Proposal: Graph Isomorphism Network (GIN) for Link Prediction in the FutureOfAIviaAI Repository

Ildar Rakiev

December 18, 2025

Objective

The goal of this project is to implement and evaluate a **Graph Isomorphism Network (GIN)** for the link prediction task on the SemanticGraph datasets from the `FutureOfAIviaAI` repository. The new method will be integrated into the existing codebase and compared against the current baseline model, which relies on hand-crafted features for each node pair.

Method to Implement

The method chosen for this project is a **Graph Isomorphism Network (GIN) for link prediction** on the SemanticGraph datasets. The core idea is to use a GIN encoder to learn node embeddings from the graph structure, and then apply a shallow neural decoder to predict the probability of an edge between two nodes. Concretely, a multi-layer GIN will aggregate information from neighbors to obtain node representations, and a small MLP will take a function of the two node embeddings (e.g., concatenation or element-wise product) to output a link probability.

Formally, a single GIN layer updates node representations as

$$h_v^{(k)} = \mathrm{MLP}^{(k)}\Big((1 + \epsilon^{(k)}) \cdot h_v^{(k-1)} + \sum_{u \in \mathcal{N}(v)} h_u^{(k-1)}\Big),$$

where $h_v^{(k)}$ is the embedding of node v at layer k, $\mathcal{N}(v)$ is the set of neighbors of v, and $\epsilon^{(k)}$ is a learnable or fixed scalar. After K layers, the final node embeddings $z_v = h_v^{(K)}$ are used for link prediction.

For a candidate edge (u, v), an edge representation will be constructed from the node embeddings, for example

$$e_{uv} = [z_u \,\|\, z_v] \quad \text{or} \quad e_{uv} = z_u \odot z_v,$$

and passed through an MLP with a sigmoid output to obtain the link probability $p(e_{uv} = 1)$.

Motivation: Why GIN

GIN is theoretically motivated as a powerful message-passing GNN whose discriminative power matches the Weisfeiler–Lehman graph isomorphism test, enabling it to capture subtle structural differences in graphs better than classic GCN-type architectures. This makes it a natural candidate for *link prediction in an evolving knowledge network*, where the existence of a future edge often depends on fine-grained structural patterns and higher-order neighborhoods rather than simple heuristics.

The current baseline model in the `FutureOfAIviaAI` repository (M6) uses 15 predefined, hand-crafted features for each pair of nodes and applies a simple learning model on top of these features. While this approach is efficient and interpretable, it is limited by the expressiveness of manually designed features and may miss complex structural patterns. In contrast, a GIN-based approach can learn task-specific structural features end-to-end from data, potentially improving AUC on challenging parts of the SemanticGraph.

From an implementation standpoint, GIN is also appealing due to its *practicality and clarity*. It can be implemented using existing PyTorch / PyTorch Geometric primitives (e.g., `GINConv`, `GIN` models), or via a custom message-passing layer, while keeping the overall pipeline consistent with the existing `evaluate_model.py` and `utils.py` structure in the repository. This allows integration into the current codebase with minimal changes to data loading and evaluation, while still adding a genuinely new and modern GNN method not yet present in the repository.

Expected Benefits

Stronger Structural Representation

By aggregating neighbor information with a WL-powerful update rule, GIN can better encode local and multi-hop structures that correlate with the appearance of new links, beyond what simple similarity scores or hand-crafted features can capture. This is particularly relevant for the SemanticGraph, which is an exponentially growing knowledge network where link formation depends on complex interactions between research topics and concepts.

End-to-End Learning of Link Features

Instead of manually designing pairwise features, the GIN encoder plus MLP decoder can learn appropriate representations directly from training data. This enables the model to adapt to the specific dynamics of the AI knowledge network and potentially discover non-trivial structural patterns that are difficult to capture with classical link prediction heuristics.

Comparable Evaluation and Extensibility

The method can be evaluated with the same AUC-ROC metric and the same SemanticGraph datasets used for the existing baseline (M6), making the results directly comparable. The plan is to implement a new script (e.g., `evaluate_gin.py`) that uses the existing utilities for dataset loading and AUC computation, and logs results in a format similar to the current baseline logs. This ensures that the new method integrates smoothly into the existing benchmarking framework and can later be extended or combined with other GNN variants.

Expected Challenges

Scalability and Training Time

The SemanticGraph datasets can be large, and running a GIN on multiple combinations of δ (time window) and cutoff (minimum degree) may be computationally more expensive than the current feature-based baseline. It may be necessary to carefully design batching strategies, consider subgraph or neighborhood sampling, or restrict full experiments to a representative subset of the 18 available datasets to keep training time manageable.

Negative Sampling and Class Balance

Link prediction requires constructing negative examples (non-edges) for training. Although the repository already provides balanced training sets (with equal numbers of positive and negative pairs) via its utility functions, it is important to ensure that this sampling strategy aligns well with the GIN training procedure. [page:1] Poorly chosen negative samples can lead to unstable optimization or biased evaluation, so this aspect must be handled carefully.

Hyperparameter Sensitivity

The performance of GNNs like GIN can depend strongly on the choice of depth, hidden dimension, activation functions, aggregation scheme, and regularization (e.g., dropout, weight decay). Too shallow a network may underfit, while too deep a network may suffer from oversmoothing on these particular knowledge graphs. Some hyperparameter tuning will likely be required to reach strong performance without excessive computational cost.

Planned Integration into the Repository

The current repository provides scripts for data creation (`create_data.py`), baseline evaluation (`evaluate_model.py`), the simple baseline model (`simple_model.py`), and various utilities including dataset preparation and AUC computation (`utils.py`). The planned integration steps are:

- Implement a new module (e.g., `gin_model.py`) containing the GIN encoder and the link prediction decoder.
- Implement a new evaluation script (e.g., `evaluate_gin.py`) that:
  - loads the SemanticGraph datasets using existing utility functions,
  - constructs the graph representation and node features,
  - trains the GIN-based link prediction model,
  - evaluates performance using AUC-ROC and logs results in the same style as the baseline.
- Optionally, provide configuration options (e.g., hidden dimension, number of layers, learning rate) to facilitate basic hyperparameter exploration.

This design keeps the existing baseline intact while adding a new, clearly separated GNN-based method that can be easily reproduced and extended.