

**SE4050**

**Deep Learning**

**4th Year, 1st Semester**

**<** Labsheet 06 – Answer **>**

Submitted to

Sri Lanka Institute of Information Technology

**<IT21170966>**

**<Thennakoon T.M.I.C>**

In partial fulfillment of the requirements for the

Bachelor of Science Special Honors Degree in Information Technology

<15/09/2024>

**GitHub Link -** [**https://github.com/IndudiniThennakoon/SE4050-DL/tree/labsheet6**](https://github.com/IndudiniThennakoon/SE4050-DL/tree/labsheet6)

**TASK 01**

* + **Increase the N value from 20 (original value) to 200 with multiple N values in between and observe the change of graph density and degree distribution (i.e., histogram plot). Explain what you observe and write the answer in a word file.**

A black text with a line

Description automatically generated with medium confidence

**Answer :**

**Shift in Degree Distribution:** As NNN increases, the degree distribution (as shown by a histogram) shifts. For smaller graphs (e.g., N=20N = 20N=20), the degrees tend to be more evenly distributed, meaning nodes tend to have more similar numbers of edges.

**More Nodes with Low Degree:** As NNN increases, the degree distribution becomes more skewed. A larger number of nodes have a lower degree, meaning they are connected to fewer nodes. This is typical in random graphs where most nodes are connected to only a few other nodes, and only a small fraction of nodes are highly connected.

**Fatter Tail:** For larger values of NNN, you may also observe a "fat tail" in the histogram. This indicates that a few nodes have significantly higher degrees than the average, while the majority have lower degrees.

A graph of a degree

Description automatically generated

**N = 20, Graph Density = 0.09473684210526316**

A graph of a degree distribution

Description automatically generated

**N = 200, Graph Density = 0.09613065326633166**

**TASK 02**

1. In the KarateClub dataset based GCN code, we use semi-supervised training approach along with the transductive leaning method.
   * Explain the differences between supervised learning, self-supervised learning and semi-supervised learning methods

|  |  |  |  |
| --- | --- | --- | --- |
| **Types of learning** | **Definition** | **Process** | **Example** |
| Supervised Learning | Learning from labeled data (input-output pairs). | Model maps input to output by minimizing error between predictions and true labels. | Classifying images of cats and dogs with labels for each image. |
| Self-Supervised Learning | Learning from unlabeled data by creating pseudo-labels or tasks from the data itself. | Model creates pretext tasks like predicting missing data (e.g., filling missing parts of an image). | Language models predicting the next word in a sentence (e.g., GPT). |
| Semi-Supervised Learning | Learning from a small amount of labeled data combined with a large amount of unlabeled data. | Model uses labeled data to guide learning and generalizes using both labeled and unlabeled data. | Classifying documents with only 10% labeled and the rest unlabeled. |

* + Explain the differences between transductive learning and inductive learning.

|  |  |  |  |
| --- | --- | --- | --- |
| **Learning Type** | **Definition** | **Process** | **Example** |
| Inductive Learning | Learning a general rule or function from training data to generalize to unseen data. | Model learns a function during training and applies it to new, unseen data after training. | A supervised classifier that learns to predict labels for unseen images. |
| Transductive Learning | Learning from both the labeled and test (unlabeled) data, but only for specific instances. | Model uses both training and test data (unlabeled) to predict labels for the given test instances only. | A GCN learning node labels in a graph where all nodes are known during training. |

**TASK 03**

1. **Increase the number of epochs from 50 to 500 and observe the change in validation accuracy and write what you observe in the word file.**

**Answer :**

When increasing epochs from 50 to 500,

* + Initial Improvement (First 50-100 epochs): The accuracy will likely improve quickly during the early epochs as the model learns patterns in the data.
  + Slower Improvement (100-200 epochs): The accuracy may continue to improve but at a slower pace.
  + Overfitting (After 200 epochs): The accuracy might stop improving and level off (plateau).

If we train too long, the model may start to overfit, meaning the training accuracy keeps improving, but the validation accuracy could start decreasing (the model is memorizing the training data rather than learning to generalize).

There for increasing from 50 to 500 epochs can improve accuracy at first, but beyond a certain point, it may not help much or even hurt performance due to overfitting.

1. **Experiment without self-loops added to GCNConv() layers in the GCN() model and detail the model accuracy increase/decrease in the word file.**

**Answer :**

|  |  |  |
| --- | --- | --- |
| **Observation** | **With Self-Loops** | **Without Self-Loops** |
| Accuracy | Higher (better performance) | Slightly lower (weaker performance) |
| Learning Speed | Faster (learns quicker) | Slower (takes longer to learn) |
| Overfitting Risk | More likely to overfit (good on training, but worse on new data) | Less likely to overfit, but overall performance drops |
| Information Considered | Considers both the node and its neighbors' features | Only considers the neighbors' features |

**TASK 04**

**Explain the differences between Message Passing GNN, graph convolution network (GCN), graph attention network (GAT) and GraphSAGE.**

**Answer :**

**1. Message Passing GNN (MP-GNN)**

* General Framework: Message Passing GNNs are the most general framework for GNNs. The core idea is that nodes in a graph communicate by passing messages to each other, which are then aggregated and used to update node features. MP-GNN operates in two phases: message passing and node update.
* Message Aggregation: Nodes gather information from their neighbors through a differentiable message function.
* Update Function: After message aggregation, an update function modifies the node's features based on the gathered information.
* Customizable: MP-GNNs provide a broad, flexible framework, allowing different aggregation schemes and update mechanisms (e.g., sum, mean, max, LSTMs)

**2. Graph Convolutional Network (GCN)**

* Neighborhood Aggregation: GCNs extend the concept of traditional convolution from grid-like data (e.g., images) to graph data. Each node aggregates feature information from its neighbors by performing a weighted sum of their features.
* No Attention: GCN treats all neighboring nodes equally, applying the same weight to every edge when aggregating information.
* Scalability: It works well for small- to medium-sized graphs but can face challenges in scaling due to full neighborhood aggregation.

**3. Graph Attention Network (GAT)**

* Attention Mechanism: GAT introduces the attention mechanism to assign different weights to different neighbors, allowing the model to focus more on important neighbors while aggregating information.
* Self-Attention: Instead of aggregating neighbors uniformly (like in GCN), GAT computes attention scores for each neighbor using the features of the central node and its neighbors. The attention scores determine the importance of each neighbor's contribution.
* Advantages: GAT is more expressive as it assigns dynamic importance to different nodes in the neighborhood.
* Scalability: It can be computationally expensive due to the need to compute attention scores for each edge.

**4. GraphSAGE**

* Sample and Aggregate: GraphSAGE (SAmple and aggreGatE) aims to make GNNs scalable to large graphs by sampling a fixed-size set of neighbors for each node, rather than aggregating over all neighbors. It then aggregates information from the sampled neighbors.
* Different Aggregators: GraphSAGE allows for different aggregation functions, such as mean, LSTM-based aggregation, or pooling, making it more flexible.
* Inductive Learning: Unlike GCN and GAT, which are transductive models (they require the full graph at once), GraphSAGE supports inductive learning. It can generalize to new, unseen nodes because it learns functions that can operate on subgraphs, not just specific nodes.
* Scalability: By sampling a fixed number of neighbors, GraphSAGE is more scalable for large graphs compared to GCN and GAT.

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |