CSL7870 Assignment 2
Nitish Bhardwaj
B21AI056

- **Data Preprocessing**
  - Cloned the CoDEx dataset from github, set the size argument as ‘s’ for codex-s and code as ‘en’ for retrieving the english dataset.
  - Using the load_triples function created my train, valid and test dataset.
  - Each had three columns: head, relation and tail. The data contained were ids starting with ‘Q’ for head and tail, with ‘P' for relation followed by number.

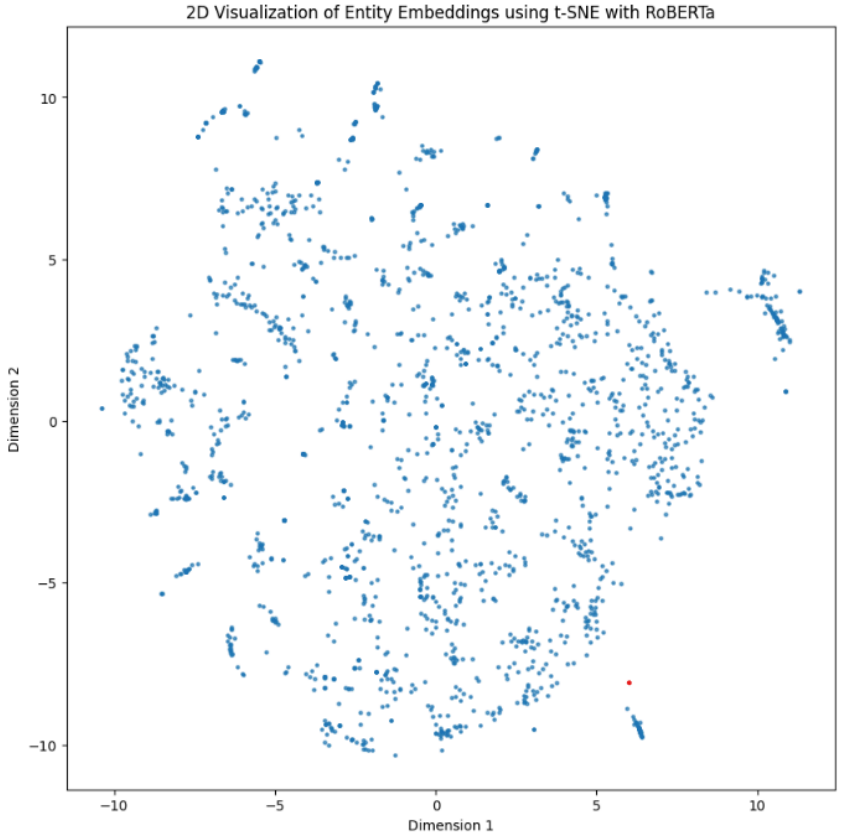
	head	relation	tail
0	Q7604	P1412	Q188
1	Q78608	P509	Q12078
2	Q739	P463	Q656801
3	Q192279	P1412	Q7737
4	Q55	P463	Q1969730

  - It had 2034 unique entities and 42 relations.
  - One can use the load_entities and load_relations function to search the actual content available to these ids.
  - Content consist of ‘wiki’ which is wikipedia link (only for head and tail ids), ‘description’ and ‘label’

```
{'wiki': 'https://en.wikipedia.org/wiki/Leonhard_Euler', 'description': 'Swiss mathematician', 'label': 'Leonhard Euler'}
{'label': 'languages spoken, written, or signed', 'description': 'language(s) that a person speaks, writes or signs, including the native language(s)'}
{'wiki': 'https://en.wikipedia.org/wiki/German_language', 'description': 'West Germanic language', 'label': 'German'}
```

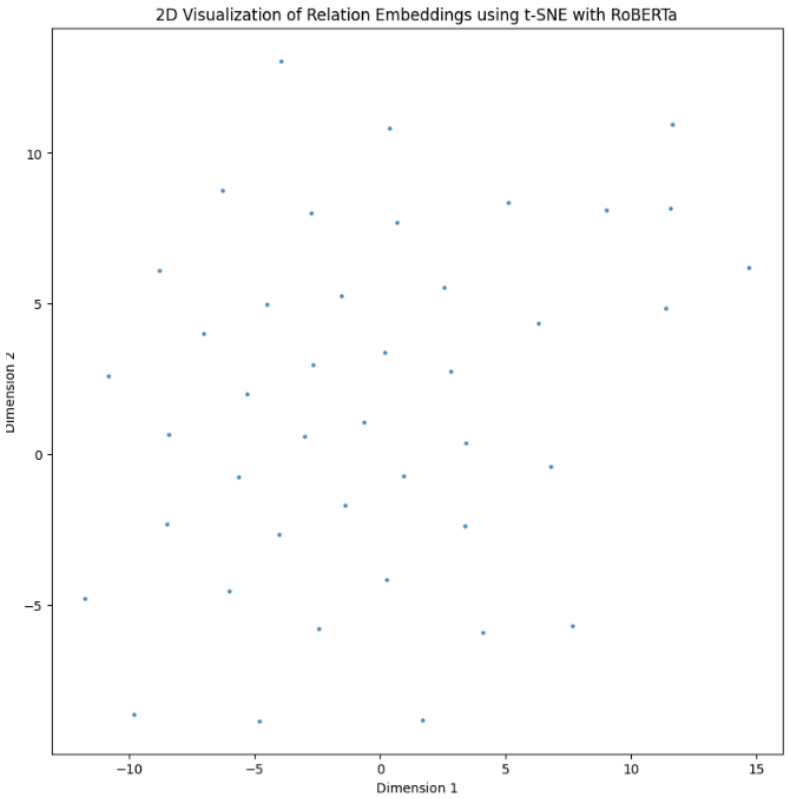
  - We used the description given for our task linked to these ids.
  - If description was not available corresponding to any id, then used the label as description.
  - These descriptions will be further passed through LLM to get embeddings.
  - For structuring the input graph, I tried all three approaches : adjacency list, adjacency matrix and edge list.
    - For adjacency list: {head:{relation:[list of all tails]}
    - For adjacency matrix: {relation: matrix with nodes as head (along row) and tail (along column), and setting cell value to 1 where this relation type exist}
    - For edge list: [[list of heads], [list of tails]] and an edge type: [list contains relation type corresponding to each head and tail combination].
    - Analysis: Since, it is a heterogeneous graph with multiple relationships, edge list way of structuring would be best because:
      - For an adjacency matrix, a large amount of memory is wasted because creating an adjacency matrix corresponding to each relation type creates a sparse matrix.
      - For adjacency lists, handling multiple relationships requires nested dictionaries, making it complex to manage and inefficient in memory for heterogeneous graphs.
      - Edge list method provides a scalable way to structure graphs as well as being directly compatible with PyTorch Geometric.

- **Entity Feature Initialization with LLM**
    - Used ‘roberta-base’ model to generate embeddings of each entity based on their description.
    - Token size was set to 128 since descriptions were all of smaller length, preventing useless padding which causes memory wastage.
    - For faster processing, I did it in batches of 128.
    - Embedding size was set to 768, which creates each entity description into a vector of size 768.
    - These embeddings (node embeddings and relation embeddings) are feature vectors for our GNN model.
    - Visualized the embeddings:
        - Node embeddings: First reduced the dimension to 50 using PCA then to 2 for t-SNE plot.



We can see that clusters are formed, although we can not say much about the clusters since nodes (entities) are not labeled. Still we can see that some clusters are formed which show similar topic descriptions are clustered together while different topics are separated properly. We can see more solid evaluation upon the relation embeddings tSNE plot.

- Relation Embeddings: First reduced the dimension to 30 using PCA then to 2 for t-SNE plot.



Here each cluster corresponds to a relation. As we can see that each relation is separated and we know that relations are related to the descriptions of their corresponding head and tail, it ensures that embeddings are fairly good.

- **Graph Neural Network Model Design**
  - Choosed Relational Graph Convolutional Network (R-GCN) as a suitable architecture, as it is designed to handle graphs with multiple types of edges (relations) between nodes.
  - Didn't choose GAT as it uses attention mechanisms to weigh the importance of neighboring nodes, it does not inherently handle multiple relation types.
  - Didn't choose GCN as it also doesn't directly handle multiple relations but aggregates neighboring node features using a single weight matrix for all relations, which limits its expressiveness in multi-relational graphs.
  - In an R-GCN, each relation type is represented by a separate weight matrix, allowing the model to learn different transformations for each relation. The model aggregates information from neighboring nodes by using relation-specific transformations, making it effective for handling the heterogeneous nature of relations in the dataset.
  - Each relation type in the dataset is encoded through distinct embeddings, ensuring that the network can process and learn from the diversity of relationships between entities.
  - In our RGCN model class, initialized it with:
    - Node embeddings, relation embeddings
    - Embedding dimensions, hidden dimensions
    - Convolutional layers (first layer: embedding dim to hidden dim, intermediate n-2 layers: hidden dim to hidden dim, final layer: hidden dim to embedding dim)

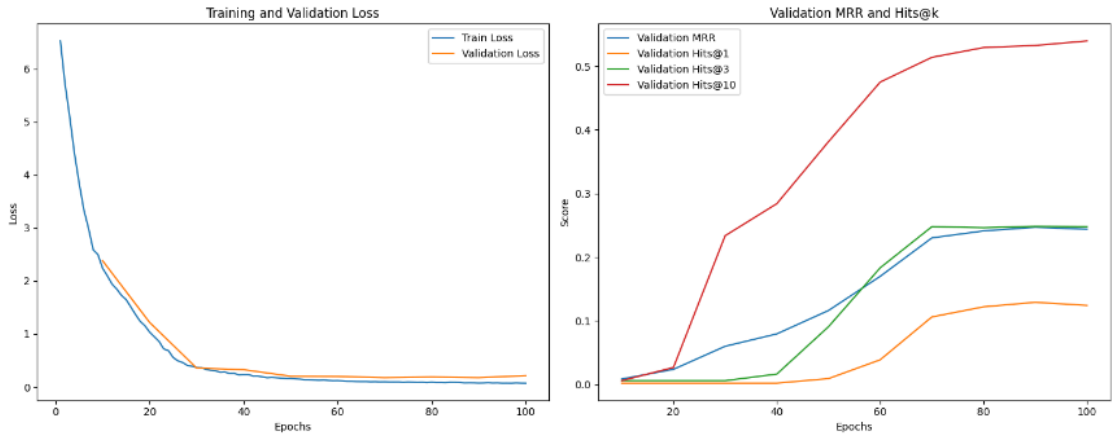
        - Dropout layer
    - The forward pass incorporates learning of both node and relation embeddings. The forward pass is as follows:
        - The model starts with pre-defined node embeddings representing the initial features for each node.
        - Relation embeddings are looked up for each edge based on its type, producing embeddings of shape.
        - Source and destination node embeddings are multiplied by their respective relation embeddings, and then summed to form edge embeddings.
        - The edge embeddings are aggregated into node embeddings using index_add to update the source and destination nodes based on the edges they belong to.
        - The updated node embeddings are passed through RGCN layers, applying graph convolutions, ReLU activation, and dropout for regularization.
        - The forward pass returns the updated node embeddings and the relation embeddings which are easily learnable through back propagation.

- **Knowledge Base Completion Task**
    - Implemented a score_triplet function which gives a combined score of such a combination. The score for the triplet is computed by taking the element-wise product of the head, relation, and tail embeddings which then aggregates to a single score. This will be used for link prediction tasks / knowledge graph completion.
    - Implemented the generate_negative_samples function which helps to create negative samples by randomly corrupting the original triplet (head, relation, tail). For each triplet, it randomly changes either the head or the tail entity with a 50% chance, ensuring that only one part of the triplet is modified at a time. If the head is corrupted, the tail stays the same, and vice versa. These negative samples, which represent invalid triplets, are used during training to help the model differentiate between valid and invalid triplets. By scoring the true and corrupted triplets, the model learns to recognize valid relationships and improve its accuracy.
    - Training Strategy:
        - Trained for 100 epochs, Adams optimizer used and Margin-based ranking loss is used as loss function.
        - The model processes positive triplets (head, relation, tail) and generates negative samples by randomly corrupting the head or tail of the triplet.
        - It computes scores for both the positive triplets and the negative samples using the score_triplet method. These scores represent the model's confidence in the validity of each triplet.
        - The margin-based ranking loss is calculated by comparing the positive and negative scores. The loss function encourages the model to assign higher scores to valid triplets than to corrupted ones.
        - Margin-based ranking loss:
          $L = 1/N * \sum max(0, \gamma - positive\ sample\ score + negative\ sample\ score)$
          Here γ is the margin which is set to 1 in our implementation.
        - Model is evaluated upon validation set after every 10 epochs.

