Social Media Course Project

Application of Transformer-Like Deep Neural Network Architecture For Graph Edge Prediction

Bruno Guzzo
242504

Winter 2025

Abstract

This project explores the application of transformer-like Graph Neural Networks (GNNs) for edge prediction in semantic graph. The project aims to demonstrate the effectiveness of GNNs with architectural features inspired by transformer networks in capturing complex relationships and predicting potential connections within a graph dataset.

1 Introduction

The main given guideline assignment for the present project is *"Application of GNN on semantic graph generated by LLMs"*. To address such abstract assignment we chose *"home-made"* approach in which we build from scratch a dataset and a neural network to solve the assignment.
Our baseline idea is to build a dataset of graphs form Wikipedia article links where each node is a page title, then design a custom neural network to operate on such data type.
To design such custom neural network we get inspired by the latest development in natural language processing and graph neural network.
Thus, we adopt a deep neural network architecture that extend the encoder stack of the transformer Vaswani et al. [2023] with the application of graph attention network Veličković et al. [2018].

2 Graph Attention

Graph Attention Networks (GATs) are a class of neural network architectures designed to operate on graph-structured data. They leverage a self-attention mechanism Vaswani

et al. [2023] to dynamically learn the importance of neighboring nodes, enabling effective feature aggregation while being independent of the underlying graph structure and allowing inductive learning.

2.0.1 Graph Attention Layer

The fundamental building block of a GAT Veličković et al. [2018] is the *graph attentional layer*. Given a graph with N nodes, each represented by a feature vector $\mathbf{h}_i \in \mathbb{R}^F$ ($i \in \{1, \ldots, N\}$), the objective is to produce a new set of node features, $\mathbf{h}'_i \in \mathbb{R}^{F'}$.

2.0.2 Feature Transformation

Initially, each node's feature vector is linearly transformed:

$$\hat{\mathbf{h}}_i = \mathbf{W}\mathbf{h}_i, \tag{1}$$

where $\mathbf{W} \in \mathbb{R}^{F' \times F}$ is a learnable weight matrix.

2.0.3 Attention Mechanism

Next, an attention mechanism is employed to compute the importance of node j's features to node i. This is achieved through a shared attentional mechanism $a : \mathbb{R}^{F'} \times \mathbb{R}^{F'} \to \mathbb{R}$:

$$e_{ij} = a(\hat{\mathbf{h}}_i, \hat{\mathbf{h}}_j). \tag{2}$$

In practice, a is often implemented as a single-layer feed-forward neural network with parameters $\mathbf{a} \in \mathbb{R}^{2F'}$:

$$e_{ij} = \text{LeakyReLU}\left(\mathbf{a}^T[\hat{\mathbf{h}}_i || \hat{\mathbf{h}}_j]\right), \tag{3}$$

where $||$ denotes concatenation and LeakyReLU is a nonlinear activation function.

2.0.4 Masked Attention

To incorporate graph structure, the attention mechanism is masked. The coefficients e_{ij} are only computed for nodes $j \in \mathcal{N}_i$, where $\mathcal{N}_i$ is the set of neighbors of node i (including i itself).

2.0.5 Normalization

The coefficients are then normalized using a softmax function over the neighborhood of each node:

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k \in \mathcal{N}_i} \exp(e_{ik})}. \tag{4}$$

2.0.6 Feature Aggregation

Finally, the new feature vector for node i is calculated by aggregating the transformed features of its neighbors, weighted by the normalized attention coefficients:

$$\mathbf{h}'_i = \sigma\left(\sum_{j\in\mathcal{N}_i} \alpha_{ij}\hat{\mathbf{h}}_j\right), \tag{5}$$

where σ is a nonlinear activation function.

2.0.7 Multi-Head Attention

To stabilize the learning process and capture different aspects of node relationships, multi-head attention is utilized. The operation of the graph attention layer is performed K times in parallel, each with a separate transformation and attention parameters $\mathbf{W}^k$ and $\mathbf{a}^k$ (respectively). The output of each head is a new node feature vector, $\mathbf{h}_i^{'k}$.

2.0.8 Concatenation

The output from the different heads can be concatenated together:

$$\mathbf{h}'_i = \Big\|_{k=1}^{K} \sigma\left(\sum_{j\in\mathcal{N}_i} \alpha_{ij}^k\hat{\mathbf{h}}_j^k\right). \tag{6}$$

In this case, the final output feature vector will have the dimension KF'.

2.0.9 Averaging

Alternatively, for the final layers, the outputs can be averaged followed by the final activation:

$$\mathbf{h}'_i = \sigma\left(\frac{1}{K}\sum_{k=1}^{K}\left(\sum_{j\in\mathcal{N}_i} \alpha_{ij}^k\hat{\mathbf{h}}_j^k\right)\right). \tag{7}$$

Here, the final output feature vector has the dimension F'.

3 Wikipedia Articles Link Dataset

Given the necessity of a consistent graph dataset to train our GNN we chose to build a suited dataset to address the project needs. Each data point of our dataset is a graph where each node is a Wikipedia article title and the edges represent links between them. The presence of an edge (u, v) means that the article u cite article v in its body or vice-versa. Following this principle we obtain a non-directed graph where the edges indicate that two articles are somehow related.

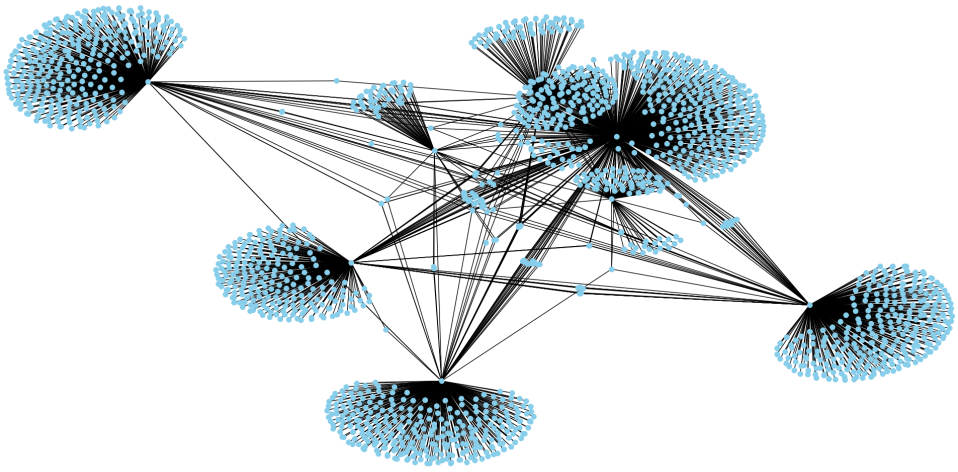

Figure 1: A graph of 2000 nodes built form the root article title 'Sustainability'.

3.1 Graph Extraction Algorithm

To obtain the graphs of our dataset we adopt a **breadth first strategy** to explore Wikipedia article links starting for a **root article**. This prioritize the inclusion of the immediate neighborhood of the root article. We adopt this methodology against a **depth first strategy** so we can have all the immediate neighbors of the root node, thus leading to a **robust representation** of the root article. This is particular helpful when using GNN where the network learn a node representation using the information present in it's first neighbors.

The final graph produced by our algorithm is limited by a maximum size of **20000 nodes**. This allow us to obtain a graph that capture the deep structure of Wikipedia.
Below we present a simplified **pseudo-code** of the BFS algorithm used, the final version is **recursive** and have several constraint to enforce graph consistency.

3.2 Node Embedding

To efficiently use our dataset with GNN it is needed to **embed each node** in way that it can be processed in a numerical way. To do so, we relay on state-of-the-art sentence embedding model to transform node labels in numeral representation. We chose to use a freely and open-source available **sentence-transformers** model: **all-MiniLM-L6-v2** Reimers and Gurevych [2019].

Algorithm 1 Wikipedia Graph Extraction using BFS

Require: $rootArticleTitle$, $maxNodes$

$graph \leftarrow$ new Graph
$visited \leftarrow$ new Set, initialized with $rootArticleTitle$
$queue \leftarrow$ new Queue, enqueue $rootArticleTitle$
$graph$.addNode($rootArticleTitle$)
while $queue$ is not empty and $|graph.nodes| < maxNodes$ **do**
    $pageTitle \leftarrow queue$.dequeue()
    $page \leftarrow$ getWikipediaPage($pageTitle$) ▷ Handles Page/Disambiguation Errors
    if $page$ is not valid **then**
        continue ▷ Skip to the next page in queue
    end if
    for $linkTitle$ in $page.links$ **do**
        if $|graph.nodes| \geq maxNodes$ **then**
            break ▷ Exit loop
        end if
        if $linkTitle \in visited$ **then**
            $graph$.addEdge($pageTitle$, $linkTitle$)
        else
            $visited$.add($linkTitle$)
            $queue$.enqueue($linkTitle$)
            $graph$.addNode($linkTitle$)
            $graph$.addEdge($pageTitle$, $linkTitle$)
        end if
    end for
end while
if not isConnected($graph$) **then**
    Error: Graph is not connected
end if
return $graph$

