APMTH 220 Geometric Methods for Machine Learning **Spring 2024**

Graph Representation Learning with Language Models

Name: Yilin Wang *Submission Date: 05/02/2024*

1 Abstract

When using graph neural networks (GNNs) to process text-attributed graphs (TAGs), we typically use fast, non-contextualized methods to extract coarse-grained text features, which might limit the performance of GNNs. In this project, we leverage the language model's (LM) ability to extract fine-grained, contextualized features to improve GNN's learning. We propose two approaches: (1) directly utilizing LLM's representation of texts as input features to GNNs, and (2) aligning the representation of GNN and LLM via a contrastive learning objective. We evaluated our models on the obgn-arvix dataset for node classification and found that (1) using LLMs as feature extractors (approach 1) leads to substantial performance gain, increasing the test accuracy by over 15 points; and (2) contrastive learning (approach 2) does not lead to performance gain. Our code can be found here.

2 Introduction

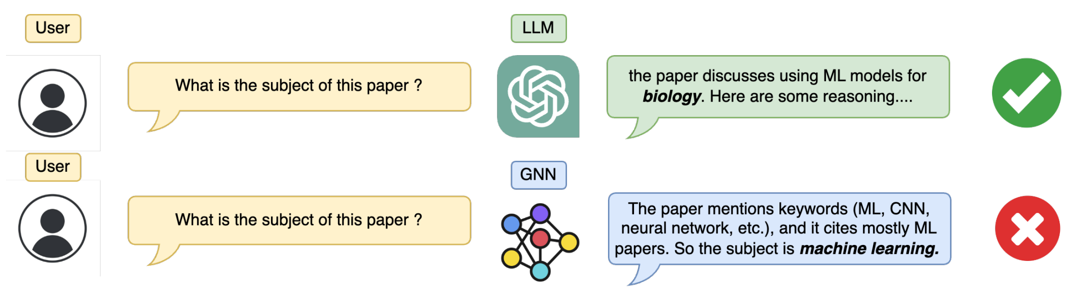


Figure 1: Possible Mistakes by GNN that would be correctly accounted for by the LLM.

Graphs with node features represented by text descriptions, known as Text-attributed graphs (TAGs), are commonly encountered in real-world scenarios, such as citation networks. When employing graph neural networks (GNNs) to learn from TAGs, the initial step involves transforming the raw text attributes into numerical features. This transformation is typically achieved using fast, non-contextualized methods. For example, in CORA and CiteSeer (Mccallum et al., 2000), the text features are represented by zero-one encoding of word occurrences. In obgn-arvix, the text features are computed as average word embeddings using a skip-gram language model like Word2Vec (Mikolov et al., 2013), which is coarse-grained and non-contextualized. Despite the effectiveness of GNNs in capturing structural and local graph information, their performance may be limited by the relatively coarse-grained nature of text features. Figure 1 provides one such example.

On the contrary, (Large) Language Models (LLMs) such as BERT (Devlin et al., 2019) and LLaMa-2 (Touvron et al., 2023) specialize in extracting fine-grained, contextualized text features. These models have demonstrated remarkable capabilities in reasoning and planning tasks. Thus, it would be beneficial to leverage the expertise of LLMs in text processing to enhance the learning of more refined representations from TAGs.

There are plenty of studies on using LLMs for graph-level tasks. Li et al. (2024) provides a comprehensive and up-to-date survey and proposes a taxonomy that classifies these works into three types: (1) LLM-as-enhancer, which uses LLM to enhance the input features for GNNs; (2) LLM-as-predictor, which directly uses LLM to perform graph-level tasks; and (3) GNN-LLM alignment, which seeks to align GNN and LLM embeddings.

When LLMs are used as enhancers, they could be used as feature extractors, where the LLM's representation of the text replaces the coarse-grained input features into the GNN (Chien et al., 2022; Duan et al., 2023a; Zhu et al., 2023; Liu et al., 2023a; Xie et al., 2023; Huang et al., 2023; Xue et al., 2023). Larger generative LLMs are sometimes used to provide more details on the raw text descriptions, which are still processed by the simpler feature extraction algorithms (He et al., 2024; Chen et al., 2024; Qian et al., 2023; Wei et al., 2024). When LLMs are directly used as the predictor, the graph structure is usually encoded in the form of text descriptions (Wang et al., 2024; Zhao et al., 2023; Guo et al., 2023; Shi et al., 2023). For example, a common prompt for in-context learning for node classification involves (1) stating the task, (2) providing the node's raw text feature, and (3) providing text descriptions of the neighbors (Liu et al., 2023b). The methodologies of GNN-LLM alignment are more diverse and are usually inspired by the ideas in multi-modal machine learning, particularly vision-language transformers. Common alignment methods include contrastive learning (Edwards et al., 2021; Su et al., 2022; Liu et al., 2024; Brannon et al., 2023), layer-wise alignment (Yang et al., 2023; Jin et al., 2023), and knowledge distillation (Mavromatis et al., 2023).

In this project, we propose two approaches to leverage the ability of LLM for tasks on TAGs : (1) directly utilizing LLM's representation of texts as input features to GNNs, and (2) aligning the representation of GNN and LLM via a contrastive learning objective. The first approach has been relatively well-explored and widely used, and we wish to provide more analysis of it by varying the configurations of the experiments. For the second approach, while there has been some work in contrastive learning to align LLM and GNN embeddings, they are predominantly focused on data relating to chemical molecule structure. This project seeks to explore the application of this idea on node classification on the obgn-arxiv dataset (Hu et al., 2021), which is a multi-subject citation network.

Through extensive experimentation, we find that (1) larger/wider GNNs perform marginally better than smaller GNNs; (2) using LLMs as feature extractors (approach 1) leads to substantial performance gain, increasing the test accuracy by over 15 points; and (3) contrastive learning (approach 2) does not lead to performance gain. We append a PDF of our code after this report and make it available on Github : https://github.com/TonyW42/am220

3 Background and Notation

A bulleted list of symbols used below and their explanations can be found in appendix 7.1

3.1 Preliminaries

Notation We are given a text-attributed graph TAG of the following form: $\mathcal{G} = (\mathcal{V}, E, \mathcal{T}, \mathcal{Y}, \mathcal{X})$, where $(\mathcal{V}, E)$ are respectively the set of vertices and edges, $\mathcal{T}$ denotes the set of text descriptions for each node in $\mathcal{V}$, $\mathcal{Y}$ denotes the class label for each node in $\mathcal{V}$, $\mathcal{X}$ refers to the set of coarse-grained text features. Let the dimension of the coarse-grained features be d_q, and let the number of classes in $\mathcal{Y}$ be m.

Problem Statement The goal of node classification is to find a function $f : \mathcal{G} \setminus \{\mathcal{Y}\} \to \mathcal{Y}$, which predicts the node class given other information in the graph. Given a model that outputs a probability distribution $\hat{p} \in \mathbb{R}^m$ and an example with label y_i, we optimize the model over the cross entropy loss

$$\mathcal{L}_{CE} = \frac{1}{N} \sum_{i=1}^{N} \hat{p}(y_i | \mathcal{G} \setminus \{\mathcal{Y}\})$$

3.2 Graph Neural Networks (GNN)

GNNs are often used for node classification. It learns node embedding via the message-passing paradigm:

$$h_v^{(0)} = x(v), \quad m(v)^{(l+1)} = f_{Agg}^{(l)} \left(h^{(l)}(v), \{h^{(l)}(u), w(u,v) | u \in \mathcal{N}(v)\} \right), \quad h(v)^{(l+1)} = f_{up}(h^{(l)}(v), m^{(l+1)}(v))$$

where $\mathcal{N}(v)$ refers to the neighbor of v, $w(u,v)$ refers to the edge weights between (u,v), $x(v)$ refers to the input feature of v. Typically, the last layer embedding is used as the node representation. Here, we slightly modify the notation. We use $h(v)^{(N-1)} \in \mathbb{R}^{d_h}$ to denote the node representation learned by the GNN. We use $h(v)^{(N)} \in \mathbb{R}^m$ to denote the GNN's prediction of output probability distribution across m classes, obtained by passing $h(v)^{(N-1)}$ through a linear + softmax layer. i.e., a N-layer GNN could consists of $N-1$ message-passing GNN layer and one classification "head". Also, let $h(v)^{(l)} \in \mathbb{R}^{d_h}$ for all $h \neq 0, N$ (we enforce the same embedding size for all hidden message passing GNN layers).

In this study, we utilize two types of GNN: graph convolutional networks GCN (Kipf and Welling, 2017) and graph attention networks GAT (Veličković et al., 2018). We include very brief descriptions on the formulations of these two networks in appendix 7.2 and leave further details to their original papers.

3.3 Language Models

Transformer (Vaswani et al., 2023) language models (LM) excel at extracting fine-gained information. In this study, we will make use of encoder-only language models $LM : \mathbb{T}^n \to \mathbb{R}^{n \times d_t}$, which takes a sequence of tokens and generate a representation of size d_t for each token. Typically, the first token is the `[CLS]` token, and we use its embedding to represent the whole text. We denote this process $LM^{CLS} : \mathbb{T}^n \to \mathbb{R}^{d_t}$

4 Proposed Approach

4.1 Approach 1: LM as feature extractor

In the first approach, we directly use an LM to extract contextual features for GNNs. Figure 2 presents the architecture of approach 1. Given a node and its features (v, t, x), where $v \in \mathcal{V}, t \in \mathcal{T}, q \in \mathcal{X}$, we first convert raw text t to a sequence of token t' using a tokenizer and then use a LM to obtain the sentence embedding $h^{LM}(v) = LM^{CLS}(t') \in \mathbb{R}^{d_t}$. We then concatenate $h^{LM}(v)$ and x to form the enhanced input features $a(v) = \text{Concat}(q, h^{LM}(v)) \in \mathbb{R}^{d_q + d_t}$, which is then used by the GNN for node classification.

While optimizing the GNN for node classification, we are not updating the language model to avoid the challenges of storing the entire computation graph, which requires excessive memory resources. Instead, we use the Sentence Transformer (Reimers and Gurevych, 2019), specifically designed for text retrieval, which offers robust, general-purpose text representations without the need to learn task-specific LM representations.

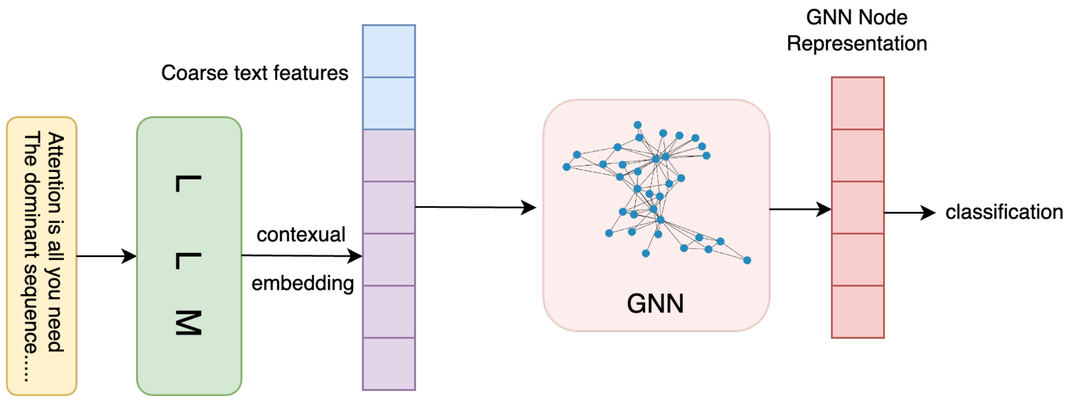


Figure 2: Using LLM's contextual representation for GNN feature enhancement

4.2 Approach 2: GNN and LLM Representation Alignment

Figure 3 presents the architecture of approach 2. Here, we first feed $\mathcal{G} \setminus \{Y\}$ to the GNN to extract node representation $h^{N-1}_{GNN}(v) \in \mathbb{R}^{d_h}$ for every node v. Similar to approach 1, we use the LLM to extract contextualized representation $h^{LM}(v) \in \mathbb{R}^{d_t}$ for texts associated with each node v. We then use a linear transformation to map the GNN and LLM's embedding to the same size d:

$$
\begin{aligned}
t(v) &= W^L h^{LM}(v) & W^L &\in \mathbb{R}^{d \times d_t} \\
g(v) &= W^G h^{N-1}_{GNN}(v) & W^G &\in \mathbb{R}^{d \times d_h}
\end{aligned}
$$

Given a batch of nodes $B = \{v_1, v_2, \cdots, v_{bs}\}$ with batch size bs, we define the matrix $t_B, g_B \in \mathbb{R}^{bs \times d}$, where $t_B[i] = t(v_i)$, $g_B[i] = g(v_i)$. Following CLIP (Radford et al., 2021), we compute the contrastive loss $\mathcal{L}_{contrastive}$ as follows:

$$
\begin{aligned}
\mathcal{L}(t_B, g_B) &= -\log \mathrm{Softmax}\left(t_B \cdot g_B^T \times \exp(\tau)\right) \\
\mathcal{L}_{contrastive} &= \frac{1}{bs}(\mathcal{L}(t_B, g_B) + \mathcal{L}(g_B, t_B))
\end{aligned}
$$

where $\tau = 0.07$ is the temperature hyperparameter. We also optimize both the LM and the GNN on cross-entropy loss for node classification, so the overall loss is

$$
\mathcal{L} = \mathcal{L}_{contrastive} + \frac{1}{bs}\sum_{i=1}^{bs}\left(\mathcal{L}_{CE}(h^N_{GNN}(v_i), y_i) + \mathcal{L}_{CE}(\mathrm{Softmax}(W^{LM}_{head} h^{LM}(v)), y_i)\right)
$$

where y_i is the label on node v_i, $W^{LM}_{head} \in \mathbb{R}^{m \times d_t}$ maps the LM's sentence representation to the class label distribution. $\mathcal{L}_{CE}$ refers to the cross entropy loss defined before.

5 Results

5.1 Data

we use the obgn-arxiv dataset (Hu et al., 2021) for all experiments. Obgn-arxiv is a citation network that contains 169,343 nodes for node classification across 40 categories ($m = 40$). Each node is a paper

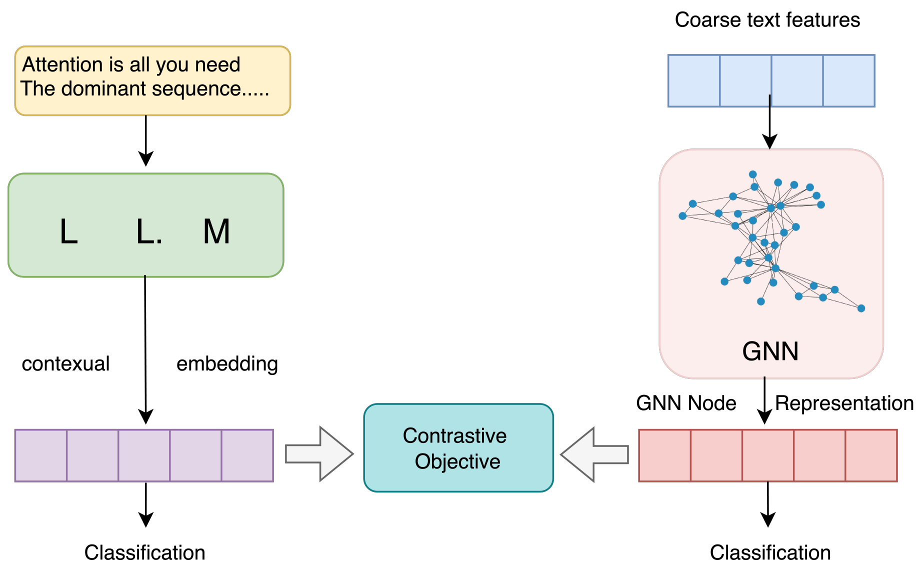


Figure 3: Aligning LLM and GNN embedding space via contrastive learning

and categories refer to different academic subjects. The raw feature is a 128-dimensional vector obtained by averaging the embeddings of words using a skip-gram model. The dataset also provides the title and the abstract of each paper. The dataset provides the split for training, validation, and test data. For all experiments, we train on the training split and report the classification accuracy on the test split.

5.2 Experimental Setup

We run three sets of experiments. First, we run baseline experiments using only GNNs and the input features provided in the dataset. We experimented with a 2 or 3 layer graph convolutions network (GCN) with hidden size $d_h = 16, 32, 64, 128, 256$. We also experimented a 2-layer graph attention network (GAT) with 8 heads and hidden size $d_h = 16, 32, 64$ *for each head.* we tried to fit a 3-layer GAT or with $d_h = 128, 256$, but unfortunately the GPU does not have enough memory to run these specifications. For the latter two set of experiments, we we use a 2-layer GCN with $d_h = 64$ and 2-layer GAT with $d_h = 32$

For the second set of experiments, we experimented with using the Sentence Transformer to extract different kinds of text features: the title, the abstract, and them combined. For the first and second set of experiments, we use an Adam optimizer with learning rate 0.001. For the second set of experiment only, we also use a weight decay of 5e-4 as regularization. We train all models for 5000 epochs, as we observe that the loss have converged after 5000 epochs for all models. The sentence transformer's text embedding has dimension 768 (i.3., $d_t = 768$).

For the third set of experiments, we use distill-bert-uncased (Sanh et al., 2020) as the LM backbone. We first trained the GNN to convergence and then doing the contrastive finetuning (along with continued training on node classification). For each type of GNN, we perform 4 different experiments based on two dimensions: (1) whether to train the LM to convergence (on node classification) before contrastive finetuning, and (2) whether to use approach 1 (Sentence transformer's representation of abstract and title combined) on the GNN. For both the LM and the GNN, we use the Adam optimizer. For the GNN, we used a learning rate of 0.001 and weight decay of 5e-4; for the LM, we used a learning rate of 2e-5.

The batch size is 16 for contrastive finetuning.

5.3 Baseline Results

Model / hidden size	16	32	64	128	256
2-layer GCN	51.37	52.53	53.31	53.53	53.93
3-layer GCN	50.37	52.37	52.26	53.50	**54.35**
2-layer GAT	53.98	54.14	54.26	-	-

Table 1: Test set accuracy for baseline models under different configurations. Increasing the hidden size of the GNN marginally increase accuracy. Numbers in row 1 denote the hidden size d_h of the GNN

Table 1 shows the results of the baseline experiments. Generally, we see that the model performs better with a "wider" GNN (i.e., with a larger hidden size). However, the gain in performance is rather marginal. Further, although the best result is achieved by a 3-layer GCN with $d_h = 256$, in general increasing the number of layers (i.e., depth) of the GNN does not lead to performance gain, but leads to a drop in accuracy in most cases. In general, GATs perform slightly better than GCNs, though GATs likely consume more FLOPs as they take longer to train, and #FLOPs are usually positively correlated with performance. Since including both the title and abstract does not lead to better performance than only including the abstract, the abstract likely contains all the information in the title.

5.4 Approach 1 Results

Model / feature type	None	Title	Abstract	Title + Abstract
GCN	53.31	62.04	69.34	69.30
GAT	54.13	63.38	**71.05**	70.94

Table 2: Test set accuracy for using LLM as feature extractor. Including the LLM's representation of the abstract leads to substantial performance gains ($>$ 15 points in accuracy), whereas information in the title leads to a smaller improvement.

Table 2 shows the results of approach 1, where we use LLM to extract rich text features for GNN. We see that including the Sentence transformer's representation of the abstract leads to a substantial performance boost, increasing the test set accuracy by over 15 points for both GNN models. Including the Sentence transformer's representation for the title leads to a smaller (8 point) improvement. This is expected, as the abstract is more detailed and contains more information than the title.

5.5 Approach 2 Results

Table 3 presents the results from using a contrastive learning objective to align LLM and GNN embeddings (Approach 2). Comparisons between Config 1 vs Config 2 and Config 3 vs Config 4 show that pre-training the LM on node classification doesn't significantly affect performance. Experiments employing Approach 1, which utilizes Sentence Transformer features, demonstrate notably higher performance than those that do not. Additionally, comparing Table 2 with Table 3 indicates that contrastive fine-tuning has a minimal impact on the results.

	Tuned LM?	Approach 1?	GCN Accuracy	GAT Accuracy
Config 1	✓	×	53.95	54.86
Config 2	×	×	53.58	54.97
Config 3	✓	✓	69.82	**70.63**
Config 4	×	✓	70.04	70.45

Table 3: Test set accuracy for approach 2 on contrastive learning. "Config X" refers to the different training configurations explained by the second table. ✓ to "Tuned LM?" means that we first train the LM on node classification before contrastive finetuning. ✓ to "Approach 1?" means that we are enhancing the GNN with Sentence Transformer features described in approach 1.

6 Discussion and Conclusion

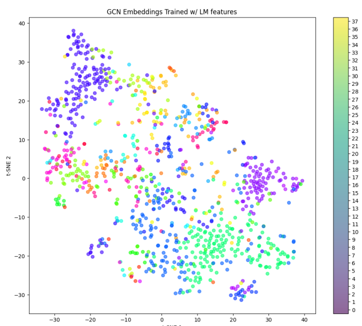


Figure 4: T-SNE visualization of GCN embedding with LM features

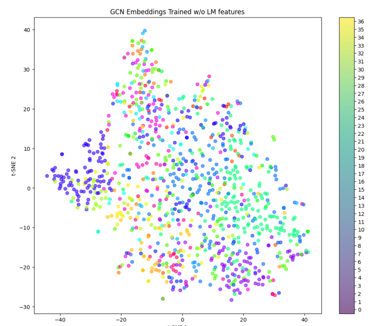


Figure 5: T-SNE visualization of GCN embedding without LM features

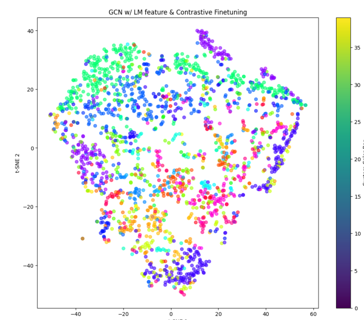


Figure 6: T-SNE visualization of GCN embedding with LM features & contrastive tuning

6.1 Discussion on Results

Visualization Figure 4, 5, 6 shows the t-sne visualization of the GCN's embedding H_{GCN}^{N-1} for three different configuration: (1) trained on node classification with LM's features (approach 1), (2) trained on node classification without LM's features (baseline), and (3) fine-tuned on contrastive loss and node classification with LM's features (Config 3 in approach 2). We see that approach 1 seems to separate the GNN's embeddings of nodes belonging to different classes more clearly. Approach 2 seems to transform the shape of the embedding space more drastically than approach 1, but each classes are less separated. This could be plausible, as the purpose of contrastive learning is to align representation spaces, while approach 1 serves as an enhancement to original representations.

Contrastive Learning In table 3, we observe that contrastive learning does not lead to performance gain. The most likely reason is that the batch size is too small. With a batch size of 16, the contrastive task might have been too easy for both models, so contrastive learning does not force them to learn more fine-grained information from each other. As a reference, the original CLIP model (Radford et al., 2021) uses a batch size of 32,768 during pretraining. Also, contrastive fine-tuning for 2 epochs might have been insufficient. To address this issue, we could (1) increase batch size, (2) train for more epochs, and (3) more carefully tune the hyper-parameters. We leave these to future work due to computation constraints, as (1) would require us to parallelize the computation across multiple GPUs, and (2) & (3) require substantially more run time.

Other Observations There are several interesting observations not formally discussed in the results section. First, we could directly use the LM for classification. We train the Distill-BERT model for 3 epochs, and it achieves an accuracy of 72.40 on the test set. As the LM outperforms the best GNN that uses (frozen) sentence transformer features, it might suggest that training an approach-1 GNN while also updating its underlying feature-extracting LM might yield higher performance gain. However, doing so is very costly as running both the forward pass (to extract feature) and the backward path (to update model) on the LM are very expensive, and we empirically observe that GNN takes a lot of epochs to converge. Thus, we leave optimization on this algorithm to future research. Also, we observe that the approach-1 GNN is more prone to overfitting as evidenced by the validation loss, and this is the motivation for the weight decay regularization described in section 5.2. This is expected, as using the LM feature increases the input embedding size, causing the GNN to more easily overfit.

6.2 Future Research

Aside from the future directions mentioned above, which are mostly motivated by the results, here are a few other directions for future research that are more ideological and theoretical.

LLM Following Duan et al. (2023b), we could try to incorporate the graph's structural information in the LM. For example, we could append a sentence saying "Here is a list of papers that this work cites", followed by a list of the neighboring paper's titles, to each node's text. Also, we could use generative LMs for feature enhancement for the GNN or to perform in-context learning for direct classification.

Efficiency As mentioned before, any operation involving the LM is very costly. Any training involving the Distill-BERT model takes roughly an hour per epoch. There are plenty of methods for improving LM efficiency like pruning and quantization (Han et al., 2016). While it is worthwhile to implement these methods in our experimental setup, it would be interesting to investigate whether we could devise better efficiency techniques specific to our scenario (i.e., using LM for GNN).

Design Choices We have evaluated our model under various configurations, there are several other experiments we could run. For approach 1, we might consider using only the LM's text representation as input instead of combining it with coarse-grained features to see how it affects performance. In approach 2, since the LM significantly outperforms the GNN, initially freezing the LM during the early stages of contrastive fine-tuning could be beneficial. This would encourage the GNN to imitate the LM without the LM being adversely affected by the GNN's initial lower performance.

6.3 Conclusion

In this paper, we propose two methods that leverage the language model's (LM) ability to extract fine-grained features from text for processing text-attributed graphs (TAGs) using GNNs. The first method involves directly using LM representations as input features for GNNs, which significantly increases test accuracy by over 15 points. The second method employs a contrastive learning objective to align GNN and LM representations; however, this approach did not yield a performance gain. Our analysis, which includes visualization of GNN embeddings, supports these findings. Looking ahead, future research could focus on expanding the scale of contrastive finetuning (e.g., larger batch sizes and extended training time), enhancing the integration of LMs into GNN (such as end-to-end training, incorporating graph structural information into LMs, and improving LM deployment efficiency), and experimenting under various other configurations of the models.

References

William Brannon, Suyash Fulay, Hang Jiang, Wonjune Kang, Brandon Roy, Jad Kabbara, and Deb Roy. 2023. Congrat: Self-supervised contrastive pretraining for joint graph and text embeddings.

Zhikai Chen, Haitao Mao, Hang Li, Wei Jin, Hongzhi Wen, Xiaochi Wei, Shuaiqiang Wang, Dawei Yin, Wenqi Fan, Hui Liu, and Jiliang Tang. 2024. Exploring the potential of large language models (llms) in learning on graphs.

Eli Chien, Wei-Cheng Chang, Cho-Jui Hsieh, Hsiang-Fu Yu, Jiong Zhang, Olgica Milenkovic, and Inderjit S Dhillon. 2022. Node feature extraction by self-supervised multi-scale neighborhood prediction.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding.

Keyu Duan, Qian Liu, Tat-Seng Chua, Shuicheng Yan, Wei Tsang Ooi, Qizhe Xie, and Junxian He. 2023a. Simteg: A frustratingly simple approach improves textual graph learning.

Keyu Duan, Qian Liu, Tat-Seng Chua, Shuicheng Yan, Wei Tsang Ooi, Qizhe Xie, and Junxian He. 2023b. Simteg: A frustratingly simple approach improves textual graph learning.

Carl Edwards, ChengXiang Zhai, and Heng Ji. 2021. Text2Mol: Cross-modal molecule retrieval with natural language queries. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 595–607, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Jiayan Guo, Lun Du, Hengyu Liu, Mengyu Zhou, Xinyi He, and Shi Han. 2023. Gpt4graph: Can large language models understand graph structured data ? an empirical evaluation and benchmarking.

Song Han, Huizi Mao, and William J. Dally. 2016. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding.

Xiaoxin He, Xavier Bresson, Thomas Laurent, Adam Perold, Yann LeCun, and Bryan Hooi. 2024. Harnessing explanations: Llm-to-lm interpreter for enhanced text-attributed graph representation learning.

Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. 2021. Open graph benchmark: Datasets for machine learning on graphs.

Xuanwen Huang, Kaiqiao Han, Dezheng Bao, Quanjin Tao, Zhisheng Zhang, Yang Yang, and Qi Zhu. 2023. Prompt-based node feature extractor for few-shot learning on text-attributed graphs.

Bowen Jin, Wentao Zhang, Yu Zhang, Yu Meng, Xinyang Zhang, Qi Zhu, and Jiawei Han. 2023. Patton: Language model pretraining on text-rich networks.

Thomas N. Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks.

Yuhan Li, Zhixun Li, Peisong Wang, Jia Li, Xiangguo Sun, Hong Cheng, and Jeffrey Xu Yu. 2024. A survey of graph meets large language model: Progress and future directions.

Hao Liu, Jiarui Feng, Lecheng Kong, Ningyue Liang, Dacheng Tao, Yixin Chen, and Muhan Zhang. 2023a. One for all: Towards training one graph model for all classification tasks.

Hao Liu, Jiarui Feng, Lecheng Kong, Ningyue Liang, Dacheng Tao, Yixin Chen, and Muhan Zhang. 2023b. One for all: Towards training one graph model for all classification tasks.

Shengchao Liu, Weili Nie, Chengpeng Wang, Jiarui Lu, Zhuoran Qiao, Ling Liu, Jian Tang, Chaowei Xiao, and Anima Anandkumar. 2024. Multi-modal molecule structure-text model for text-based retrieval and editing.

Costas Mavromatis, Vassilis N. Ioannidis, Shen Wang, Da Zheng, Soji Adeshina, Jun Ma, Han Zhao, Christos Faloutsos, and George Karypis. 2023. Train your own gnn teacher: Graph-aware distillation on textual graphs.

Andrew Mccallum, Kamal Nigam, and Jason Rennie. 2000. Automating the construction of internet portals.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space.

Chen Qian, Huayi Tang, Zhirui Yang, Hong Liang, and Yong Liu. 2023. Can large language models empower molecular property prediction?

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. 2021. Learning transferable visual models from natural language supervision.

Nils Reimers and Iryna Gurevych. 2019. 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.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2020. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter.

Yaorui Shi, An Zhang, Enzhi Zhang, Zhiyuan Liu, and Xiang Wang. 2023. Relm: Leveraging language models for enhanced chemical reaction prediction.

Bing Su, Dazhao Du, Zhao Yang, Yujie Zhou, Jiangmeng Li, Anyi Rao, Hao Sun, Zhiwu Lu, and Ji-Rong Wen. 2022. A molecular multimodal foundation model associating molecule graphs with natural language.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023. Llama 2: Open foundation and fine-tuned chat models.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2023. Attention is all you need.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph attention networks.

Heng Wang, Shangbin Feng, Tianxing He, Zhaoxuan Tan, Xiaochuang Han, and Yulia Tsvetkov. 2024. Can language models solve graph problems in natural language?

Wei Wei, Xubin Ren, Jiabin Tang, Qinyong Wang, Lixin Su, Suqi Cheng, Junfeng Wang, Dawei Yin, and Chao Huang. 2024. Llmrec: Large language models with graph augmentation for recommendation.

Han Xie, Da Zheng, Jun Ma, Houyu Zhang, Vassilis N. Ioannidis, Xiang Song, Qing Ping, Sheng Wang, Carl Yang, Yi Xu, Belinda Zeng, and Trishul Chilimbi. 2023. Graph-aware language model pre-training on a large graph corpus can help multiple graph applications.

Rui Xue, Xipeng Shen, Ruozhou Yu, and Xiaorui Liu. 2023. Efficient large language models fine-tuning on graphs.

Junhan Yang, Zheng Liu, Shitao Xiao, Chaozhuo Li, Defu Lian, Sanjay Agrawal, Amit Singh, Guangzhong Sun, and Xing Xie. 2023. Graphformers: Gnn-nested transformers for representation learning on textual graph.

Jianan Zhao, Le Zhuo, Yikang Shen, Meng Qu, Kai Liu, Michael Bronstein, Zhaocheng Zhu, and Jian Tang. 2023. Graphtext: Graph reasoning in text space.

Jing Zhu, Xiang Song, Vassilis N. Ioannidis, Danai Koutra, and Christos Faloutsos. 2023. Touchup-g: Improving feature representation through graph-centric finetuning.

7 Appendix

7.1 Symbols Dictionary

- $\mathcal{G}$: The text-attributed graph
- $\mathcal{V}$: the set of nodes
- E: the set of edges
- $\mathcal{T}$: the set of texts corresponding to each node
- $\mathcal{Y}$: the set of labels corresponding to each node
- $\mathcal{X}$: the set of coarse-grained input features corresponding to each node.
- m: the number of classes in $\mathcal{Y}$
- $\mathbb{T}$: the set of tokens (vocabulary) in the language model.
- LM^{CLS}: a language model that extracts the sentence-level representation, which is the LM's representation for the `[CLS]` token.
- d_q: the dimension of coarse-gained input features. i.e., $\forall x \in \mathcal{X}, x \in \mathbb{R}^{d_q}$
- d_t: the embedding size of the language model's representation.
- d_h: the hidden size of the GNN.
- $h^{LM}(v)$: The language model's representation of node v (obtained by encoding its raw text features). $h^{LM}(v) \in \mathbb{R}^{d_t}$
- $h^{N-1}_{GNN}(v)$: The GNN's representation of node v. $h^{N-1}_{GNN}(v) \in \mathbb{R}^{d_h}$
- $h^{N}_{GNN}(v)$: The GNN's output probability among m classes. $h^{N}_{GNN}(v) \in \mathbb{R}^{m}$

7.2 Formulation of GCN and GAT

The graph convolution network (GCN) defines the aggregation and update function as follows (modified from class slides):

$$m^{(l+1)}(v) = \sum_{u \in \mathcal{N}(v) \cup \{b\}} \frac{h^{(l)}(u)}{\sqrt{|\mathcal{N}(u)| \cdot |\mathcal{N}(v)|}}$$

$$h^{(l+1)}(v) = \sigma\left(W^{(l)} \sum_{u \in \mathcal{N}(v) \cup \{b\}} \frac{h^{(l)}(u)}{\sqrt{|\mathcal{N}(u)| \cdot |\mathcal{N}(v)|}} \right)$$

The graph attention network (GAT) defines the aggregation and update function as follows (modified from the original paper (Veličković et al., 2018)):

$$\alpha_{i,j} = \frac{\exp\left(\text{LeakyReLU}\left(a^T [W h^{(l)}(i) || W h^{(l)}(j)]\right)\right)}{\sum_{k \in \mathcal{N}(i)} \exp\left(\text{LeakyReLU}\left(a^T [W h^{(l)}(i) || W h^{(l)}(k)]\right)\right)}$$

$$m^{(l+1)}(v) = \sum_{u \in \mathcal{N}(v)} \alpha_{u,v} h^{(l)}(v)$$

$$h^{(l+1)}(v) = \sigma(W^{(l)} m^{(l+1)}(v))$$

where a, W are trainable parameters and $[\cdot||\cdot]$ denote the concatenation function.