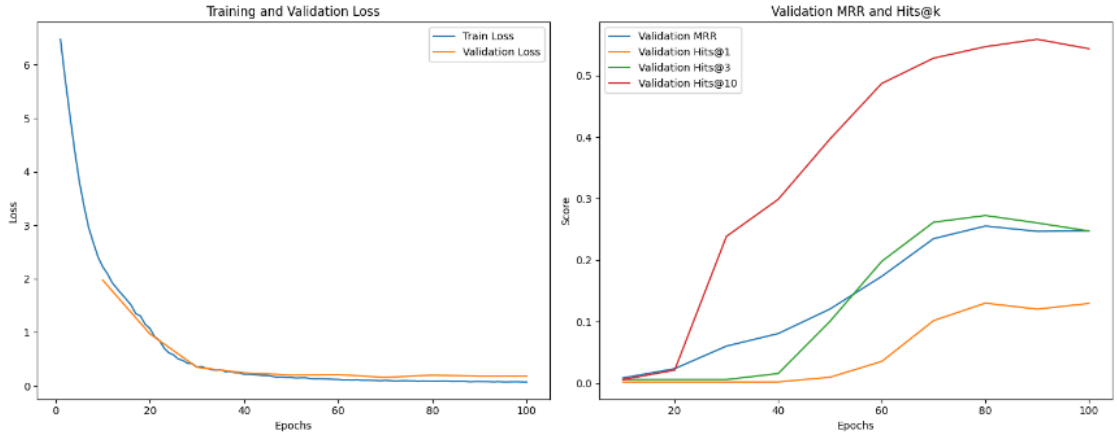
        - During validation, the model's performance is assessed using Mean Reciprocal Rank (MRR) and Hits@k metrics for various values of k (i.e. 1, 3, 10).
        - Training and validation losses, along with validation MRR and Hits@k scores, are plotted over the course of training to visualize the model's performance.
    - Evaluation Strategy:
        - The model is evaluated on each sample in the test dataset, where for each triple (head, relation, tail), the model computes the score for all possible tail entities.
        - For each test sample, the model predicts scores for all possible tail entities. These scores are then sorted in descending order. The rank of the true tail entity is determined based on its position in the sorted list.
        - The MRR is computed by calculating the reciprocal of the rank of the true tail for each test sample, and averaging this value across all test samples.
          $MRR = 1/N * \sum_{i=1 \text{ to } N} 1/\text{rank}_i$
          rank_i is the rank of the true tail for the i^{th} query.
        - For each test sample, the model checks if the true tail is among the top-k predicted tails. The Hits@k metric counts how many times the true tail is within the top k predictions, and the results are averaged over all test samples.
        - The function get_top_k_tails is used to predict the top-k tails for a given head and relation, showing the most likely entities as the tail for the input (head, relation).
- **Hyperparameter Tuning**
    - Performed 7 experiments, varying different parameters in each experiment to bring the insights on which parameter affects model performance.
    - Experiments details:

Experiment	Hyperparameter Changes
Exp. 1	Base case (hidden dim=768, num_layers=2, dropout=0.0, lr=0.01, initial embedding=llm-based)
Exp. 2	Decreased hidden dim (hidden dim=512, num_layers=2, dropout=0.0, lr=0.01, initial embedding=llm-based)
Exp. 3	Increased num_layers (hidden dim=768, num_layers=3, dropout=0.0, lr=0.01, initial embedding=llm-based)
Exp. 4	Increased dropout rate (hidden dim=768, num_layers=2, dropout=0.2, lr=0.01, initial embedding=llm-based)
Exp. 5	Decreased lr (hidden dim=768, num_layers=2, dropout=0.0, lr=0.005, initial embedding=llm-based)
Exp. 6	Increased lr (hidden dim=768, num_layers=2, dropout=0.0, lr=0.05, initial embedding=llm-based)
Exp. 7	Random embedding initialization (hidden dim=768, num_layers=2, dropout=0.0, lr=0.01, initial embedding= random)

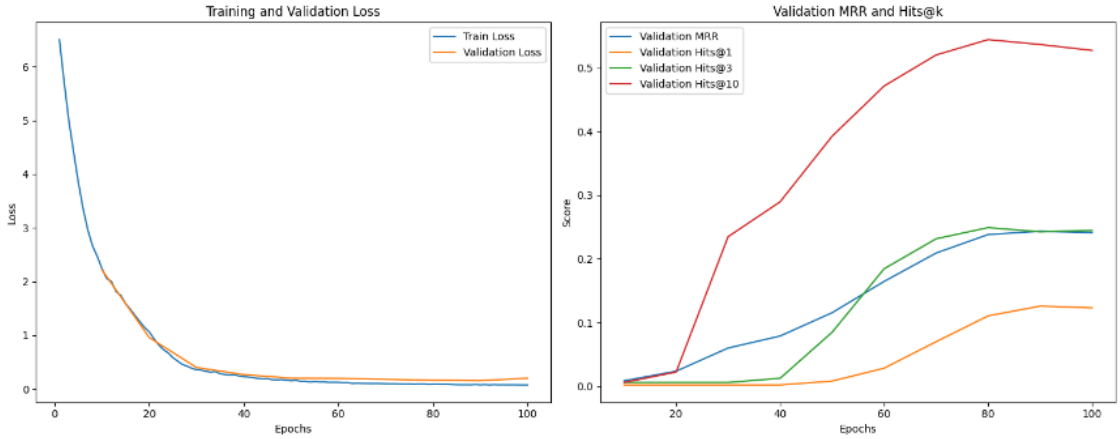
- Experiments Analysis:
  - Note: Since validation is done only after every 10 epochs, results will be shown for val loss and corresponding evaluation metric from 10th epoch onwards.
  - Experiment 1: Started with some hyperparameters which we set as baseline parameters. Training was smooth, loss is found decreasing and evaluation metrics are also increasing.
    Mean Reciprocal Rank (MRR): 0.2488
    Hits@10: 0.5339, Hits@3: 0.2445, Hits@1: 0.1329



