DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

HW3

Abhinav Barnawal   *Shreyansh Singh*   *Si Siddhanth Raja*
: *2020CS50415*   *2020CS10385*   *2020CS50443*
Class: *2301-COL761*
Session: *2023-24*
: *cs5200415@iitd.ac.in*   *cs1200385@iitd.ac.in*   *cs5200443@iitd.ac.in*

Course: *Data Mining* – Instructor: *Prof. Sayan Ranu*
Submission date: *November 26, 2023*

Contents

Q1

We uniformly sampled `10,00,000` points of six dimensions `[1, 2, 4, 8, 16, 32, 64]` used `numpy` library to calculate the three types of distances (L_1, L_2, L_{inf}), and the result is reported in Figure 1. To reproduce the result, migrate to the `A3` folder and run the following command:

```
bash interface1.sh
```

We obtained the following plot:

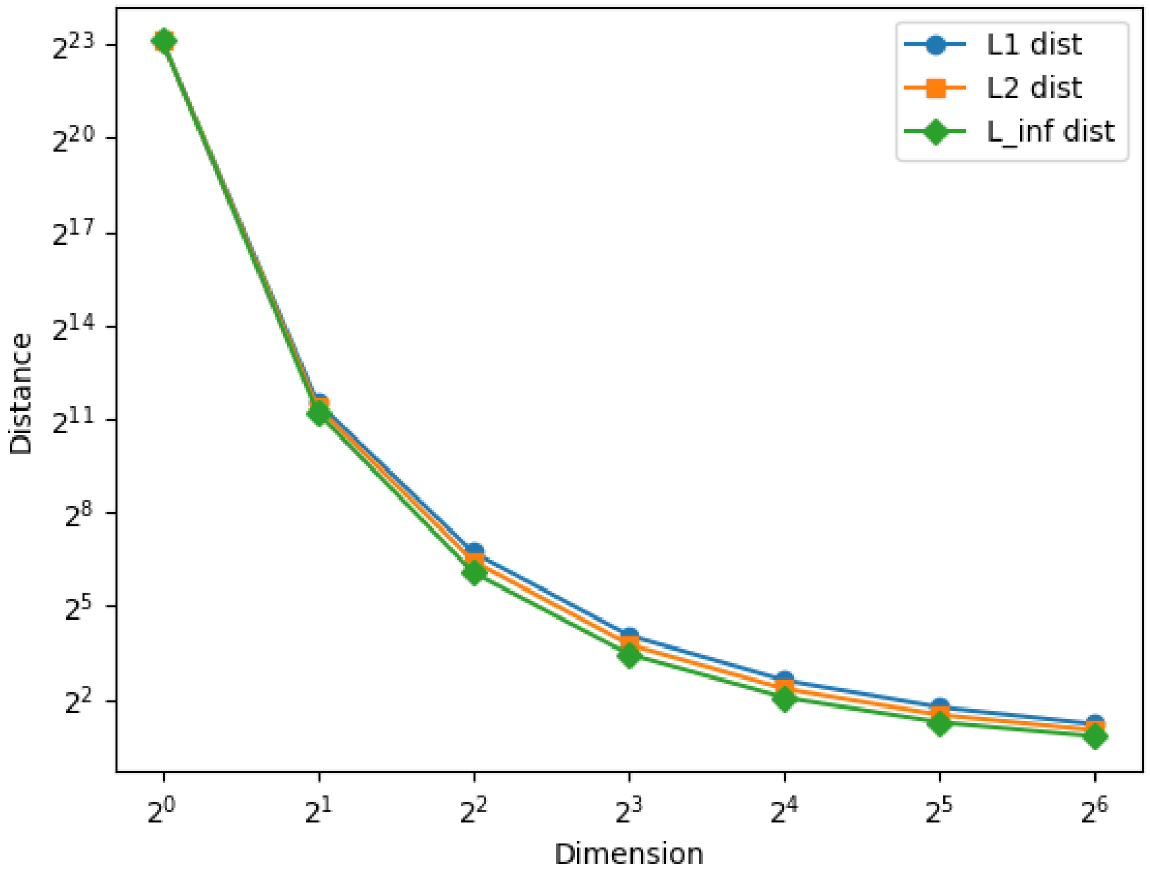


Figure 1: Ratio of the maximum distance to the minimum distance between two points