6ai0pusce

May 3, 2024

Graph Representation Learning with Language Models

AM 220 course project

Yilin wang

1 Preliminaries

Please run all cells in this part to install all packages and download all data

[1]:
```
import os
import torch
os.environ['TORCH'] = torch.__version__
print(torch.__version__)

!pip install -q torch-scatter -f https://data.pyg.org/whl/torch-${TORCH}.html
!pip install -q torch-sparse -f https://data.pyg.org/whl/torch-${TORCH}.html
!pip install -q git+https://github.com/pyg-team/pytorch_geometric.git
!pip install ogb
!pip install transformers
!pip install sentence_transformers
```

```
2.2.1+cu121
                           10.9/10.9 MB
80.4 MB/s eta 0:00:00
                           5.0/5.0 MB
38.7 MB/s eta 0:00:00
  Installing build dependencies … done
  Getting requirements to build wheel … done
  Preparing metadata (pyproject.toml) … done
  Building wheel for torch-geometric (pyproject.toml) … done
Collecting ogb
  Downloading ogb-1.3.6-py3-none-any.whl (78 kB)
                           78.8/78.8 kB
3.1 MB/s eta 0:00:00
Requirement already satisfied: torch>=1.6.0 in
/usr/local/lib/python3.10/dist-packages (from ogb) (2.2.1+cu121)
Requirement already satisfied: numpy>=1.16.0 in /usr/local/lib/python3.10/dist-
packages (from ogb) (1.25.2)
```

```
Requirement already satisfied: tqdm>=4.29.0 in /usr/local/lib/python3.10/dist-
packages (from ogb) (4.66.2)
Requirement already satisfied: scikit-learn>=0.20.0 in
/usr/local/lib/python3.10/dist-packages (from ogb) (1.2.2)
Requirement already satisfied: pandas>=0.24.0 in /usr/local/lib/python3.10/dist-
packages (from ogb) (2.0.3)
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.10/dist-
packages (from ogb) (1.16.0)
Requirement already satisfied: urllib3>=1.24.0 in
/usr/local/lib/python3.10/dist-packages (from ogb) (2.0.7)
Collecting outdated>=0.2.0 (from ogb)
  Downloading outdated-0.2.2-py2.py3-none-any.whl (7.5 kB)
Requirement already satisfied: setuptools>=44 in /usr/local/lib/python3.10/dist-
packages (from outdated>=0.2.0->ogb) (67.7.2)
Collecting littleutils (from outdated>=0.2.0->ogb)
  Downloading littleutils-0.2.2.tar.gz (6.6 kB)
  Preparing metadata (setup.py) … done
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-
packages (from outdated>=0.2.0->ogb) (2.31.0)
Requirement already satisfied: python-dateutil>=2.8.2 in
/usr/local/lib/python3.10/dist-packages (from pandas>=0.24.0->ogb) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-
packages (from pandas>=0.24.0->ogb) (2023.4)
Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/dist-
packages (from pandas>=0.24.0->ogb) (2024.1)
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.10/dist-
packages (from scikit-learn>=0.20.0->ogb) (1.11.4)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-
packages (from scikit-learn>=0.20.0->ogb) (1.4.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.20.0->ogb) (3.5.0)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-
packages (from torch>=1.6.0->ogb) (3.14.0)
Requirement already satisfied: typing-extensions>=4.8.0 in
/usr/local/lib/python3.10/dist-packages (from torch>=1.6.0->ogb) (4.11.0)
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist-packages
(from torch>=1.6.0->ogb) (1.12)
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-
packages (from torch>=1.6.0->ogb) (3.3)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages
(from torch>=1.6.0->ogb) (3.1.3)
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages
(from torch>=1.6.0->ogb) (2023.6.0)
Collecting nvidia-cuda-nvrtc-cu12==12.1.105 (from torch>=1.6.0->ogb)
  Using cached nvidia_cuda_nvrtc_cu12-12.1.105-py3-none-manylinux1_x86_64.whl
(23.7 MB)
Collecting nvidia-cuda-runtime-cu12==12.1.105 (from torch>=1.6.0->ogb)
  Using cached nvidia_cuda_runtime_cu12-12.1.105-py3-none-manylinux1_x86_64.whl
```

```
(823 kB)
Collecting nvidia-cuda-cupti-cu12==12.1.105 (from torch>=1.6.0->ogb)
  Using cached nvidia_cuda_cupti_cu12-12.1.105-py3-none-manylinux1_x86_64.whl
(14.1 MB)
Collecting nvidia-cudnn-cu12==8.9.2.26 (from torch>=1.6.0->ogb)
  Using cached nvidia_cudnn_cu12-8.9.2.26-py3-none-manylinux1_x86_64.whl (731.7
MB)
Collecting nvidia-cublas-cu12==12.1.3.1 (from torch>=1.6.0->ogb)
  Using cached nvidia_cublas_cu12-12.1.3.1-py3-none-manylinux1_x86_64.whl (410.6
MB)
Collecting nvidia-cufft-cu12==11.0.2.54 (from torch>=1.6.0->ogb)
  Using cached nvidia_cufft_cu12-11.0.2.54-py3-none-manylinux1_x86_64.whl (121.6
MB)
Collecting nvidia-curand-cu12==10.3.2.106 (from torch>=1.6.0->ogb)
  Using cached nvidia_curand_cu12-10.3.2.106-py3-none-manylinux1_x86_64.whl
(56.5 MB)
Collecting nvidia-cusolver-cu12==11.4.5.107 (from torch>=1.6.0->ogb)
  Using cached nvidia_cusolver_cu12-11.4.5.107-py3-none-manylinux1_x86_64.whl
(124.2 MB)
Collecting nvidia-cusparse-cu12==12.1.0.106 (from torch>=1.6.0->ogb)
  Using cached nvidia_cusparse_cu12-12.1.0.106-py3-none-manylinux1_x86_64.whl
(196.0 MB)
Collecting nvidia-nccl-cu12==2.19.3 (from torch>=1.6.0->ogb)
  Using cached nvidia_nccl_cu12-2.19.3-py3-none-manylinux1_x86_64.whl (166.0 MB)
Collecting nvidia-nvtx-cu12==12.1.105 (from torch>=1.6.0->ogb)
  Using cached nvidia_nvtx_cu12-12.1.105-py3-none-manylinux1_x86_64.whl (99 kB)
Requirement already satisfied: triton==2.2.0 in /usr/local/lib/python3.10/dist-
packages (from torch>=1.6.0->ogb) (2.2.0)
Collecting nvidia-nvjitlink-cu12 (from nvidia-cusolver-
cu12==11.4.5.107->torch>=1.6.0->ogb)
  Using cached nvidia_nvjitlink_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl
(21.1 MB)
Requirement already satisfied: MarkupSafe>=2.0 in
/usr/local/lib/python3.10/dist-packages (from jinja2->torch>=1.6.0->ogb) (2.1.5)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.10/dist-packages (from requests->outdated>=0.2.0->ogb)
(3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-
packages (from requests->outdated>=0.2.0->ogb) (3.7)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.10/dist-packages (from requests->outdated>=0.2.0->ogb)
(2024.2.2)
Requirement already satisfied: mpmath>=0.19 in /usr/local/lib/python3.10/dist-
packages (from sympy->torch>=1.6.0->ogb) (1.3.0)
Building wheels for collected packages: littleutils
  Building wheel for littleutils (setup.py) … done
  Created wheel for littleutils: filename=littleutils-0.2.2-py3-none-any.whl
size=7029
```

```
sha256=6779afc762d1e05a026cc54390e0e4466aa5f7a0ab2031da214dd5b44dcc7d08
  Stored in directory: /root/.cache/pip/wheels/3d/fe/b0/27a9892da57472e538c7452a
721a9cf463cc03cf7379889266
Successfully built littleutils
Installing collected packages: littleutils, nvidia-nvtx-cu12, nvidia-nvjitlink-
cu12, nvidia-nccl-cu12, nvidia-curand-cu12, nvidia-cufft-cu12, nvidia-cuda-
runtime-cu12, nvidia-cuda-nvrtc-cu12, nvidia-cuda-cupti-cu12, nvidia-cublas-
cu12, outdated, nvidia-cusparse-cu12, nvidia-cudnn-cu12, nvidia-cusolver-cu12,
ogb
Successfully installed littleutils-0.2.2 nvidia-cublas-cu12-12.1.3.1 nvidia-
cuda-cupti-cu12-12.1.105 nvidia-cuda-nvrtc-cu12-12.1.105 nvidia-cuda-runtime-
cu12-12.1.105 nvidia-cudnn-cu12-8.9.2.26 nvidia-cufft-cu12-11.0.2.54 nvidia-
curand-cu12-10.3.2.106 nvidia-cusolver-cu12-11.4.5.107 nvidia-cusparse-
cu12-12.1.0.106 nvidia-nccl-cu12-2.19.3 nvidia-nvjitlink-cu12-12.4.127 nvidia-
nvtx-cu12-12.1.105 ogb-1.3.6 outdated-0.2.2
Requirement already satisfied: transformers in /usr/local/lib/python3.10/dist-
packages (4.40.1)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-
packages (from transformers) (3.14.0)
Requirement already satisfied: huggingface-hub<1.0,>=0.19.3 in
/usr/local/lib/python3.10/dist-packages (from transformers) (0.20.3)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/dist-
packages (from transformers) (1.25.2)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.10/dist-packages (from transformers) (24.0)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.10/dist-
packages (from transformers) (6.0.1)
Requirement already satisfied: regex!=2019.12.17 in
/usr/local/lib/python3.10/dist-packages (from transformers) (2023.12.25)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-
packages (from transformers) (2.31.0)
Requirement already satisfied: tokenizers<0.20,>=0.19 in
/usr/local/lib/python3.10/dist-packages (from transformers) (0.19.1)
Requirement already satisfied: safetensors>=0.4.1 in
/usr/local/lib/python3.10/dist-packages (from transformers) (0.4.3)
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.10/dist-
packages (from transformers) (4.66.2)
Requirement already satisfied: fsspec>=2023.5.0 in
/usr/local/lib/python3.10/dist-packages (from huggingface-
hub<1.0,>=0.19.3->transformers) (2023.6.0)
Requirement already satisfied: typing-extensions>=3.7.4.3 in
/usr/local/lib/python3.10/dist-packages (from huggingface-
hub<1.0,>=0.19.3->transformers) (4.11.0)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.10/dist-packages (from requests->transformers) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-
packages (from requests->transformers) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in
```

```
/usr/local/lib/python3.10/dist-packages (from requests->transformers) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.10/dist-packages (from requests->transformers) (2024.2.2)
Collecting sentence_transformers
  Downloading sentence_transformers-2.7.0-py3-none-any.whl (171 kB)
                           171.5/171.5

kB 5.1 MB/s eta 0:00:00
Requirement already satisfied: transformers<5.0.0,>=4.34.0 in
/usr/local/lib/python3.10/dist-packages (from sentence_transformers) (4.40.1)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages
(from sentence_transformers) (4.66.2)
Requirement already satisfied: torch>=1.11.0 in /usr/local/lib/python3.10/dist-
packages (from sentence_transformers) (2.2.1+cu121)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages
(from sentence_transformers) (1.25.2)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-
packages (from sentence_transformers) (1.2.2)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages
(from sentence_transformers) (1.11.4)
Requirement already satisfied: huggingface-hub>=0.15.1 in
/usr/local/lib/python3.10/dist-packages (from sentence_transformers) (0.20.3)
Requirement already satisfied: Pillow in /usr/local/lib/python3.10/dist-packages
(from sentence_transformers) (9.4.0)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-
packages (from huggingface-hub>=0.15.1->sentence_transformers) (3.14.0)
Requirement already satisfied: fsspec>=2023.5.0 in
/usr/local/lib/python3.10/dist-packages (from huggingface-
hub>=0.15.1->sentence_transformers) (2023.6.0)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-
packages (from huggingface-hub>=0.15.1->sentence_transformers) (2.31.0)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.10/dist-
packages (from huggingface-hub>=0.15.1->sentence_transformers) (6.0.1)
Requirement already satisfied: typing-extensions>=3.7.4.3 in
/usr/local/lib/python3.10/dist-packages (from huggingface-
hub>=0.15.1->sentence_transformers) (4.11.0)
Requirement already satisfied: packaging>=20.9 in
/usr/local/lib/python3.10/dist-packages (from huggingface-
hub>=0.15.1->sentence_transformers) (24.0)
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist-packages
(from torch>=1.11.0->sentence_transformers) (1.12)
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-
packages (from torch>=1.11.0->sentence_transformers) (3.3)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages
(from torch>=1.11.0->sentence_transformers) (3.1.3)
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.1.105 in
/usr/local/lib/python3.10/dist-packages (from
torch>=1.11.0->sentence_transformers) (12.1.105)
```

```
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.1.105 in
/usr/local/lib/python3.10/dist-packages (from
torch>=1.11.0->sentence_transformers) (12.1.105)
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.1.105 in
/usr/local/lib/python3.10/dist-packages (from
torch>=1.11.0->sentence_transformers) (12.1.105)
Requirement already satisfied: nvidia-cudnn-cu12==8.9.2.26 in
/usr/local/lib/python3.10/dist-packages (from
torch>=1.11.0->sentence_transformers) (8.9.2.26)
Requirement already satisfied: nvidia-cublas-cu12==12.1.3.1 in
/usr/local/lib/python3.10/dist-packages (from
torch>=1.11.0->sentence_transformers) (12.1.3.1)
Requirement already satisfied: nvidia-cufft-cu12==11.0.2.54 in
/usr/local/lib/python3.10/dist-packages (from
torch>=1.11.0->sentence_transformers) (11.0.2.54)
Requirement already satisfied: nvidia-curand-cu12==10.3.2.106 in
/usr/local/lib/python3.10/dist-packages (from
torch>=1.11.0->sentence_transformers) (10.3.2.106)
Requirement already satisfied: nvidia-cusolver-cu12==11.4.5.107 in
/usr/local/lib/python3.10/dist-packages (from
torch>=1.11.0->sentence_transformers) (11.4.5.107)
Requirement already satisfied: nvidia-cusparse-cu12==12.1.0.106 in
/usr/local/lib/python3.10/dist-packages (from
torch>=1.11.0->sentence_transformers) (12.1.0.106)
Requirement already satisfied: nvidia-nccl-cu12==2.19.3 in
/usr/local/lib/python3.10/dist-packages (from
torch>=1.11.0->sentence_transformers) (2.19.3)
Requirement already satisfied: nvidia-nvtx-cu12==12.1.105 in
/usr/local/lib/python3.10/dist-packages (from
torch>=1.11.0->sentence_transformers) (12.1.105)
Requirement already satisfied: triton==2.2.0 in /usr/local/lib/python3.10/dist-
packages (from torch>=1.11.0->sentence_transformers) (2.2.0)
Requirement already satisfied: nvidia-nvjitlink-cu12 in
/usr/local/lib/python3.10/dist-packages (from nvidia-cusolver-
cu12==11.4.5.107->torch>=1.11.0->sentence_transformers) (12.4.127)
Requirement already satisfied: regex!=2019.12.17 in
/usr/local/lib/python3.10/dist-packages (from
transformers<5.0.0,>=4.34.0->sentence_transformers) (2023.12.25)
Requirement already satisfied: tokenizers<0.20,>=0.19 in
/usr/local/lib/python3.10/dist-packages (from
transformers<5.0.0,>=4.34.0->sentence_transformers) (0.19.1)
Requirement already satisfied: safetensors>=0.4.1 in
/usr/local/lib/python3.10/dist-packages (from
transformers<5.0.0,>=4.34.0->sentence_transformers) (0.4.3)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-
packages (from scikit-learn->sentence_transformers) (1.4.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-
```

```
learn->sentence_transformers) (3.5.0)
Requirement already satisfied: MarkupSafe>=2.0 in
/usr/local/lib/python3.10/dist-packages (from
jinja2->torch>=1.11.0->sentence_transformers) (2.1.5)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.10/dist-packages (from requests->huggingface-
hub>=0.15.1->sentence_transformers) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-
packages (from requests->huggingface-hub>=0.15.1->sentence_transformers) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.10/dist-packages (from requests->huggingface-
hub>=0.15.1->sentence_transformers) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.10/dist-packages (from requests->huggingface-
hub>=0.15.1->sentence_transformers) (2024.2.2)
Requirement already satisfied: mpmath>=0.19 in /usr/local/lib/python3.10/dist-
packages (from sympy->torch>=1.11.0->sentence_transformers) (1.3.0)
Installing collected packages: sentence_transformers
Successfully installed sentence_transformers-2.7.0
```

