All in One: Multi-Task Prompting for Graph Neural Networks

May 11, 2024

Introduction

Background

- Now many studies turn to "**pre-training and fine-tuning**" , which means pre-training a graph model with easily accessible data, and then transferring the graph knowledge to a new domain or task via tuning the last layer of the pre-trained model.
- Although much progress has been achieved on pre-training strategies, there still **exists a huge gap between these pretexts and multiple downstream tasks.**

Motivation

Inspired by the prompt learning in natural language processing (NLP),We study the prompting topic for graphs **with the motivation of filling the gap between pretrained models and various graph tasks.**

Prompt learning

It has shown notable effectiveness in **generalizing pre-trained language models to a wide range of language applications.** For example, a sentiment task like "KDD2023 will witness many high-quality papers.I feel so [MASK]" can be easily transferred to a word prediction task via a preset prompt ("I feel so [MASK]").

A solution

A promising solution to the above problems is to extend "pre-training and fine-tuning" to "pre-training, prompting, and fine-tuning".

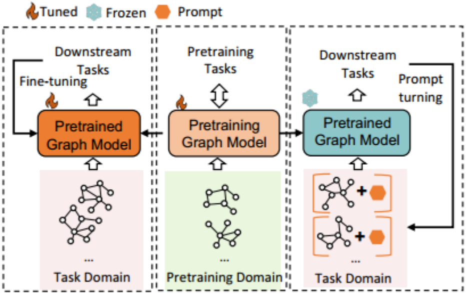


Figure 1: Fine-tuning, Pre-training, and Prompting.

Challenges

- Designing the graph prompt is more intractable than language prompts.

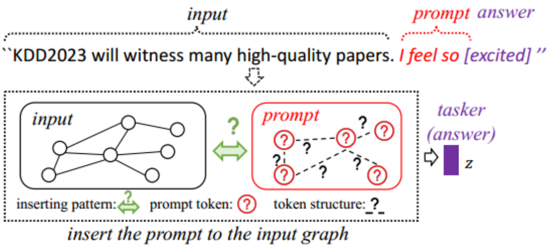


Figure 2: Our graph prompt inspired by the language prompt.

Challenges

- There is a huge difficulty in reconciling downstream problems to the pre-training task.
- learning a reliable prompt usually needs huge manpower and is more sensitive to prompt initialization in the multi-task setting.

Contributions

- unify the format of the language prompt and graph prompt in one way so that we can smoothly transfer the prompt idea from NLP to graphs, then we **design the graph prompt from prompt tokens, token structures,and prompt inserting patterns.**
- propose to **reformulate node-level and edge-level tasks to graph-level tasks by induced graphs from original graphs.**
- introduce the **meta-learning technique** over multiple tasks to learn better prompts.
- analyze why our method works and confirm the effectiveness of our method via **extensive experiments.**

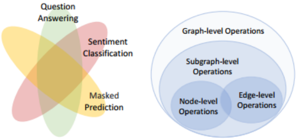
Reformulating Downstream Tasks

Why Reformulate Downstream Tasks

Things are a little complicated in the graph domain since graph-related tasks are far from similar.

Why Reformulate to the Graph Level.

Compared with node-level and edge-level tasks, graph-level tasks are more general and contain the largest overlapping task sub-spaces for knowledge transfer, which has been adopted as the mainstream task in many graph pre-training models.



(a) NLP tasks (b) graph tasks

Figure 3: Task space in NLP and graph

Reformulating Downstream Tasks

How to Reformulate Downstream Tasks.

We reformulate node-level and edge-level tasks to graph-level tasks by **building induced graphs for nodes and edges**, respectively.

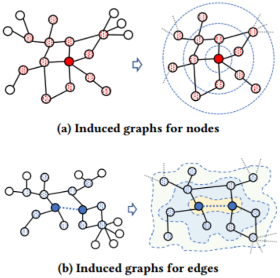


Figure 4: Induced graphs for nodes and edges

Prompt Graph Design

Prompt Tokens

$\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V} = \{v_1, v_2, \cdots, v_N\}$,each node has a feature vector denoted by $\mathbf{x}_i \in \mathbb{R}^{1\times d}$;
$\mathcal{G}_p = (\mathcal{P}, \mathcal{S})$ where $\mathcal{P} = \left\{p_1, p_2, \cdots, p_{|\mathcal{P}|}\right\}$, each token $p_i \in \mathcal{P}$ can be represented by a token vector $p_i \in \mathbb{R}^{1\times d}$.

Token Structures

$\mathcal{S} = \{(p_i, p_j) \mid p_i, p_j \in \mathcal{P}\}$ is the token structure denoted by pair-wise relations among tokens.

Prompt Graph Design

we propose three methods to design the prompt token structures:

- the first way is to learn tunable parameters:
  $\mathcal{A} = \bigcup_{\substack{i=1 \\ j=i+1}}^{|\mathcal{P}|-1} \{a_{ij}\}$, a_{ij} is a tunable parameter.
- the second way is to use the **dot product of each prompt token pair** and prune them according to the dot value.
  $(p_i, p_j) \in \mathcal{S} \text{iff} \sigma(\mathbf{p}_i \cdot \mathbf{p}_j) < \delta$.
- the third way is to treat the tokens as independent and then we have $S = \emptyset$.

Prompt Graph Design

Inserting Patterns

w_{ik} is a weighted value to prune unnecessary connections according to the dot product between prompt tokens and input graph nodes:

w_{ik}:

$$w_{ik} = \begin{cases} \sigma(\mathbf{p}_k \cdot \mathbf{x}_i^T), & \text{if} \sigma(\mathbf{p}_k \cdot \mathbf{x}_i^T) > \delta \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

then use a tailored connection like $\hat{\mathbf{x}}_i = \mathbf{x}_i + \sum_{k=1}^{|\mathcal{P}|} w_{ik}\mathbf{p}_k$.
As an alternative and special case, we can also use a more simplified way to get $\hat{\mathbf{x}}_i = \mathbf{x}_i + \sum_{k=1}^{|\mathcal{P}|} \mathbf{p}_k$.

Multi-task Prompting via Meta Learning

Constructing Meta Prompting Tasks

Let τ_i be the i-th task with supporting data $\mathcal{D}^s_{\tau_i}$ and querying data $\mathcal{D}^q_{\tau_i}$.

Applying Meta-learning to Graph Prompting

We use $f_{\theta,\phi|\pi^*}$ to denote the pipeline with prompt graph (θ),pre-trained model (π^*, fixed),and downstream tasker (ϕ). $\mathcal{L}_{\mathcal{D}}(f)$ be the task loss.

Multi-task Prompting via Meta Learning

Then for each task τ_i the corresponding parameters can be updated as follows:

$$\theta_i^k = \theta_i^{k-1} - \alpha \nabla_{\theta_i^{k-1}} \mathcal{L}_{\mathcal{D}^s_{\tau_i}} \left(f_{\theta_i^{k-1}, \phi_i^{k-1} | \pi^*} \right),$$

$$\phi_i^k = \phi_i^{k-1} - \alpha \nabla_{\phi_i^{k-1}} \mathcal{L}_{\mathcal{D}^s_{\tau_i}} \left(f_{\theta_i^{k-1}, \phi_i^{k-1} | \pi^*} \right).$$

minimize the meta loss on various tasks:

$$\theta^*, \phi^* = \arg\min_{\theta,\phi} \sum_{\tau_i \in \mathcal{T}} \mathcal{L}_{\mathcal{D}^q_{\tau_i}} \left(f_{\theta_i, \phi_i | \pi^*} \right)$$

use the second-order gradient to update θ (ϕ) :

$$\begin{aligned}
\theta \leftarrow & \theta - \beta \cdot g_\theta^{second} \\
= & \theta - \beta \cdot \sum_{\tau_i \in \mathcal{T}} \nabla_\theta \mathcal{L}_{\mathcal{D}^q_{\tau_i}} \left(f_{\theta_i, \phi_i | \pi^*} \right) \\
= & \theta - \beta \cdot \sum_{\tau_i \in \mathcal{T}} \nabla_{\theta_i} \mathcal{L}_{\mathcal{D}^q_{\tau_i}} \left(f_{\theta_i, \phi_i | \pi^*} \right) \cdot \nabla_\theta \left(\theta_i \right) \\
= & \theta - \beta \cdot \sum_{\tau_i \in \mathcal{T}} \nabla_{\theta_i} \mathcal{L}_{\mathcal{D}^q_{\tau_i}} \left(f_{\theta_i, \phi_i | \pi^*} \right) \cdot \left(\mathbf{I} - \alpha \mathbf{H}_\theta \left(\mathcal{L}_{\mathcal{D}^s_{\tau_i}} \left(f_{\theta_i, \phi_i | \pi^*} \right) \right) \right)
\end{aligned}$$