We observe that the average ratio of the maximum distance to the minimum distance of a sample of 100 points from all the other points in the dataset decreases drastically from 2^{23} to $\sim 2^1$ as we increase the dimension from 1 to 64. This depicts the famous `Curse of Dimensionality`. This happens because the volume of the space grows exponentially with the dimension since the volume of a hyper-cube of edge length l is l^d. So, the same number of points spread out in a larger space to become sparser. Hence, the ratio of maximum to minimum distance from a point decreases.

Q2

Classification

Below, we present our approach to the classification task. With the best model, we could achieve an `ROC-AUC` score of 0.77 on the validation dataset.

Layer Selection and Architecture:

1. **GINEConv Layers:** We have chosen GINEConv as the primary convolutional layer. Our model starts with two layers of GINEConv Layers. GIN is known for its ability to learn structural information from the graph. Utilizing GINEConv allows the model to effectively aggregate and process information from neighboring nodes. The sequential modules within GINEConv consisting of Linear, ReLU, and Linear layers capture nonlinearities and higher-order relationships in the graph. The increment in hidden channels (from 64 to 256) aims to capture increasingly abstract and high-level graph representations.

2. **Fully Connected (Linear) Layers:** After GINEConv, we have used two fully connected layers. These layers serve as the graph's feature transformation to a representation suitable for classification. ReLU activation functions are employed after each fully connected layer to introduce non-linearity and aid in feature extraction. Starting with 256 and gradually decreasing to 32, it aims to condense the learned information into a lower-dimensional space while retaining important graph characteristics.

3. **Activation Functions:** We have chosen ReLU as the activation function due to its simplicity and effectiveness in overcoming the vanishing gradient problem commonly encountered in deep neural networks. It facilitates faster convergence during training by allowing only positive values to pass through, aiding the model's capacity to learn complex nonlinear relationships within the graph data.

4. **Dropout Layer:** Dropout with a probability of 0.6 is applied to the fully connected layers during training to prevent overfitting by randomly dropping a fraction of the neurons. Dropout helps improve the model's generalization by reducing reliance on specific neurons, making the model more robust.

5. **Output Layer:** The final linear layer with a sigmoid activation function produces the binary classification output. The sigmoid activation function squashes the output to a range between 0 and 1, interpreting it as the probability of the graph belonging to a certain class.

Implementation Details

1. **Node and Edge Encoder:** We have tried encoding the node and edge features using the Encoder provided. Since the GCN layer doesn't take edge attributes as input, we tried to concatenate the encoded edge features to the node features of both of its endpoints. We also modified the Encoder class to use different aggregation functions and concatenation operations instead of the simple summation

over the vectors representing each feature. None of these experiments resulted in a performance gain.

2. **Preprocessing:** On close inspection of the node and edge features, we observed that the different features have different distributions. Some had a small range, while others had a larger range, with some features taking only binary values. We concluded that some node features are categorical while some are numerical, indicating the properties of the atoms in a molecule. Similarly, the edge features may represent the properties of the bonds between atoms.
   So, we decided on normalizing the numerical features and one-hot encoding the categorical features to prepare the data for input to the GNN model. The motivation behind this preprocessing is to convert the data into a format that can be used by machine learning algorithms. Normalizing the numerical features scales them to a similar range, which can help the learning algorithm converge faster.
   However, even these techniques did not result in any significant gains.

Evaluation and Reasoning:

After exploring various convolutional layers such as GATConv and GCNConv, experimenting with multiple activation functions, and iteratively adjusting the dimensions of hidden layers, the architecture presented represents the culmination of our efforts. This refined model emerged as the most effective for the task at hand. The decision to settle on this specific architecture was rooted in extensive evaluation and iterative refinement. The selected design showcased superior performance in learning intricate graph features and capturing complex relationships within the data. This model demonstrated remarkable proficiency through systematically exploring various architectures and hyperparameters, validating its selection as the most optimized configuration among the alternatives considered.