[2]:
```
import torch
from torch_geometric.nn import GCNConv
import torch.nn.functional as F
from torch_geometric.data import DataLoader
from ogb.nodeproppred import PygNodePropPredDataset, Evaluator
from transformers import AutoTokenizer, AutoModel
from torch.utils.data import Dataset, DataLoader
import pandas as pd
from tqdm import tqdm
from sentence_transformers import SentenceTransformer, util
from torch_geometric.nn import GATConv
from itertools import chain
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
```

[3]:
```
!gdown 1qoCKJyFgMx08h9SgQoeIkYAPnV5sJfHW
!gdown 1Y6hEqmjr3fXQuW1AvQ_l3ps5Rl2vnZHB
!gdown 1E0G_0_SgjZLb7ikRLtaQKTYC4mx8sBV1
!gdown 1Rm-tajqdiNsfta-yErgMGH3xIKvrBFv4
!gdown 1KDaf1DX3AwwYOuBWhjx6ll0BergtwIv4
```

```
Downloading…
From (original):
https://drive.google.com/uc?id=1qoCKJyFgMx08h9SgQoeIkYAPnV5sJfHW
From (redirected): https://drive.google.com/uc?id=1qoCKJyFgMx08h9SgQoeIkYAPnV5sJ
fHW&confirm=t&uuid=b988d53a-035f-4cb6-b864-d103d0779bba
```

```
To: /content/all.pt
100% 825M/825M [00:11<00:00, 73.3MB/s]
Downloading…
From (original):
https://drive.google.com/uc?id=1Y6hEqmjr3fXQuW1AvQ_l3ps5Rl2vnZHB
From (redirected): https://drive.google.com/uc?id=1Y6hEqmjr3fXQuW1AvQ_l3ps5Rl2vn
ZHB&confirm=t&uuid=ced86766-f9e9-47df-99ce-9b0674320bf2
To: /content/abstract.pt
100% 825M/825M [00:10<00:00, 79.2MB/s]
Downloading…
From: https://drive.google.com/uc?id=1EOG_0_SgjZLb7ikRLtaQKTYC4mx8sBV1
To: /content/nodeidx2paperid.csv
100% 2.94M/2.94M [00:00<00:00, 244MB/s]
Downloading…
From (original): https://drive.google.com/uc?id=1Rm-tajqdiNsfta-
yErgMGH3xIKvrBFv4
From (redirected): https://drive.google.com/uc?id=1Rm-tajqdiNsfta-
yErgMGH3xIKvrBFv4&confirm=t&uuid=0d1ca1d7-ca4e-4b59-a800-ae546cffea9d
To: /content/titleabs.tsv
100% 210M/210M [00:02<00:00, 80.1MB/s]
Downloading…
From (original):
https://drive.google.com/uc?id=1KDaf1DX3AwwYOuBWhjx6l1OBergtwIv4
From (redirected): https://drive.google.com/uc?id=1KDaf1DX3AwwYOuBWhjx6l1OBergtw
Iv4&confirm=t&uuid=3bb19a4d-eb84-4b38-9718-59168c454d13
To: /content/model_checkpoint.pth
100% 266M/266M [00:02<00:00, 105MB/s]
```

[5]:
```
dataset = PygNodePropPredDataset(name='ogbn-arxiv', root='dataset/')
data = dataset[0]
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
data = data.to(device)
```

2 Baseline Model

[6]:
```
class GCN(torch.nn.Module):
    def __init__(self, num_features, hidden_channels, num_classes,␣
 ↪num_additional_layers = 0):
        super(GCN, self).__init__()
        self.conv1 = GCNConv(num_features, hidden_channels)
        self.convs = torch.nn.ModuleList()
        for i in range(num_additional_layers):
            self.convs.append(GCNConv(hidden_channels, hidden_channels))
        self.conv2 = GCNConv(hidden_channels, num_classes)

    def forward(self, x, edge_index):
```

```
        x = self.conv1(x, edge_index)
        x = F.relu(x)
        x = F.dropout(x, p=0.1, training=self.training)
        for conv in self.convs:
            x = conv(x, edge_index)
            x = F.relu(x)
            x = F.dropout(x, p=0.1, training=self.training)
        x = self.conv2(x, edge_index)
        return F.log_softmax(x, dim=1)


class GAT(torch.nn.Module):
    def __init__(self, num_features, hidden_channels, num_classes,␣
 ↪num_additional_layers):
        super(GAT, self).__init__()
        self.conv1 = GATConv(num_features, hidden_channels, heads=8)
        self.convs = torch.nn.ModuleList()
        for i in range(num_additional_layers):
            self.convs.append(GATConv(hidden_channels * 8, hidden_channels * 8,␣
 ↪heads=8))
        self.conv2 = GATConv(hidden_channels * 8, num_classes, heads=1,␣
 ↪concat=True)

    def forward(self, x, edge_index):
        x = self.conv1(x, edge_index)
        x = F.relu(x)
        x = F.dropout(x, p=0.05, training=self.training)
        for conv in self.convs:
            x = conv(x, edge_index)
            x = F.relu(x)
            x = F.dropout(x, p=0.05, training=self.training)
        x = self.conv2(x, edge_index)
        return F.log_softmax(x, dim=1)
```

[7]:

```
split_idx = dataset.get_idx_split()
train_idx, valid_idx, test_idx = split_idx["train"], split_idx["valid"],␣
 ↪split_idx["test"]
```

Please adjust here to choose which baseline model you want to run.