Multi-task Prompting via Meta Learning

Overall Learning Process

Algorithm 1: Overall Learning Process

Input: Overall pipeline $f_{\theta,\phi|\pi^*}$ with prompt parameter θ, pre-trained model with frozen parameter π^*, and task head parameterized by ϕ; Multi-task episodes $\mathcal{E} = \{\mathcal{E}_1, \cdots, \mathcal{E}_n\}$;

Output: Optimal pipeline $f_{\theta^*,\phi^*|\pi^*}$

Initialize θ and ϕ

while *not done* **do**

    // inner adaptation

    Sample $\mathcal{E}_i \in \mathcal{E}$ where $\mathcal{E}_i = (\mathcal{T}_{\mathcal{E}_i}, \mathcal{L}_{\mathcal{E}_i}, \mathcal{S}_{\mathcal{E}_i}, \mathcal{Q}_{\mathcal{E}_i})$

    for $\tau_{\triangleleft t} \in \mathcal{T}_{\mathcal{E}_i}, \triangleleft = g, n, \ell$ **do**

        $\theta_{\tau_{\triangleleft t}}, \phi_{\tau_{\triangleleft t}} \leftarrow \theta, \phi$

        $\theta_{\tau_{\triangleleft t}} \leftarrow \theta_{\tau_{\triangleleft t}} - \alpha \nabla_{\theta_{\tau_{\triangleleft t}}} \mathcal{L}^{(\triangleleft)}_{\mathcal{D}^s_{\tau_{\triangleleft t}}} \left(f_{\theta_{\tau_{\triangleleft t}}, \phi_{\tau_{\triangleleft t}} | \pi^*} \right)$

        $\phi_{\tau_{\triangleleft t}} \leftarrow \phi_{\tau_{\triangleleft t}} - \alpha \nabla_{\phi_{\tau_{\triangleleft t}}} \mathcal{L}^{(\triangleleft)}_{\mathcal{D}^s_{\tau_{\triangleleft t}}} \left(f_{\theta_{\tau_{\triangleleft t}}, \phi_{\tau_{\triangleleft t}} | \pi^*} \right)$

    end

    // outer meta update

    Update θ, ϕ by Equation (4) on $\mathcal{Q}_{\mathcal{E}_i} = \{\mathcal{D}^q_{\tau_{\triangleleft t}} | \tau_{\triangleleft t} \in \mathcal{T}_{\mathcal{E}_i}, \triangleleft = g, n, \ell\}$

end

return $f_{\theta^*,\phi^*|\pi^*}$

Why It Works

Flexibility

For any graph G with adjacency matrix A and node feature matrix X, it is proved that we can always learn an appropriate prompt token p^* making the following equation stand:
$\varphi^* (\mathbf{A}, \mathbf{X} + p^*) = \varphi^*(g(\mathbf{A}, \mathbf{X})) + O_{p\varphi}$

$O_{p\varphi}$

It denotes the error bound between the manipulated graph and the prompting graph. This error bound is related to some non-linear layers of the model (unchangeable) and **the quality of the learned prompt (changeable).**

Why It Works

In this paper:

$$\varphi^* \left(\psi(G, G_p^*)\right) = \varphi^*(\mathbf{g}(\mathbf{A}, \mathbf{X})) + O^*_{p\varphi},$$

Table 6: Error bound discussed by section 3.5.2 RED (%): average reduction of each method to the original error.