Results

We also implemented Linear Regression as the baseline model and compared the performance of our model against the baseline model and the random model. The results are as shown below:

Figure 2 shows that the validation loss decreases till ~ 80 epochs and then increases. According to our analysis, the best state of our model is obtained at the 78^{th} epoch, which correlates with the observed behavior. However, beyond 80 epochs, the training loss decreases while the validation loss increases, indicating over-fitting. So, we save our best model at the 78^{th} epoch.

The Gradient Norm generally increases throughout training (See Figure 4). This is contrary to our expectations that the norms of the gradients would decrease as the training process converges to a critical point. Despite the increasing gradient, the training process is reasonably successful as the validation loss decreases to a low level around the 78^{th} epoch. This is known as the Ill-Conditioning problem.

Our model gives the best $ROC - AUC$ score compared to the other models as shown in Figure 3. However, since the dataset is skewed with four graphs of class 0 for every graph of class 1, our model generalizes well to classify 0-class graphs as 0 but fails to classify 1-class graphs as 1. We tried undersampling to tackle the issue but could not achieve significant improvement.

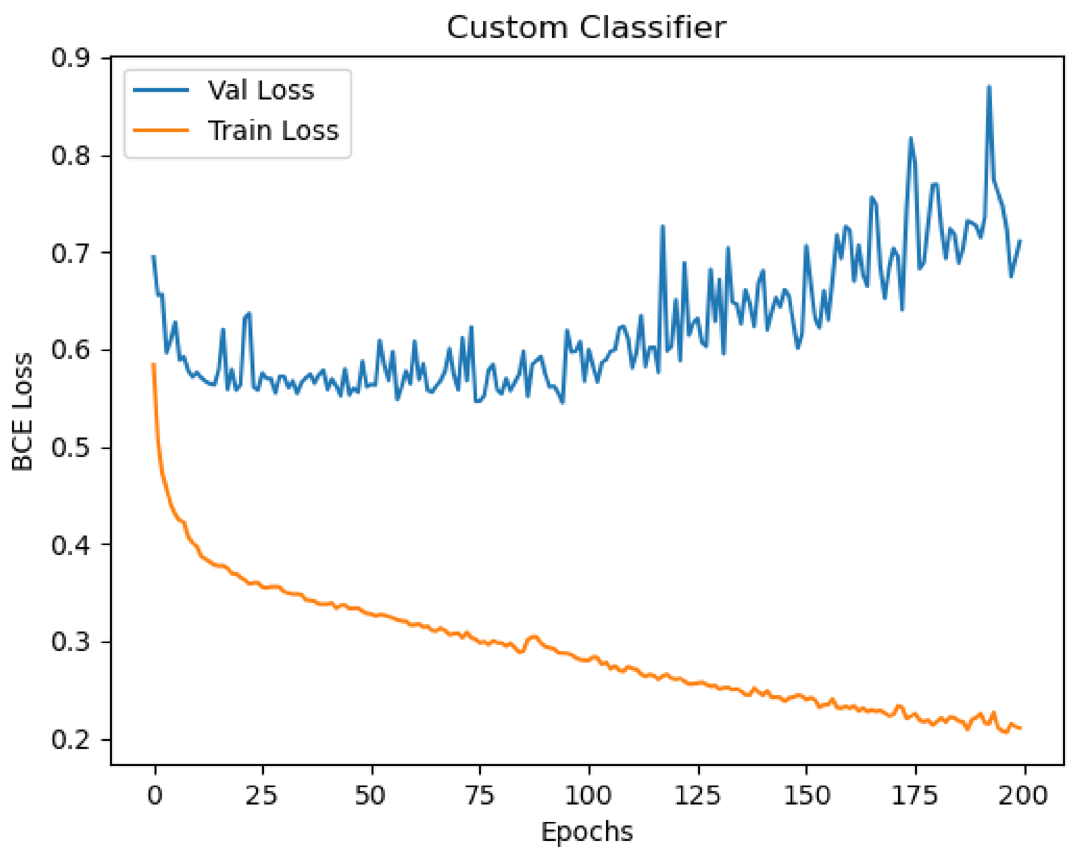


Figure 2: Learning Curve for Custom Classifier over 200 epochs

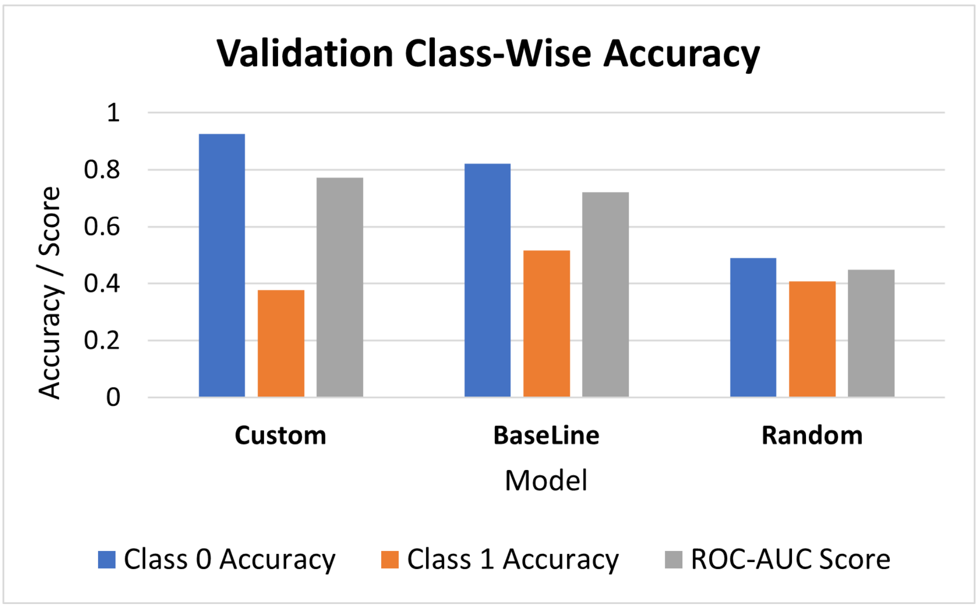


Figure 3: Showing Relative performance of our model in comparison to the baseline and the random models