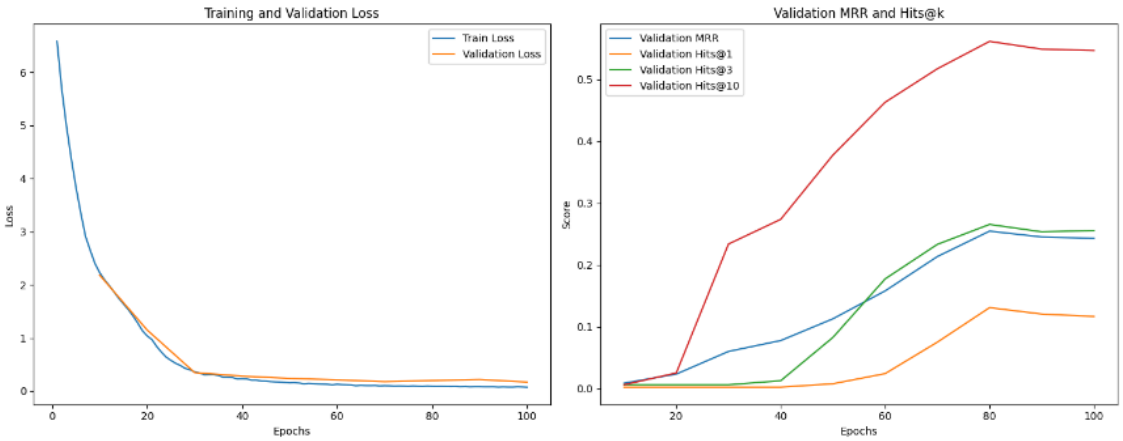
  - Experiment 2: Decreased hidden dim to 512, keeping rest hyperparameters same as base case. The training was smooth similar to base case, but evaluation metric found a decrease after 90th epochs on validation set.
    Mean Reciprocal Rank (MRR): 0.2427
    Hits@10: 0.5427, Hits@3: 0.2522, Hits@1: 0.1204
    MRR is lower than base case MRR and hits@1 but other hit rates are higher. This suggests that while the model is less effective at ranking the correct tail at the top, it is better at including the required tail in the list of top-k predictions compared to the base case.



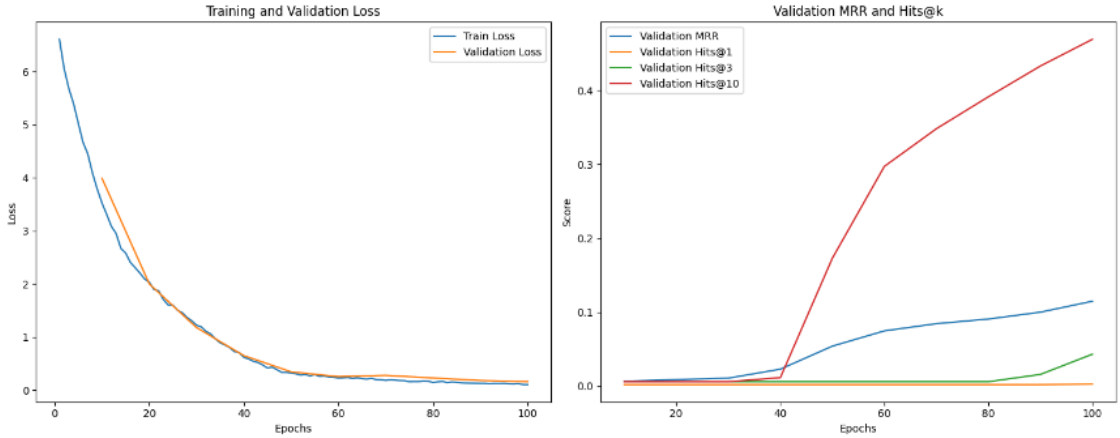
- Experiment 3: Increased the num of layers, keeping rest hyperparameters the same as base case. The training was smooth similar to the base case, but evaluation metric found a decrease after the 80th epochs on the validation set.
  Mean Reciprocal Rank (MRR): 0.2410
  Hits@10: 0.5202, Hits@3: 0.2440, Hits@1: 0.1209
  All evaluation metrics are lower than base case which suggests that overall model performance decreased when trained for the same no. of epochs. Possible cause might be since architecture size increased due to the addition of 1 more layer, it requires more epochs to train to achieve the same or better performance.