[8]:

```
# model = GAT(num_features=data.num_features,
#              hidden_channels=32,
#              num_classes=dataset.num_classes,
#              num_additional_layers = 0).to(device)
model = GCN(num_features=data.num_features,
            hidden_channels=64,
            num_classes=dataset.num_classes,
```

```
                num_additional_layers = 0).to(device)
```

[9]:
```
data = data.to(device)
optimizer = torch.optim.Adam(model.parameters(), lr=0.001, weight_decay=5e-4)
def train():
    model.train()
    optimizer.zero_grad()
    out = model(data.x, data.edge_index)
    loss = F.nll_loss(out[train_idx], data.y.squeeze(1)[train_idx])
    loss.backward()
    optimizer.step()
    return loss

for epoch in range(5000):
    loss = train()
    if epoch % 200 == 0:
        print(f'Epoch: {epoch}, Loss: {loss.item()}')
```

```
Epoch: 0, Loss: 3.6995298862457275
Epoch: 200, Loss: 1.86920964717865
Epoch: 400, Loss: 1.570776104927063
Epoch: 600, Loss: 1.4867894649505615
Epoch: 800, Loss: 1.4529249668121338
Epoch: 1000, Loss: 1.4275974035263062
Epoch: 1200, Loss: 1.4151098728179932
Epoch: 1400, Loss: 1.4044201374053955
Epoch: 1600, Loss: 1.394255518913269
Epoch: 1800, Loss: 1.3854221105575562
Epoch: 2000, Loss: 1.3803577423095703
Epoch: 2200, Loss: 1.376020908355713
Epoch: 2400, Loss: 1.3719271421432495
Epoch: 2600, Loss: 1.3674284219741821
Epoch: 2800, Loss: 1.3633217811584473
Epoch: 3000, Loss: 1.3619248867034912
Epoch: 3200, Loss: 1.3591666221618652
Epoch: 3400, Loss: 1.3550549745559692
Epoch: 3600, Loss: 1.3542957305908203
Epoch: 3800, Loss: 1.3518812656402588
Epoch: 4000, Loss: 1.3516632318496704
Epoch: 4200, Loss: 1.3473693132400513
Epoch: 4400, Loss: 1.3450846672058105
Epoch: 4600, Loss: 1.3443336486816406
Epoch: 4800, Loss: 1.344861626625061
```

[10]:
```
def test():
    model.eval()
    out = model(data.x, data.edge_index)
```

```
    pred = out.argmax(dim=1)  # Get the index of the max log-probability
    correct = pred[test_idx] == data.y.squeeze(1)[test_idx]  # Compare against␣
↪ground-truth labels
    acc = int(correct.sum()) / len(test_idx)  # Compute accuracy
    return acc

accuracy = test()
print(f'Validation Accuracy: {accuracy}')
```

```
Validation Accuracy: 0.5363866427998272
```

3 Getting LM embedding

[11]:
```
## reads raw texts for each node
df = pd.read_csv('/content/titleabs.tsv', header = None, sep = '\t',
                 names = ["id", "title", "abstract"])
df.head()
```

[11]:
```
        id                                              title  \
0   200971          ontology as a source for rule generation
1   549074  a novel methodology for thermal analysis a 3 d…
2   630234  spreadsheets on the move an evaluation of mobi…
3   803423  multi view metric learning for multi view vide…
4  1102481    big data analytics in future internet of things

                                            abstract
0  This paper discloses the potential of OWL (Web…
1  The semiconductor industry is reaching a fasci…
2  The power of mobile devices has increased dram…
3  Traditional methods on video summarization are…
4  Current research on Internet of Things (IoT) m…
```

3.1 Sentence transformer (DO NOT RUN)

Getting the sentence transformer embedding. We have already computed these and stored them in drive. Please do not run it again (if you are starting fresh, only run this once).

[9]:
```
df['combined'] = df['title'] + '\n' + df['abstract']
model = SentenceTransformer('all-mpnet-base-v2')
```

```
/usr/local/lib/python3.10/dist-packages/huggingface_hub/utils/_token.py:88:
UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab
(https://huggingface.co/settings/tokens), set it as secret in your Google Colab
and restart your session.
You will be able to reuse this secret in all of your notebooks.
```

```
Please note that authentication is recommended but still optional to access
public models or datasets.
  warnings.warn(

modules.json:   0%|          | 0.00/349 [00:00<?, ?B/s]

config_sentence_transformers.json:   0%|          | 0.00/116 [00:00<?, ?B/s]

README.md:   0%|          | 0.00/10.6k [00:00<?, ?B/s]

sentence_bert_config.json:   0%|          | 0.00/53.0 [00:00<?, ?B/s]

config.json:   0%|          | 0.00/571 [00:00<?, ?B/s]

model.safetensors:   0%|          | 0.00/438M [00:00<?, ?B/s]

tokenizer_config.json:   0%|          | 0.00/363 [00:00<?, ?B/s]

vocab.txt:   0%|          | 0.00/232k [00:00<?, ?B/s]

tokenizer.json:   0%|          | 0.00/466k [00:00<?, ?B/s]

special_tokens_map.json:   0%|          | 0.00/239 [00:00<?, ?B/s]

1_Pooling/config.json:   0%|          | 0.00/190 [00:00<?, ?B/s]
```

[ ]:
```
abstract_embedding = model.encode(df["abstract"].tolist(), show_progress_bar =
 ↪True)
```

```
Batches:   0%|          | 0/5617 [00:00<?, ?it/s]
```

[ ]:
```
torch.save(abstract_embedding, "abstract.pt")
```

[ ]:
```
array([[ 0.06709803, -0.00260303, -0.03525066, …,  0.01122417,
         0.00227784, -0.0024321 ],
       [ 0.05002695, -0.06211789, -0.00733226, …, -0.04582901,
        -0.05350998, -0.00621984],
       [-0.01707286, -0.01199244, -0.01796226, …,  0.05112026,
         0.00148729, -0.00747634],
       …,
       [-0.01329558,  0.11748262, -0.03455109, …,  0.00232268,
        -0.03262008, -0.04931226],
       [ 0.02520715,  0.0554713 , -0.00194554, …,  0.02012073,
        -0.00034638, -0.03606958],
       [ 0.00272224,  0.01826248, -0.0160021 , …,  0.03682491,
        -0.02521036, -0.06207232]], dtype=float32)
```

[ ]:
```
all_embedding = model.encode(df["combined"].tolist(), show_progress_bar = True)
```

```
Batches:   0%|          | 0/5617 [00:00<?, ?it/s]
```

[ ]:
```
torch.save(all_embedding, "all.pt")
```

[ ]:
```
title_embedding = model.encode(df["title"].tolist(), show_progress_bar = True)
```

```
Batches:   0%|          | 0/5617 [00:00<?, ?it/s]
```

[ ]:
```
torch.save(title_embedding, "title.pt")
```

[ ]:
```
log_abstract = {
    "title": title_embedding,
    "abstract": abstract_embedding,
    "all": all_embedding,
}
torch.save(log_abstract, "log_abstract.pt")
```

[ ]:
```
from google.colab import files
files.download("/content/log_abstract.pt")
```

4 Approach 1: Using LM feature for GNN

[12]:
```
nodeid2paperid = pd.read_csv('/content/nodeidx2paperid.csv')
nodeid2paperid.head()
```

[12]:

	node idx	paper id
0	0	9657784
1	1	39886162
2	2	116214155
3	3	121432379
4	4	231147053

[13]:
```
abstract_embedding = torch.load('abstract.pt')
all_embedding = torch.load('/content/all.pt')
```

NOTE: `lm_emb = all_embedding` use the combined embedding for title + abstract. change it to `lm_emb = abstract_embedding` to use embedding for abstract only, etc.

[14]:
```
paperids = nodeid2paperid['paper id'].tolist()
paperids_df = df.id.tolist()
paperid2dfid = {paperid: i for i, paperid in enumerate(paperids_df)}
dfids = [paperid2dfid[i] for i in paperids]
lm_emb = all_embedding ## TODO: could change this
x_lm = torch.tensor(lm_emb[torch.tensor(dfids)]).to(device)
data.x_lm = x_lm.to(device)
data.num_lm_features = 768

## could add
data.num_lm_features = 768 + 128
data.x_lm = torch.cat((data.x, data.x_lm), dim=1)
```

[15]:
```python
# model = GAT(num_features=data.num_lm_features,
#             hidden_channels=32,
#             num_classes=dataset.num_classes,
#             num_additional_layers = 0).to(device)
model = GCN(num_features=data.num_lm_features,
            hidden_channels=64,
            num_classes=dataset.num_classes,
            num_additional_layers = 0).to(device)
```

[16]:
```python
data = data.to(device)
optimizer = torch.optim.Adam(model.parameters(), lr=0.001, weight_decay=5e-5)
def train():
    model.train()
    optimizer.zero_grad()
    out = model(data.x_lm, data.edge_index)
    loss = F.nll_loss(out[train_idx], data.y.squeeze(1)[train_idx])
    with torch.no_grad():
      val_loss = F.nll_loss(out[valid_idx], data.y.squeeze(1)[valid_idx])
    loss.backward()
    optimizer.step()
    return loss, val_loss

for epoch in range(2000):  ## change this to 5000, here is 2000 for saving␣
 ↪time, we ran it for 5000 epochs for all experiments.
    loss, val_loss = train()
    if epoch % 200 == 0:
        print(f'Epoch: {epoch}, Loss: {loss.item()}, valid loss: {val_loss.
 ↪item()}')
```

```
Epoch: 0, Loss: 3.777076005935669, valid loss: 3.754049301147461
Epoch: 200, Loss: 1.1956003904342651, valid loss: 1.173519492149353
Epoch: 400, Loss: 1.0399047136306763, valid loss: 1.0381194353103638
Epoch: 600, Loss: 0.9864274263381958, valid loss: 1.001455545425415
Epoch: 800, Loss: 0.9527455568313599, valid loss: 0.9856254458427429
Epoch: 1000, Loss: 0.9250034093856812, valid loss: 0.9769924879074097
Epoch: 1200, Loss: 0.9044860601425171, valid loss: 0.9672080278396606
Epoch: 1400, Loss: 0.8855103254318237, valid loss: 0.9640610814094543
Epoch: 1600, Loss: 0.8698300719261169, valid loss: 0.9623537659645081
Epoch: 1800, Loss: 0.8521376252174377, valid loss: 0.9573588371276855
```

[17]:
```python
def test():
    model.eval()
    out = model(data.x_lm, data.edge_index)
    pred = out.argmax(dim=1)  # Get the index of the max log-probability
    correct = pred[test_idx] == data.y.squeeze(1)[test_idx]  # Compare against␣
 ↪ground-truth labels
    acc = int(correct.sum()) / len(test_idx)  # Compute accuracy
```

```
    return acc

accuracy = test()
print(f'Validation Accuracy: {accuracy}')
```

```
Validation Accuracy: 0.6941135320864967
```

5 Directly optimizing an LM for classification

Here, we directly use an LM to predict the paper class from abstract + title.

[13]:
```
from copy import deepcopy
train_df = df.iloc[
    [paperid2dfid[i] for i in nodeid2paperid.iloc[train_idx.tolist(), :]["paper␣
 ↪id"].tolist()],
    :
]
train_df.loc[:, "label"] = deepcopy(data.y.squeeze(1)[train_idx].cpu().numpy())
dev_df = df.iloc[
    [paperid2dfid[i] for i in nodeid2paperid.iloc[valid_idx.tolist(), :]["paper␣
 ↪id"].tolist()],
    :
]
dev_df.loc[:, "label"] = deepcopy(data.y.squeeze(1)[valid_idx].cpu().numpy())
test_df = df.iloc[
    [paperid2dfid[i] for i in nodeid2paperid.iloc[test_idx.tolist(), :]["paper␣
 ↪id"].tolist()],
    :
]
test_df.loc[:, "label"] = deepcopy(data.y.squeeze(1)[test_idx].cpu().numpy())
```

```
<ipython-input-13-7af28983a24b>:7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  train_df.loc[:, "label"] =
deepcopy(data.y.squeeze(1)[train_idx].cpu().numpy())
<ipython-input-13-7af28983a24b>:12: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  dev_df.loc[:, "label"] = deepcopy(data.y.squeeze(1)[valid_idx].cpu().numpy())
<ipython-input-13-7af28983a24b>:17: SettingWithCopyWarning:
```

```
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  test_df.loc[:, "label"] = deepcopy(data.y.squeeze(1)[test_idx].cpu().numpy())
```