Prompt Solutions	Token Number	Drop Nodes	Drop Edges	Mask Features	RED (%)
Original Error (without prompt)	0	0.9917	2.6330	6.8209	-
Naive Prompt (Equation 5)	1	0.8710	0.5241	2.0835	66.70↓
Our Prompt Graph (with token, structure, and inserting patterns)	3	0.0875	0.2337	0.6542	90.66↓
	5	0.0685	0.1513	0.4372	93.71↓
	10	0.0859	0.1144	0.2600	95.59↓

Why It Works

Efficiency

- In our prompt learning framework, we only need to **tune the prompt** with the pre-trained graph model frozen, making the training process converge faster than traditional transfer tuning.
- time complexity
- memory friendly

Why It Works

Compatibility

Our method focuses on the input data manipulation and it relies less on the downstream tasks. This means we have a larger tolerance for the task head.

Prompt without Task Head Tuning:
Pretext: GraphCL [36], a graph contrastive learning task that tries to maximize the agreement between a pair of views from the same graph.
Downstream Tasks: node/edge/graph classification.
Prompt Answer: *node classification*. Assume there are k categories for the nodes. We design the prompt graph with k sub-graphs (a.k.a sub-prompts) where each sub-graph has n tokens. Each sub-graph corresponds to one node category. Then we can generate k graph views for all input graphs. We classify the target node with label ℓ ($\ell = 1, 2, \cdots, k$) if the ℓ-th graph view is closest to the induced graph. It is similar to edge/graph classification.

Experiment

- Q1: How effective is our method under the few-shot learning background for multiple graph tasks?
- Q2: How adaptable is our method when transferred to other domains or tasks?
- Q3: How do the main components of our method impact the performance?
- Q4: How efficient is our model compared with traditional approaches?
- Q5: How powerful is our method when we manipulate graphs?

Experiment

Q1: How effective is our method under the few-shot learning background for multiple graph tasks?

Table 2: Node-level performance (%) with 100-shot setting. IMP (%): the average improvement of prompt over the rest.

Training schemes	Methods	Cora			CiteSeer			Reddit			Amazon			Pubmed		
		Acc	F1	AUC	Acc	F1	AUC	Acc	F1	AUC	Acc	F1	AUC	Acc	F1	AUC
supervised	GAT	74.45	73.21	82.97	83.00	83.20	89.33	55.64	62.03	65.38	79.00	73.42	97.81	75.00	77.56	79.72
	GCN	77.55	77.45	83.71	88.00	81.79	94.79	54.38	52.47	56.82	95.36	93.99	96.23	53.64	66.67	69.89
	GT	74.25	75.21	82.04	86.33	85.62	90.13	61.50	61.38	65.56	85.50	86.01	93.01	51.50	67.34	71.91
pre-train + fine-tune	GraphCL+GAT	76.05	76.78	81.96	87.64	88.40	89.93	57.37	66.42	67.43	78.67	72.26	95.65	76.03	77.05	80.02
	GraphCL+GCN	78.75	79.13	84.90	87.49	89.36	90.25	55.00	65.52	74.65	96.00	95.92	98.33	69.37	70.00	74.74
	GraphCL+GT	73.80	74.12	82.77	88.50	88.92	91.25	63.50	66.06	68.04	94.39	93.62	96.97	75.00	78.45	75.05
	SimGRACE+GAT	76.85	77.48	83.37	90.50	91.00	91.56	56.59	65.47	67.77	84.50	84.73	89.69	72.50	68.21	81.97
	SimGRACE+GCN	77.20	76.39	83.13	83.50	84.21	93.22	58.00	55.81	56.93	95.00	94.50	98.03	77.50	75.71	87.53
	SimGRACE+GT	77.40	78.11	82.95	87.50	87.05	91.85	66.00	69.95	70.03	79.00	73.42	97.58	70.50	73.30	74.22
prompt	GraphCL+GAT	76.50	77.26	82.99	88.00	90.52	91.82	57.84	67.02	75.33	80.01	75.62	97.96	77.50	78.26	83.02
	GraphCL+GCN	79.20	79.62	85.29	88.50	91.59	91.43	56.00	68.57	78.82	96.50	96.37	98.70	72.50	72.64	79.57
	GraphCL+GT	75.00	76.00	83.36	91.00	91.00	93.29	65.50	66.08	68.86	95.50	95.43	97.56	76.50	79.11	76.00
	SimGRACE+GAT	76.95	78.51	83.55	93.00	93.14	92.44	57.63	66.64	69.43	95.50	95.43	97.56	73.00	74.04	81.89
	SimGRACE+GCN	77.85	76.57	83.79	90.00	89.47	94.87	59.50	55.97	59.46	95.00	95.24	98.42	78.00	78.22	87.66
	SimGRACE+GT	78.75	79.53	85.03	91.00	91.26	95.62	69.50	71.43	70.75	86.00	83.72	98.24	73.00	73.79	76.64
IMP (%)		1.47	1.94	1.10	3.81	5.25	2.05	3.97	5.04	6.98	4.49	5.84	2.24	8.81	4.55	4.62
Reported Acc of GPPT (Label Ratio 50%)		77.16	–	–	65.81	–	–	92.13	–	–	86.80	–	–	72.23	–	–
appr. Label Ratio of our 100-shot setting		~ 25%			~ 18%			~ 1.7%			~ 7.3%			~ 1.5%		