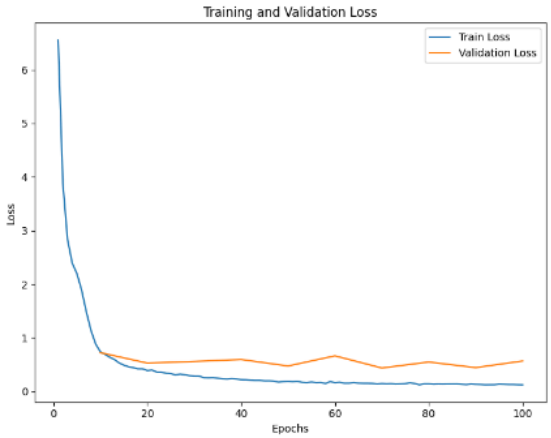
- Experiment 4: Increased dropout rate, keeping rest hyperparameters the same as base case. The training was smooth similar to base case, while evaluation metrics saw a decrease followed by a saturation after the 75th epoch.
  Mean Reciprocal Rank (MRR): 0.2405
  Hits@10: 0.5263, Hits@3: 0.2555, Hits@1: 0.1160
  Hits@3 is higher than the base case, while the other metrics are lower, suggesting that the model is less effective at ranking the correct tail at the top or including the required tail in the top-k predictions. However, it is better at ranking the tails it includes within the top 10 list.

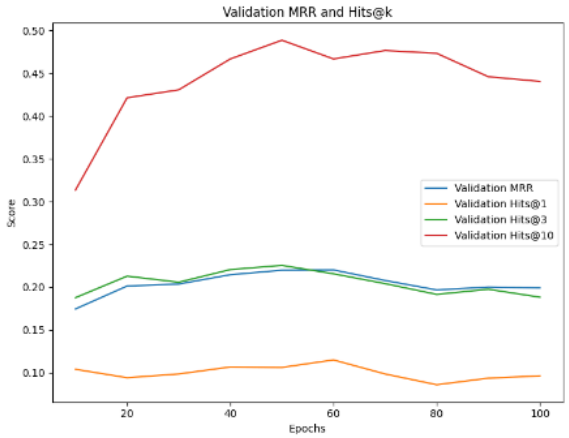


- Experiment 5: Decreased the learning rate, keeping rest hyperparameters the same as base case. The training was smooth similar to base case, but the evaluation metrics indicate room for improvement. This is because the model with the lower learning rate has not yet fully reached the performance level achieved by the model trained with a learning rate of 0.01. The reduced learning rate may require more time or epochs to effectively learn and reach the same performance.
  Mean Reciprocal Rank (MRR): 0.1129
  Hits@10: 0.4721, Hits@3: 0.0361, Hits@1: 0.0016
  All evaluation metrics are lower than base case, suggesting that it requires more epochs to train to achieve the same level of performance.

