Lab: Graph Neural Networks (GNN)

Instructor: Fragkiskos Malliaros
TA: Vahan Martirosyan

January 22, 2026

Description

In this lab, you will learn to use a powerful representation learning family: *Graph Neural Networks*. In particular, you will implement a Graph Convolution Network (GCN) [1] to solve both a node classification task and a graph classification task..

Description

Graph Neural Networks (GNNs) are powerful deep learning architectures for representation learning of graph data. They achieve state-of-the-art results in a wide variety of tasks by deriving very informative node embeddings that incorporate both graph structure and node feature information. This is due to their specific functioning, expressed as a recursive message passing scheme, where they encode information from nodes and pass it along the edges of the graph.

For the specific case of **Graph Convolution Networks**: we consider a graph $\mathcal{G}$ with feature matrix $\mathbf{X}$, adjacency matrix $\mathbf{A}$, and diagonal degree matrix $\mathbf{D}$. Let N be the number of nodes, C the number of input features, F of output features, and $\mathbf{Z}$ the output. Note that, $\mathbf{Z} \in \mathbb{R}^{N \times F}$, $\mathbf{X} \in \mathbb{R}^{N \times C}$, $\mathbf{W} \in \mathbb{R}^{C \times F}$, where $\mathbf{W}$ is a weight matrix.

The output of one GCN layer is obtained as follows:

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

with $H_0 = X$, $H_L = Z$ and σ often chosen to be the ReLU function. Also, $\tilde{A} = A + I$ and $\tilde{D} = D + I$ (adding self loops). Each element in $H_{(l+1)}$ can thus be written as $h_i^{(l+1)} = \sigma(\Sigma_{j \in N(i)} \frac{1}{c_{ij}} W^{(l)} h_j^{(l)})$.

Part I: Node Classification

The dataset, Amazon Photo, is a segment of the Amazon co-purchase graph. Nodes represent goods and edges indicate that two goods are frequently bought together. Node features are bag-of-words encoded product reviews, and class labels are given by the product category.

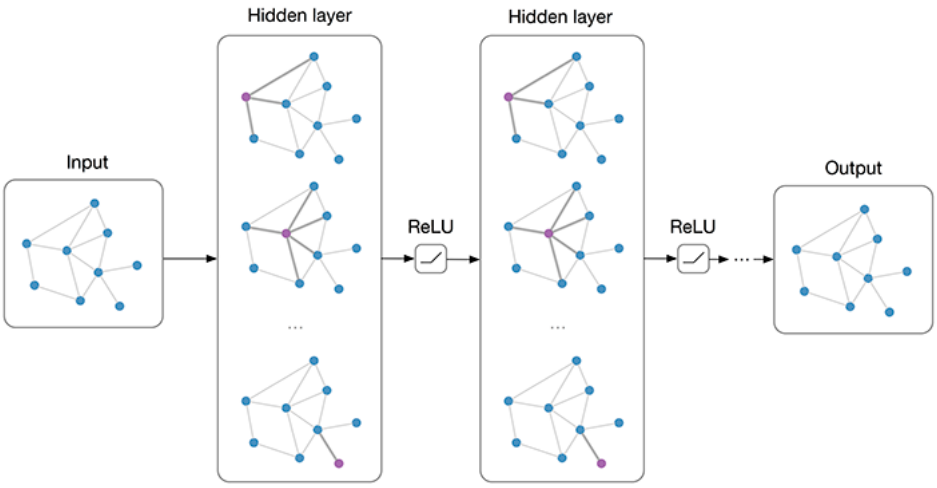

Figure 1: Multi-layer Graph Convolutional Network (GCN)

Exercise 1: Pre-processing

Before you start, make sure you have the required packages (e.g. DGL[1], torch, sklearn) installed.

1. Make sure the dataset is correctly imported in the main function. You will be using the popular Deep Graph Library (DGL) to implement a GNN architecture. DGL makes it easy to implement GNNs since it provides many state-of-the-art GNN layers and modules. You should retrieve different key properties: the feature matrix X, the label vector y, the number of classes, the number of features and the number of nodes. Careful, the graph is not a networkx instance and commands are slightly different for DGL graphs...
2. Use the `split_dataset()` method to divide the graph into training/validation/test sets for node classification.

Now that the data is pre-processed, let's define, train and evaluate a GNN model on it.

Exercise 2: Learning

1. Construct a Graph Convolution Network (GCN) inside the `GNN_model` class. You can follow the architecture in figure 1 You do not have to redefine it from scratch. GCN blocks are already defined inside the dgl function `GraphConv`. *Note*: We use the pytorch backend by default but dgl also operates with tensorflow.
2. Define an optimizer, a loss function and train the GCN model inside `train()`.
3. Although it is a great approach, GCN has some potential limits. Could you think of some? For instance, what could happen if we stack a lot of GCN layers? You can use the interactive visualization of GCN available at: `https://distill.pub/2021/understanding-gnns/#interactive`

Part II: Graph Classification

In this section, we will follow a similar idea but for a graph classification task. Instead of having a single graph where you want to assign a label to nodes, you have a set of graphs, and would like to assign

[1]https://docs.dgl.ai/#

a label to each one of them. For instance, in biology, we have the so-called protein-protein interaction networks. Each protein of is represented as a graph, and we would like to infer the function of this protein.

Exercise 3

1. Load the following datase `dgl.data.TUDataset(name='ENZYMES')`. Create a training, validation and test set (Careful the dataset format is particular).
   You can use `dgl.data.utils.split_dataset` to split the dataset.
   Finally, use the `GraphDataLoader` function to batch graphs.
2. Create a GNN model to perform the graph classification task. Reminder: we want to derive a graph representation, not just node representation.
3. Train this new model and evaluate it, similarly to part I.

References

[1] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.