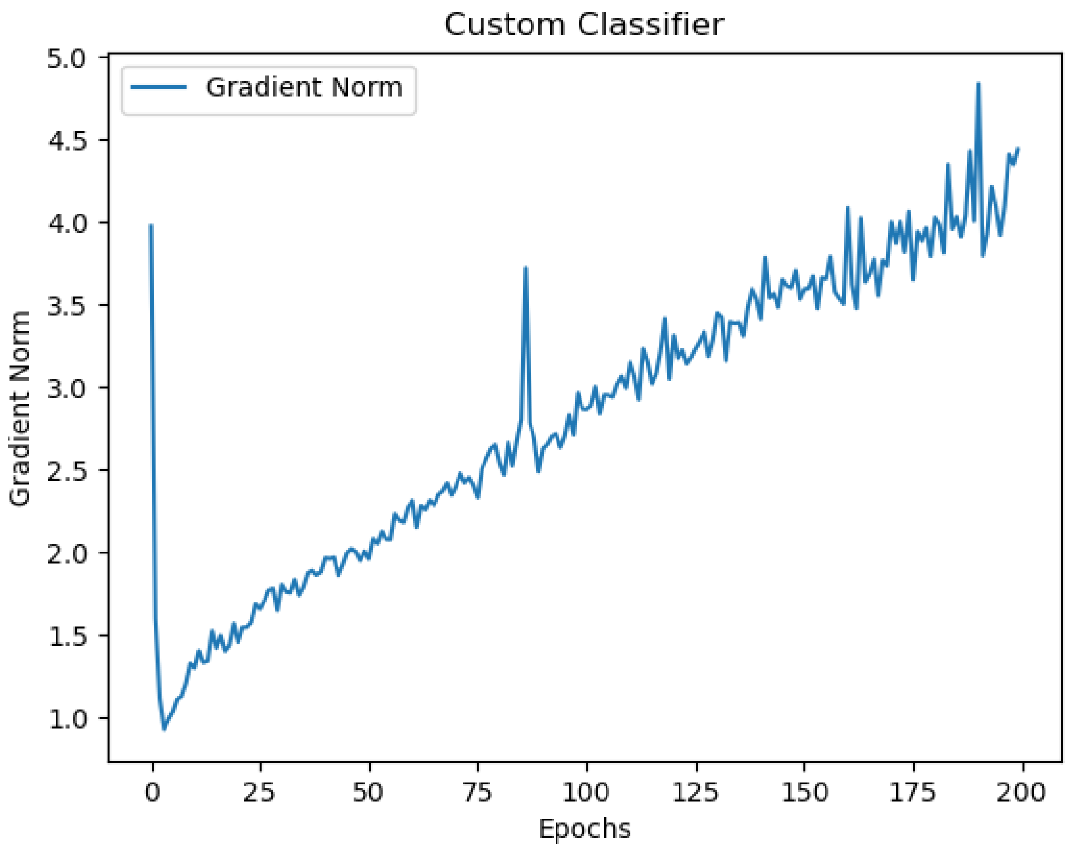


Figure 4: Gradient-Norm Curve for Custom Classifier over 200 epochs

Regression

Below, we present our approach to the regression task. The best model could achieve an $RMSE$ loss of 0.75 on the validation dataset.

Layer Selection and Architecture:

1. **GINEConv Layers:** First, we have 2 GINEConv layers. They are used to perform message passing in Graph Isomorphism Network (GIN). They incorporate information from neighboring nodes and edges, which is crucial for graph-based tasks. The choice of GINEConv layers with sequential linear operations allows the model to capture complex graph structures by transforming node features through multiple layers. The increment in hidden channels (from 32 to 64 to 128 to 256) aims to capture increasingly abstract and high-level graph representations.

2. **Fully Connected (Linear) Layers:** After GINEConv, we have used three fully connected layers. These layers process the learned representations from the GINConv layers to gradually reduce dimensionality and extract relevant features. Starting with 256 and decreasing gradually down to 32, it aims to condense the learned information into a lower-dimensional space while retaining important graph characteristics.

3. **Activation Functions:** We have used LeakyReLU activation functions, which introduce non-linearity, allowing the model to learn complex patterns in the data. LeakyReLU is applied after each linear operation, promoting the flow of information and mitigating the vanishing gradient problem.

4. **Dropout Layer:** We have used a dropout layer with a probability of 0.5 to prevent overfitting by randomly dropping connections between fully connected layers during training.

5. **Output Layer:** The final linear layer (self.regressor) maps the condensed features to the output dimension (1), predicting the continuous numerical value associated with the graphs.

Implementation Details

1. **Node and Edge Encoder:** We have tried encoding the node and edge features using the Encoder provided. Since the GCN layer doesn't take edge attributes as input, we tried to concatenate the encoded edge features to the node features of both of its endpoints. We also modified the Encoder class to use different aggregation functions and concatenation operations instead of the simple summation over the vectors representing each feature. None of these experiments resulted in a performance gain.

2. **Preprocessing:** On close inspection of the node and edge features, we observed that the different features have different distributions. Some had a small range, while others had a larger range, with some features taking only binary values. We concluded that some node features are categorical while some are numerical, indicating the properties of the atoms in a molecule. Similarly, the edge features may represent the properties of the bonds between atoms.
   So, we decided on normalizing the numerical features and one-hot encoding the categorical features to prepare the data for input to the GNN model. The motivation behind this preprocessing is to convert the data into a format that can be used by machine learning algorithms. Normalizing the numerical features scales them to a similar range, which can help the learning algorithm converge faster.
   However, even these techniques did not result in any significant gains.