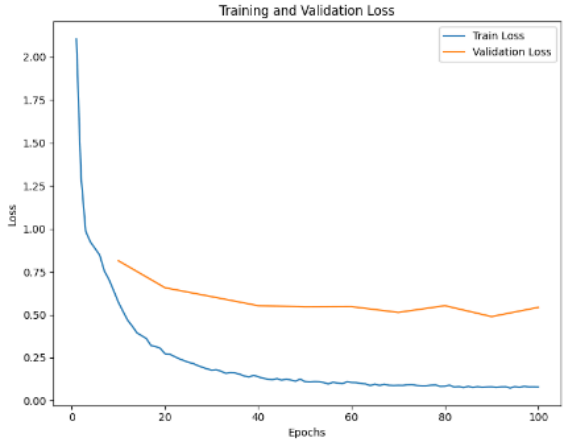


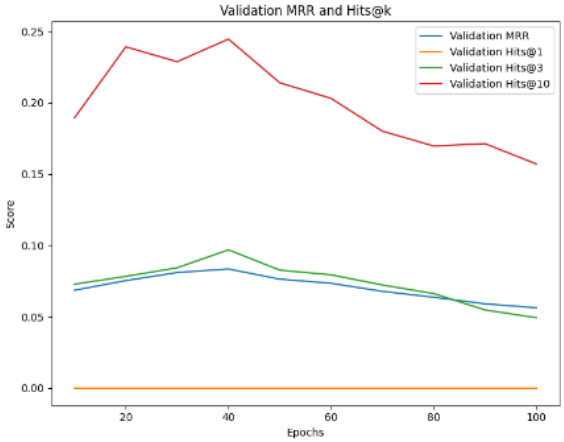
- Experiment 6: Increased the learning rate, keeping rest hyperparameters the same as base case. The training got saturated after 40 epochs. The evaluation metrics also show not much improvement after 50 epochs. This saturation suggests that a higher learning rate caused the model to converge too quickly or overshoot the optimal solution, preventing further improvement. The model did not benefit from continued training, indicating that a more gradual learning rate might have led to better results.
  Mean Reciprocal Rank (MRR): 0.1921
  Hits@10: 0.4223, Hits@3: 0.1800, Hits@1: 0.0925
  All evaluation metrics are lower than base case, suggesting that increasing the learning rate led to suboptimal model performance, leading to poorer generalization and lower evaluation scores compared to the base case.