[14]:
```python
class simple_dataset(Dataset):
  def __init__(self, df, tokenizer):
    df = df.reset_index(drop=True)
    self.df = df
    self.paperid2nodeid = {nodeid2paperid["paper id"][i]: nodeid2paperid["node␣
↪idx"][i] for i in range(len(nodeid2paperid))}

    self.tokenizer = tokenizer
  def __len__(self):
    return  len(self.df)

  def __getitem__(self, idx):
    row = self.df.iloc[idx]
    title = row['title']
    abstract = row['abstract']
    text = title + "\n" + abstract
    output = self.tokenizer(text, return_tensors="pt", padding="max_length",
                            truncation=True,
                            max_length = 256)
    for k in output: output[k] = torch.squeeze(output[k])
    output["label"] = torch.tensor(row['label'])
    output["nodeid"] = torch.tensor(self.paperid2nodeid[row['id']])
    return output

class classification_lm(torch.nn.Module):
  def __init__(self, base, num_classes):
    super(classification_lm, self).__init__()
    self.base = base
    self.head = torch.nn.Linear(768, num_classes)

  def forward(self, input_ids, attention_mask):
    outputs = self.base(input_ids, attention_mask=attention_mask)
    # return self.head(outputs.pooler_output)
    return self.head(outputs.last_hidden_state[:, 0])

model_name = "distilbert/distilbert-base-uncased"

tokenizer = AutoTokenizer.from_pretrained(model_name)
df_data_train = simple_dataset(train_df, tokenizer)
df_data_dev = simple_dataset(dev_df, tokenizer)
```

```python
df_data_test = simple_dataset(test_df, tokenizer)
train_loader = DataLoader(df_data_train, batch_size=8, shuffle = True)
dev_loader = DataLoader(df_data_dev, batch_size=8, shuffle = True)
test_loader = DataLoader(df_data_test, batch_size=8, shuffle = True)

lm_base = AutoModel.from_pretrained(model_name)
lm = classification_lm(lm_base, 40)
lm.to(device)

def train_lm(train_loader, dev_loader, test_loader):
  criterion = torch.nn.CrossEntropyLoss()
  optimizer = torch.optim.Adam(lm.parameters(), lr=2e-5)
  for epoch in range(3):
    lm.train()
    tbar_train = tqdm(train_loader, position=0, leave=True)
    for batch in tbar_train:
      optimizer.zero_grad()
      input_ids = batch.input_ids.to(device)
      attention_mask = batch.attention_mask.to(device)
      logits = lm(input_ids, attention_mask=attention_mask)
      loss = criterion(logits, batch.label.to(device))
      loss.backward()
      optimizer.step()
    dev_acc = eval(lm, dev_loader)
    test_acc = eval(lm, test_loader)
    print(f"\n Epoch {epoch}, dev acc {dev_acc}, test acc {test_acc} \n")

def eval(lm, loader):
  with torch.no_grad():
    lm.eval()
    preds = []
    labels = []
    tbar = tqdm(loader, position=0, leave=True)
    for batch in tbar:
      input_ids = batch.input_ids.to(device)
      attention_mask = batch.attention_mask.to(device)
      outputs = lm(input_ids, attention_mask=attention_mask)
      pred = outputs.argmax(dim=1).detach().cpu()
      preds.append(pred)
      labels.append(batch.label)
    acc = (torch.cat(preds) == torch.cat(labels)).sum().item() / len(torch.
 ↪cat(preds))
    return acc

# train_lm(train_loader, dev_loader, test_loader)
```

```
tokenizer_config.json:   0%|          | 0.00/28.0 [00:00<?, ?B/s]
```

```
config.json:   0%|          | 0.00/483 [00:00<?, ?B/s]

vocab.txt:   0%|          | 0.00/232k [00:00<?, ?B/s]

tokenizer.json:   0%|          | 0.00/466k [00:00<?, ?B/s]

model.safetensors:   0%|          | 0.00/268M [00:00<?, ?B/s]
```

[ ]:
```
torch.save(
    {
    'model_state_dict': lm.state_dict(),
    },
    '/content/model_checkpoint.pth')
```

[ ]:
```
from google.colab import files
files.download('/content/model_checkpoint.pth')
```

```
<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>
```

6 Approach 2: Contrastive Learning

Specify your config below; whether to use approach 1 (`use_lm_feature=True`) for contrastive learning

[15]:
```
use_lm_feature = True
if use_lm_feature:
  data.emb = data.x_lm
  data.emb_size = data.num_lm_features
else:
  data.emb = data.x
  data.emb_size = data.num_features
```

[16]:
```
class GCN_con(torch.nn.Module):
    def __init__(self, num_features, hidden_channels, num_classes,␣
 ↪num_additional_layers = 0):
        super(GCN_con, self).__init__()
        self.conv1 = GCNConv(num_features, hidden_channels)
        self.convs = torch.nn.ModuleList()
        for i in range(num_additional_layers):
            self.convs.append(GCNConv(hidden_channels, hidden_channels))
        self.conv2 = GCNConv(hidden_channels, num_classes)
        self.proj = proj(hidden_channels)

    def forward(self, x, edge_index):
        x = self.conv1(x, edge_index)
        x = F.relu(x)
        x = F.dropout(x, p=0.1, training=self.training)
```

```
        for conv in self.convs:
            x = conv(x, edge_index)
            x = F.relu(x)
            x = F.dropout(x, p=0.1, training=self.training)
        y = self.conv2(x, edge_index)
        return F.log_softmax(y, dim=1), self.proj(x)


class GAT_con(torch.nn.Module):
    def __init__(self, num_features, hidden_channels, num_classes,␣
↪num_additional_layers):
        super(GAT_con, self).__init__()
        self.conv1 = GATConv(num_features, hidden_channels, heads=8)
        self.convs = torch.nn.ModuleList()
        for i in range(num_additional_layers):
            self.convs.append(GATConv(hidden_channels * 8, hidden_channels * 8,␣
↪heads=8))
        self.conv2 = GATConv(hidden_channels * 8, num_classes, heads=1,␣
↪concat=True)
        self.proj = proj(hidden_channels * 8)

    def forward(self, x, edge_index):
        x = self.conv1(x, edge_index)
        x = F.relu(x)
        x = F.dropout(x, p=0.05, training=self.training)
        for conv in self.convs:
            x = conv(x, edge_index)
            x = F.relu(x)
            x = F.dropout(x, p=0.05, training=self.training)
        y = self.conv2(x, edge_index)
        return F.log_softmax(y, dim=1), self.proj(x)

class proj(torch.nn.Module):
    def __init__(self, input_dim, proj_dim=128):
        super().__init__()
        self.linear = torch.nn.Linear(input_dim, proj_dim)
        self.activation = torch.nn.ReLU()

    def forward(self, x):
        return self.linear(self.activation(x))
```

[18]:

```
gnn = GCN_con(num_features=data.num_lm_features,
            hidden_channels=64,
            num_classes=dataset.num_classes,
            num_additional_layers = 0).to(device)
# gnn = GAT_con(num_features=data.emb_size,
#              hidden_channels=32,
```

```
#              num_classes=dataset.num_classes,
#              num_additional_layers = 0).to(device)
```

[19]:
```
data = data.to(device)
optimizer = torch.optim.Adam(gnn.parameters(), lr=0.001, weight_decay=5e-4)
def train():
    gnn.train()
    optimizer.zero_grad()
    out, _ = gnn(data.emb, data.edge_index)
    loss = F.nll_loss(out[train_idx], data.y.squeeze(1)[train_idx])
    loss.backward()
    optimizer.step()
    return loss

for epoch in range(5000):
    loss = train()
    if epoch % 200 == 0:
        print(f'Epoch: {epoch}, Loss: {loss.item()}')
```

```
Epoch: 0, Loss: 3.7117953300476074
Epoch: 200, Loss: 1.2116265296936035
Epoch: 400, Loss: 1.0787447690963745
Epoch: 600, Loss: 1.0346462726593018
Epoch: 800, Loss: 1.0126497745513916
Epoch: 1000, Loss: 0.9944066405296326
Epoch: 1200, Loss: 0.9831110239028931
Epoch: 1400, Loss: 0.9736768007278442
Epoch: 1600, Loss: 0.9667458534240723
Epoch: 1800, Loss: 0.9636861681938171
Epoch: 2000, Loss: 0.9560675024986267
Epoch: 2200, Loss: 0.9512990117073059
Epoch: 2400, Loss: 0.9489309191703796
Epoch: 2600, Loss: 0.9468228220939636
Epoch: 2800, Loss: 0.9446706771850586
Epoch: 3000, Loss: 0.941193699836731
Epoch: 3200, Loss: 0.9405066967010498
Epoch: 3400, Loss: 0.9387608766555786
Epoch: 3600, Loss: 0.9374501705169678
Epoch: 3800, Loss: 0.9350242614746094
Epoch: 4000, Loss: 0.9354428648948669
Epoch: 4200, Loss: 0.9335297346115112
Epoch: 4400, Loss: 0.9333879351615906
Epoch: 4600, Loss: 0.9317354559898376
Epoch: 4800, Loss: 0.9301475286483765
```

Note that here, `lm.load_state_dict(torch.load('/content/model_checkpoint.pth')["model_state_dict"]` loads the LM trained on node classification. If you comment it out, we will not be using a fine-tuned LM for contrastive learning.

[20]:
```
model_name = "distilbert/distilbert-base-uncased"

tokenizer = AutoTokenizer.from_pretrained(model_name)
df_data_train = simple_dataset(train_df, tokenizer)
df_data_dev = simple_dataset(dev_df, tokenizer)
df_data_test = simple_dataset(test_df, tokenizer)
train_loader = DataLoader(df_data_train, batch_size=8, shuffle = True)
dev_loader = DataLoader(df_data_dev, batch_size=8, shuffle = True)
test_loader = DataLoader(df_data_test, batch_size=8, shuffle = True)

lm_base = AutoModel.from_pretrained(model_name)
lm = classification_lm(lm_base, 40)

## LOAD LM todo
lm.load_state_dict(torch.load('/content/model_checkpoint.
↪pth')["model_state_dict"])
lm.proj = proj(768)

lm.to(device)
```

[20]:
```
classification_lm(
  (base): DistilBertModel(
    (embeddings): Embeddings(
      (word_embeddings): Embedding(30522, 768, padding_idx=0)
      (position_embeddings): Embedding(512, 768)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (transformer): Transformer(
      (layer): ModuleList(
        (0-5): 6 x TransformerBlock(
          (attention): MultiHeadSelfAttention(
            (dropout): Dropout(p=0.1, inplace=False)
            (q_lin): Linear(in_features=768, out_features=768, bias=True)
            (k_lin): Linear(in_features=768, out_features=768, bias=True)
            (v_lin): Linear(in_features=768, out_features=768, bias=True)
            (out_lin): Linear(in_features=768, out_features=768, bias=True)
          )
          (sa_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
          (ffn): FFN(
            (dropout): Dropout(p=0.1, inplace=False)
            (lin1): Linear(in_features=768, out_features=3072, bias=True)
            (lin2): Linear(in_features=3072, out_features=768, bias=True)
            (activation): GELUActivation()
          )
          (output_layer_norm): LayerNorm((768,), eps=1e-12,
elementwise_affine=True)
```

```
        )
      )
    )
  )
  (head): Linear(in_features=768, out_features=40, bias=True)
  (proj): proj(
    (linear): Linear(in_features=768, out_features=128, bias=True)
    (activation): ReLU()
  )
)
```

[21]:
```
def test(gnn):
    gnn.eval()
    out, _ = gnn(data.emb, data.edge_index)
    pred = out.argmax(dim=1)  # Get the index of the max log-probability
    correct = pred[test_idx] == data.y.squeeze(1)[test_idx]  # Compare against␣
 ↪ground-truth labels
    acc = int(correct.sum()) / len(test_idx)  # Compute accuracy
    return acc

accuracy = test(gnn)
print(f'Validation Accuracy: {accuracy}')
```

```
Validation Accuracy: 0.6988251754006954
```