It maps sentences and paragraphs to a **384 dimensional dense vector space** and can be used for tasks like **clustering** or **semantic search**. **all-MiniLM-L6-v2** is derived from the MiniLM Wang et al. [2020] architecture and it leverages a distilled version of the **BERT** (Bidirectional Encoder Representations from Transformers) Devlin et al. [2019] model.
The description of this model is outside the scope of this project and it is already been exposed in our previous natural language processing project.,

3.3 Final Dataset

The final dataset has been generated by executing the **algorithm 1** on a list of Wikipedia article titles and saving the graph information (nodes and edges) in JSON files. We used a list of Wikipedia article titles related to **sustainability and development** (around 300), but we have also included the **top 50 most visited** article in 2024. This would ensure that the final GNN model will preserve a small but broad **generalization capability**.

However, to ensure fast data loading, we have also created a **tensor** file version of the dataset which include node embedding and labels as well. Thus, the final dataset is composed of **389 graphs** with **20000 nodes each**, this amount to roughly **7 million of nodes**. It worth it to note that, of course, some graphs could share common node labels.

4 Graph Neural Network Architecture

Before to get to the final network architecture it is important to remark decisions and ideas that lead us to develop such architecture.

- **Problem complexity**: The link prediction task is not easy, especially when dealing with thousands of nodes and edges. It is needed to capture relation between distant nodes. So, the propagation of the information form nodes to nodes is crucial to ensure optimal result. For this reason, we could hypnotize the need for a **deep neural network** to address the long information transfer needed.

- **Dealing with graphs**: Each data point of our dataset ia a huge graph, so it is needed to use state-of-the-art graph neural network methodology to deal with such objects. We chose to adopt the aforementioned **graph attention** Veličković et al. [2018] 2 since it has shown better results than convolutional graph neural network Kipf and Welling [2017].

- **Dealing with words**: Despite he graphic nature of the problem we are fundamentally trying to learn relation from words and small phrases. This is a common problem of **NLP** (Natural Language Processing). So, it is natural to look to the

solutions applied in such filed to take inspiration from. This intuition lead us to chose a sub-layer structure similar to the one used in the well-known **transformer architecture** Vaswani et al. [2023].

4.1 Network Architecture In Detail

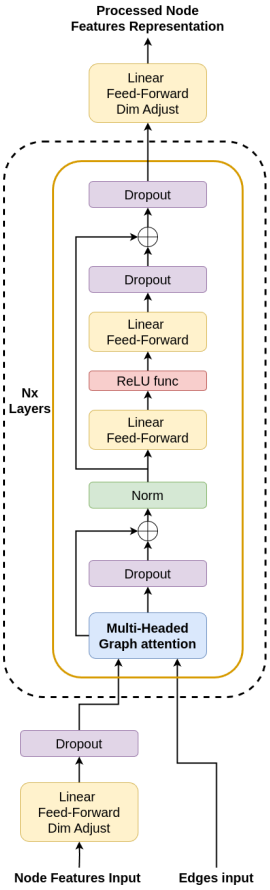

Figure 2: Graphical representation of the our final **graph neural network**.

4.2 Chosen Architecture Parameters

5 Network Training

6 Link Prediction Task Results

7 Methodology

Describe the methods you used to conduct your project. This might include data collection methods (e.g., surveys, interviews, social media data analysis), data analysis techniques, tools used, and any specific procedures you followed. Be clear and detailed enough so that someone else could replicate your work.

8 Results/Findings

Present your findings in a clear and organized manner. Use figures, tables, and other visual aids to illustrate your results effectively. Make sure your visuals are properly labeled and captioned.

Variable	Value 1	Value 2
Result 1	10	20
Result 2	30	40

Table 1: Description of your table.

9 Discussion

Interpret your results and discuss their implications. What do your findings mean? How do they relate to your research questions or objectives? Discuss any limitations of your study and suggest areas for future research.

10 Conclusion

Summarize your key findings and conclusions. Restate the significance of your project and its contributions to the field.

References

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019. URL https://arxiv.org/abs/1810.04805.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks, 2017. URL https://arxiv.org/abs/1609.02907.

Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 11 2019. URL https://arxiv.org/abs/1908.10084.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023. URL https://arxiv.org/abs/1706.03762.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks, 2018. URL https://arxiv.org/abs/1710.10903.

Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers, 2020. URL https://arxiv.org/abs/2002.10957.