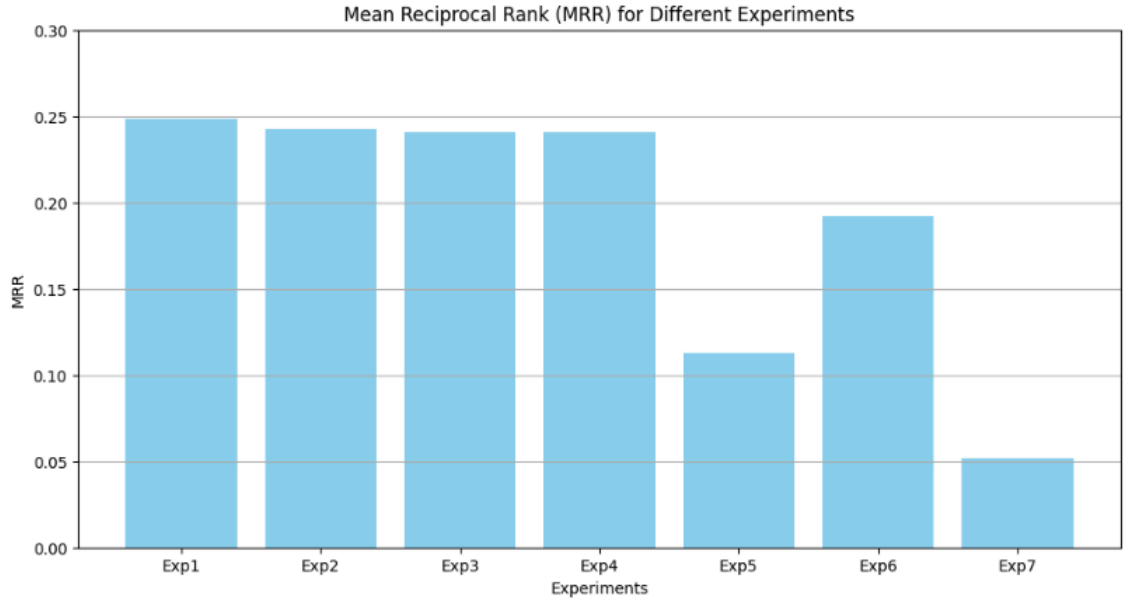


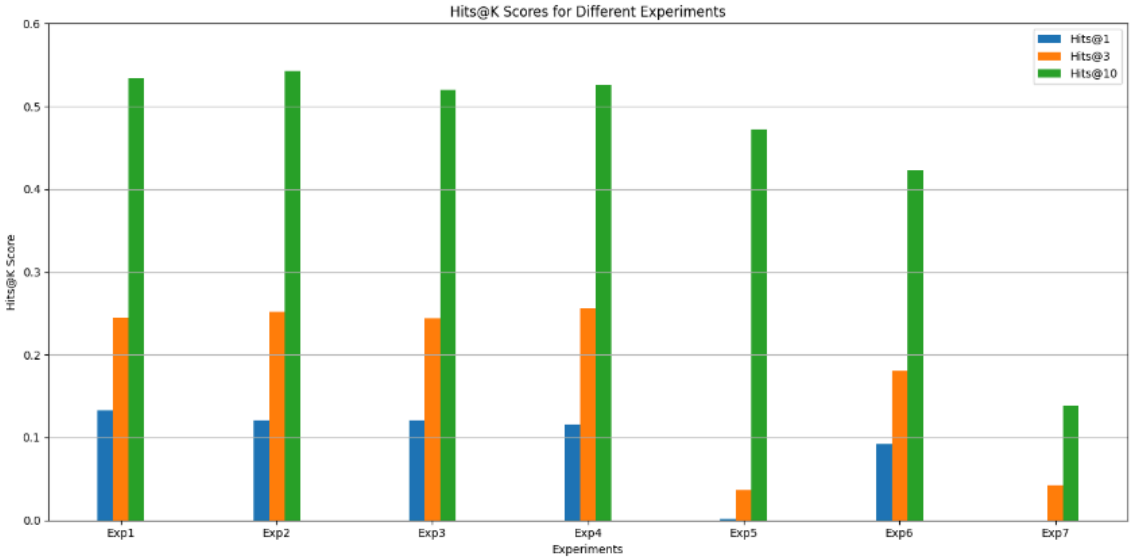
- Experiment 7: Used random initialization of embeddings, keeping rest hyperparameters the same as base case. The training was not learning properly, although the training loss seems to decrease initially, but validation loss doesn't show much significant improvement. This suggests that random initialization of embeddings may not have provided a good starting point for the model, causing it to struggle in learning meaningful patterns.
  Mean Reciprocal Rank (MRR): 0.0519,
  Hits@10: 0.1389, Hits@3: 0.0427, Hits@1: 0.0000
  All evaluation metrics are lower than base case.





- Best performing hyper-parameters:





In MRR and Hits@1, base case hyper-parameters are best, while at Hits@3 exp4 hyper-parameters are best and for Hits@10 exp2 is highest.
If we go by mean win rate, base case wins in two metrics, as well these metric scores are more important, hence optimal hyper-parameters found are of base case i.e. hidden dim=768, num_layers=2, dropout=0.0, lr=0.01, initial embedding=llm-based

- **Analysis of LLM Embedding Impact**
  - Analyzed 10 test sample results. Their descriptions were upon nationality, occupation, genre, instrument played, country, etc.
  - Of which LLM embedding model Hits@10 was 0.6 and for random embedding was 0.1 .

    - LLM embeddings perform better than random embeddings because they understand the meaning of words and their relationships. This helps them make more accurate predictions, especially for complex topics like jobs, nationalities, and music genres.
    - Random embeddings sometimes predict correctly by chance, but they don't have the same level of understanding as LLM embeddings.
    - Overall, LLM embeddings are more reliable and context-aware when figuring out connections between things.
- **LLM Embeddings Observation**
    - LLM identified occupations effectively, such as predicting "musician who plays the mandolin" for (Q319374, P106) and "person who writes plays" for (Q34970, P106), based on clear occupational roles.
    - For (Q122003, P1303), LLM correctly linked a Spanish singer-songwriter to their instrument, "fretted string instrument."
    - LLM correctly predicted the citizenship relationship for (Q206832, P27), linking a French mathematician to a country.
    - LLM had difficulty predicting the genre of a Swiss playwright's works in (Q115483, P136), showing challenges with creative fields.
    - LLM failed to capture the location of a Russian writer's group formation in (Q239652, P740), missing a clear geographic context.
    - LLM was unable to correctly predict the diplomatic relations for (Q833, P530), despite clear context regarding a country's diplomatic ties.
    - Overall, LLM embeddings excel in straightforward, factual connections but may underperform in more complex or context-dependent scenarios.
- **Challenges Encountered:**
    - **Graph Structure Selection:** We analyzed three graph structures i.e. adjacency matrix, adjacency list, and edge list, considering memory usage, scalability, and compatibility with PyTorch Geometric. A detailed analysis is provided in the Data Preprocessing section.
    - **Integrating Node and Relation Embeddings:** The challenge was to effectively incorporate both node and relation embeddings. Developed a unique approach to integrate relation information into node embeddings, detailed in the Graph Neural Network Model Design section (Forward Pass).
    - **Hyperparameter Tuning:** Finding the optimal hyperparameters was crucial. Conducted detailed experiments to test various configurations and formulated reasonable hypotheses, with results presented in the Hyperparameter Tuning section.
- **Future works:**
    - Improving LLM embeddings by training them on data from specific areas or topics. This would make the embeddings more relevant and improve performance for tasks related to those topics.
    - Addressing the high computational cost of LLM embeddings, future work could focus on optimizing model scalability, such as reducing embedding dimensions or implementing more efficient training algorithms, making the model more feasible for large-scale graph tasks.