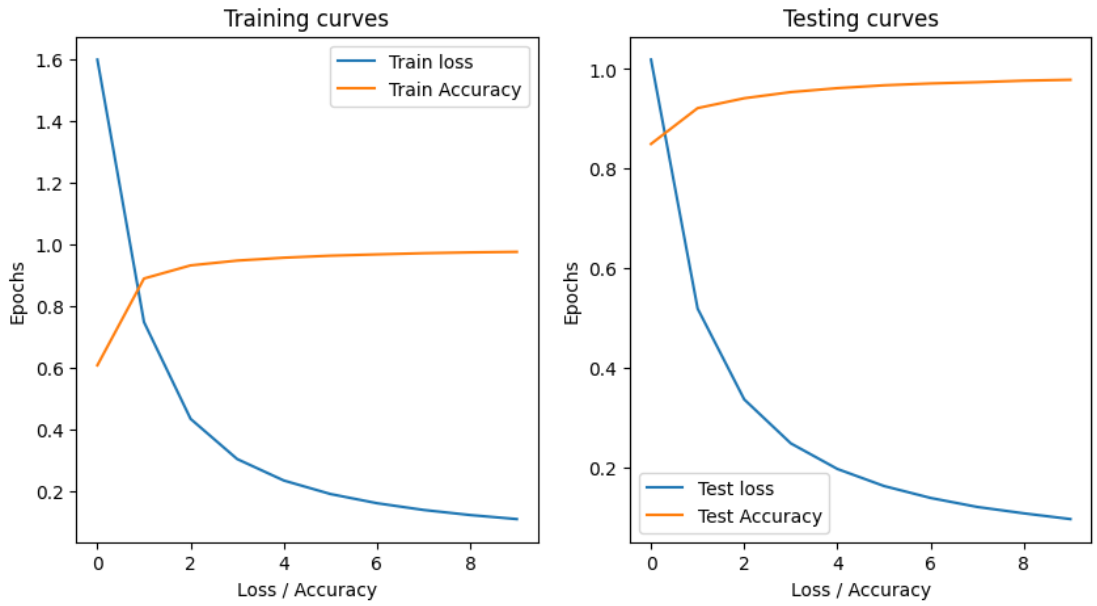
Hyperparameters:

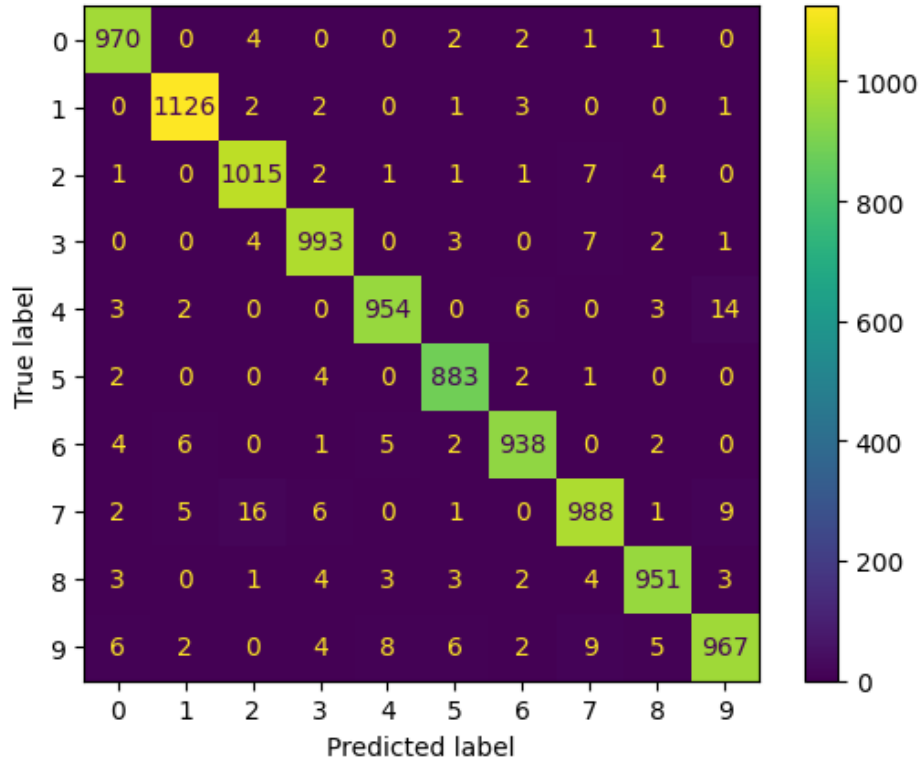
Model is trained using below Hyperparameters using Adam Optimizer.

BATCH_SIZE = 200
LEARNING_RATE = 2e-5
WEIGHT_DECAY = 1e-4
EPOCHS = 10

Results:

Test loss: 0.0965411439538002
Test accuracy: 0.9785000085830688





Comparison:

Model	Train Accuracy	Test Accuracy
CNN	0.9755	0.9785
GCN	0.9813	0.9717

For this particular dataset and Task, both the CNN and GCN seem performing equally well.

The code for above Implementation of GCN and CNN can be found here: https://github.com/KenaHemnani/gcn-image-classification-task/tree/master

References:

https://arxiv.org/pdf/1609.02907.pdf

https://www.youtube.com/watch?v=Vz5bT8Xw6Dc

https://pytorch-geometric.readthedocs.io/en/latest/notes/colabs.html

https://towardsdatascience.com/a-beginners-guide-to-graph-neural-networks-using-pytorch-geometric-part-1-d98dc93e7742