Evaluation and Reasoning:

Our model development journey involved thorough experimentation with various layer configurations and sizes. We embarked on an iterative process, exploring multiple architectures to determine the most effective design for our regression task. This approach aimed to optimize the model's ability to understand and predict continuous numerical values associated with graphs. Through rigorous experimentation and validation, we navigated through different combinations of layer types, sizes, and activation functions.

After a series of systematic evaluations, the architecture presented in our custom regressor model emerged as the most optimal. It was selected based on its ability to effectively process graph data, gradually distilling crucial information from the nodes and edges.

Results

We also implemented Linear Regression as the baseline model and compared the performance of our model against the baseline model and the random model. The results are as shown below:

The validation loss stabilizes at ~ 200 epochs. According to our analysis, the model's performance does not change much from 200 to 300 epochs despite minor improvements, which is also indicated by the fact that validation loss is not yet increasing.

Our model achieves an $RMSE$ of ~ 0.75, much lower than those achieved by the baseline and the random regressors.

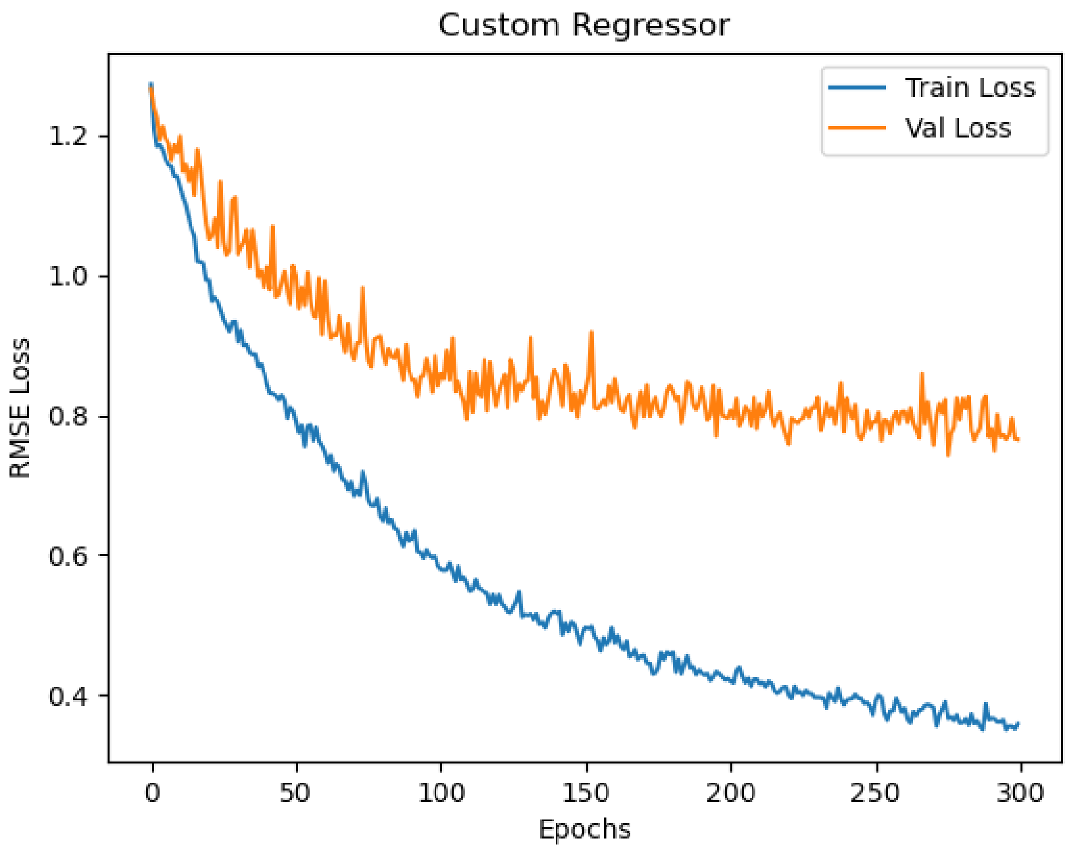


Figure 5: Learning Curve for Custom Regressor over 300 epochs

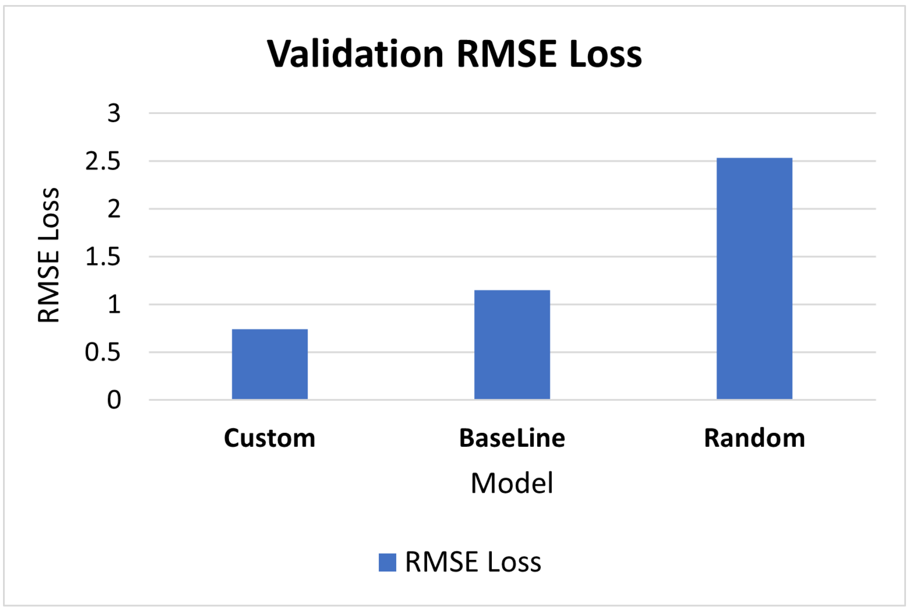


Figure 6: Showing Relative performance of our model in comparison to the baseline and the random models