[22]:
```
def contrastive_loss(lm_emb, gnn_emb, temperature=0.07):
    lm_emb_norm = F.normalize(lm_emb, p=2, dim=1)
    gnn_emb_norm = F.normalize(gnn_emb, p=2, dim=1)
    logits_lm_gnn = torch.matmul(lm_emb_norm, gnn_emb_norm.T) / temperature
    logits_gnn_lm = torch.matmul(gnn_emb_norm, lm_emb_norm.T) / temperature
    labels = torch.arange(len(lm_emb)).to(lm_emb.device)
    loss_lm = F.cross_entropy(logits_lm_gnn, labels)
    loss_gnn = F.cross_entropy(logits_gnn_lm, labels)
    return (loss_lm + loss_gnn) / 2

def train_contrastive(train_loader, dev_loader, test_loader, lm, gnn):
  gnn.train()
  criterion = torch.nn.CrossEntropyLoss()
  # optimizer = torch.optim.Adam(chain(lm.parameters(), gnn.parameters()),␣
 ↪lr=2e-5)
  optimizer_gnn = torch.optim.Adam(gnn.parameters(), lr=0.001, weight_decay =␣
 ↪5e-4)
  optimizer_lm = torch.optim.Adam(lm.parameters(), lr=2e-5)
  count = 0
  for epoch in range(2):
    lm.train()
    tbar_train = tqdm(train_loader, position=0, leave=True)
```

```
    gnn_acc = 0
    for batch in tbar_train:
      gnn.train()
      lm.train()
      ## get lm embedding
      optimizer.zero_grad()
      input_ids = batch.input_ids.to(device)
      attention_mask = batch.attention_mask.to(device)
      lm_emb = lm.proj(lm.base(input_ids, attention_mask)["last_hidden_state"][:
↪, 0])

      ## get gnn embedding
      logits_gnn_all, gnn_emb_all = gnn(data.emb, data.edge_index)
      gnn_emb = gnn_emb_all[batch.nodeid]
      logits_gnn = logits_gnn_all[batch.nodeid]

      ## contrastive loss
      infonce_loss = contrastive_loss(lm_emb, gnn_emb)

      ## classification loss
      logits_lm = lm(input_ids, attention_mask)
      lm_classification_loss = criterion(logits_lm, batch.label.to(device))
      # gnn_classification_loss = criterion(logits_gnn, batch.label.to(device))
      gnn_classification_loss = F.nll_loss(logits_gnn_all[train_idx], data.y.
↪squeeze(1)[train_idx].to(device))


      ## add loss
      loss = lm_classification_loss + gnn_classification_loss + infonce_loss
      loss.backward()
      # optimizer.step()
      optimizer_lm.step()
      optimizer_gnn.step()

      count += 1
      if count % 1000 == 0:
        gnn_acc = test(gnn)

      tbar_train.set_postfix(loss=loss.item(), gnn_acc = gnn_acc)
    dev_acc_lm = eval(lm, dev_loader)
    test_acc_lm = eval(lm, test_loader)
    gnn_acc = test(gnn)
    print(f"\n LM: Epoch {epoch}, dev acc {dev_acc_lm}, test acc {test_acc_lm}␣
↪\n")
    print(f"\n GNN: Epoch {epoch}, acc {gnn_acc} \n")
  return lm, gnn
lm, gnn = train_contrastive(train_loader, dev_loader, test_loader, lm, gnn)
```

```
100%|      | 11368/11368 [28:11<00:00,  6.72it/s, gnn_acc=0.699, loss=6.35]
100%|      | 3725/3725 [01:44<00:00, 35.58it/s]
100%|      | 6076/6076 [02:51<00:00, 35.45it/s]

 LM: Epoch 0, dev acc 0.07627772744051814, test acc 0.05861778079542415


 GNN: Epoch 0, acc 0.6990720737403041
```

7 t-sne visualization

[ ]:
```python
use_lm_feature = True
if use_lm_feature:
  data.emb = data.x_lm
  data.emb_size = data.num_lm_features
else:
  data.emb = data.x
  data.emb_size = data.num_features
```

[ ]:
```python
gnn = GCN_con(num_features=data.emb_size,
            hidden_channels=64,
            num_classes=dataset.num_classes,
            num_additional_layers = 0).to(device)
```

[ ]:
```python
data = data.to(device)
optimizer = torch.optim.Adam(gnn.parameters(), lr=0.001, weight_decay=5e-4)
def train():
    gnn.train()
    optimizer.zero_grad()
    out, _ = gnn(data.emb, data.edge_index)
    loss = F.nll_loss(out[train_idx], data.y.squeeze(1)[train_idx])
    loss.backward()
    optimizer.step()
    return loss

for epoch in range(5000):
    loss = train()
    if epoch % 200 == 0:
        print(f'Epoch: {epoch}, Loss: {loss.item()}')
```

```
Epoch: 0, Loss: 3.727536201477051
Epoch: 200, Loss: 1.1924127340316772
Epoch: 400, Loss: 1.068192958831787
Epoch: 600, Loss: 1.028207540512085
```

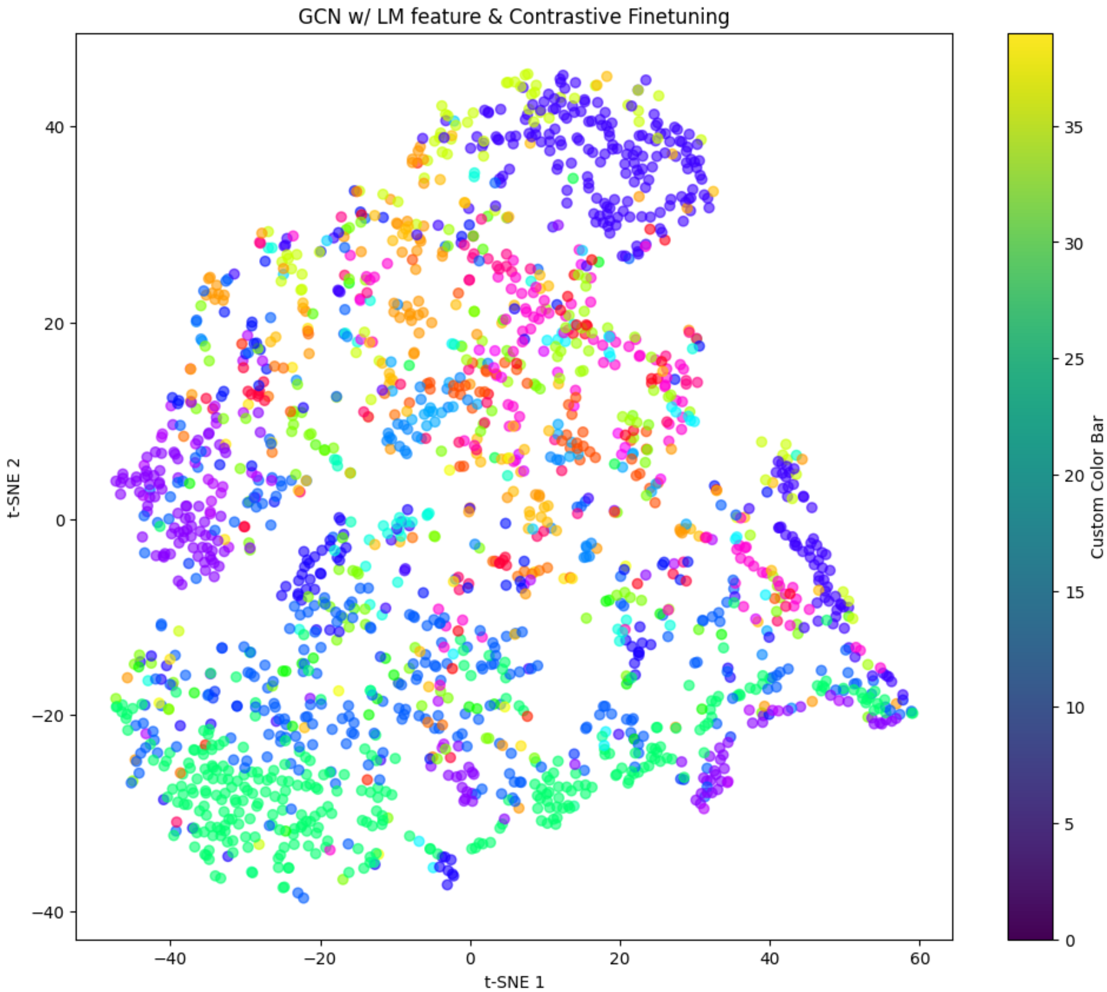
```
Epoch: 800, Loss: 1.0056724548339844
Epoch: 1000, Loss: 0.9896546006202698
Epoch: 1200, Loss: 0.977752149105072
Epoch: 1400, Loss: 0.9703963994979858
Epoch: 1600, Loss: 0.9649154543876648
Epoch: 1800, Loss: 0.96042799949646
Epoch: 2000, Loss: 0.9545736312866211
Epoch: 2200, Loss: 0.9524317383766174
Epoch: 2400, Loss: 0.9485260248184204
Epoch: 2600, Loss: 0.9458579421043396
Epoch: 2800, Loss: 0.944022536277771
Epoch: 3000, Loss: 0.9427605271339417
Epoch: 3200, Loss: 0.9391101598739624
Epoch: 3400, Loss: 0.9373682141304016
Epoch: 3600, Loss: 0.9361087083816528
Epoch: 3800, Loss: 0.9351071119308472
Epoch: 4000, Loss: 0.9341258406639099
Epoch: 4200, Loss: 0.9325274229049683
Epoch: 4400, Loss: 0.9323749542236328
Epoch: 4600, Loss: 0.9316297769546509
Epoch: 4800, Loss: 0.9304457902908325
```

[36]:

```
gnn.eval()

_, embeddings = gnn(data.emb, data.edge_index)

embeddings_np = embeddings.detach().cpu().numpy()
labels_np = data.y.squeeze().cpu().numpy()
random_idx = np.random.choice(len(embeddings_np), 2000, replace=False)
embeddings_np = embeddings_np[random_idx]
labels_np = labels_np[random_idx]

tsne = TSNE(n_components=2, random_state=0, verbose = 1)
X_reduced = tsne.fit_transform(embeddings_np)

# Plotting
fig, ax = plt.subplots(figsize=(12, 10))
# plt.figure(figsize=(12, 10))
scatter = plt.scatter(X_reduced[:, 0], X_reduced[:, 1], c=labels_np,
 ↪cmap='hsv', alpha=0.6)
plt.colorbar(scatter)

# import matplotlib.colors as mcolors
# # Generate a color bar with a different colormap
# # Here 'hot' is used for demonstration; replace it with any colormap you
 ↪prefer
# norm = mcolors.Normalize(vmin=0, vmax=39)
```

```python
# sm = plt.cm.ScalarMappable(cmap='viridis', norm=norm)
# sm.set_array([])

# # Add the color bar
# cbar = plt.colorbar(sm, ax=ax, orientation='vertical')
# cbar.set_label('Custom Color Bar')

## note: you should manually change the title.
plt.title('GCN w/ LM feature & Contrastive Finetuning')
plt.xlabel('t-SNE 1')
plt.ylabel('t-SNE 2')
plt.show()
```

```
[t-SNE] Computing 91 nearest neighbors…
[t-SNE] Indexed 2000 samples in 0.002s…
[t-SNE] Computed neighbors for 2000 samples in 0.065s…
[t-SNE] Computed conditional probabilities for sample 1000 / 2000
[t-SNE] Computed conditional probabilities for sample 2000 / 2000
[t-SNE] Mean sigma: 0.482092
[t-SNE] KL divergence after 250 iterations with early exaggeration: 72.066559
[t-SNE] KL divergence after 1000 iterations: 1.186665
```


GCN w/ LM feature & Contrastive Finetuning
t-SNE 2
t-SNE 1
Custom Color Bar
40
20
0
−20
−40
−40
−20
0
20
40
60
0
5
10
15
20
25
30
35