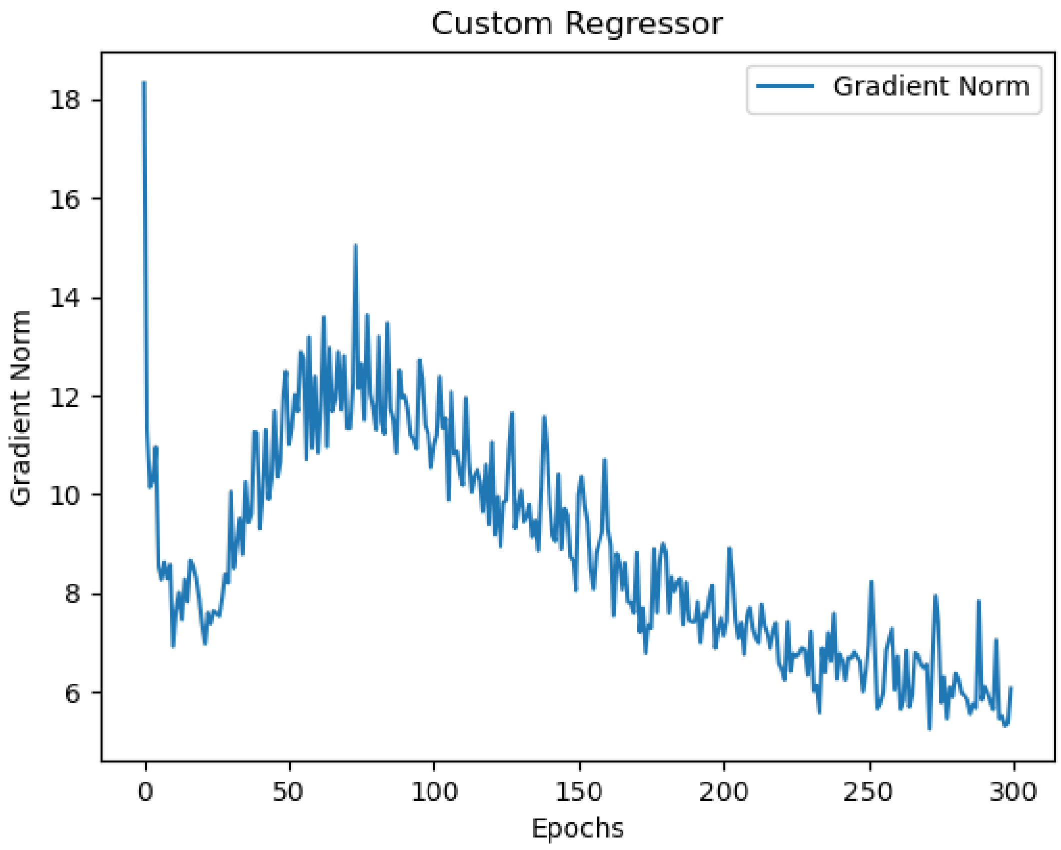
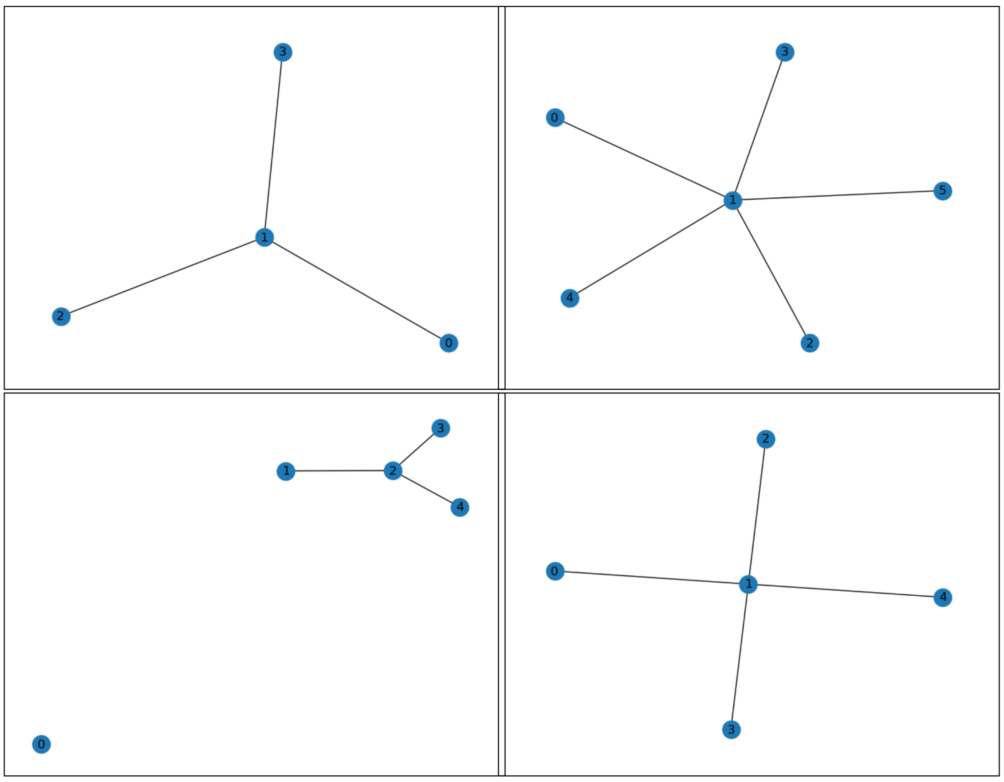


Figure 7: Gradient-Norm Curve for Custom Regressor over 300 epochs

Visualisation

On careful inspection of the graphs misclassified by the model, we observed that most of the misclassified graphs were of the following kind:



This happens because node embeddings of such graphs generated by GNNs do not collect enough information of the other nodes which makes it hard to learn.

Acknowledgement

Our sincere thanks to Prof. Sayan Ranu for his expertise and guidance that led to the completion of this work. We also thank the respected Teaching Assistants for their immense support and almost instant replies to our doubts, without whom it would have never been possible to present this work.