Figure: Enter Caption

Experiment

Q2: How adaptable is our method when transferred to other domains or tasks?

Table 3: Transferability (%) on Amazon from different level tasks spaces. Source tasks: graph-level tasks and node-level tasks. Target task: edge-level tasks.

Source task	Methods	Accuracy	F1-score	AUC score
graph level	hard	51.50	65.96	40.34
	fine-tune	62.50	70.59	53.91
	prompt	70.50	71.22	74.02
node level	hard	40.50	11.85	29.48
	fine-tune	46.00	54.24	37.26
	prompt	59.50	68.73	55.90

Experiment

Table 4: Transferability (%) from different domains. Source domains: Amazon and PubMed. Target domain: Cora

Source Domains		Amazon			PubMed		
Tasks		hard	fine-tune	prompt	hard	fine-tune	prompt
node level	Acc	26.9	64.14	65.07	55.62	57.93	62.07
	F1	13.11	77.59	80.23	66.33	70.00	76.60
	AUC	17.56	88.79	92.59	82.34	83.34	88.46
edge level	Acc	17.00	77.00	82.00	10.00	90.50	96.50
	F1	10.51	81.58	84.62	2.17	89.73	91.80
	AUC	4.26	94.27	96.19	6.15	93.89	94.70
graph level	Acc	46.00	87.50	88.00	50.00	91.00	95.50
	F1	62.76	89.11	88.12	10.00	93.90	95.60
	AUC	54.23	86.33	94.99	90.85	91.47	98.47

Experiment

Q3: How do the main components of our method impact the performance?

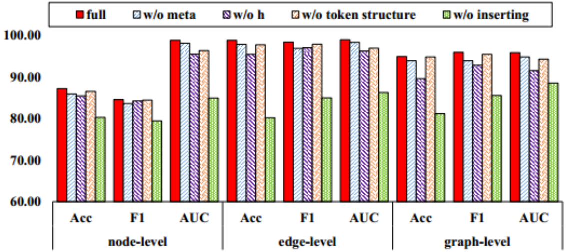


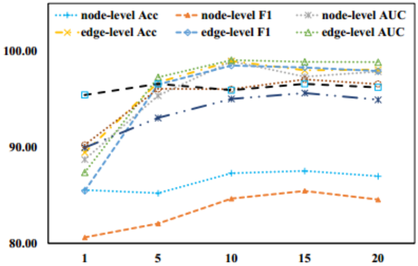
Figure 5: Effectiveness of main components

Experiment

Q4: How efficient is our model compared with traditional approaches?

Table 5: Tunable parameters comparison. RED (%): average reduction of the prompt method to others.

Methods	Cora	CiteSeer	Reddit	Amazon	Pubmed	RED (%)
GAT	~ 155K	~ 382K	~ 75K	~ 88K	~ 61K	95.4↓
GCN	~ 154K	~ 381K	~ 75K	~ 88K	~ 61K	95.4↓
GT	~ 615K	~ 1.52M	~ 286K	~ 349K	~ 241K	98.8↓
prompt	~ 7K	~ 19K	~ 3K	~ 4K	~ 3K	–



Experiment

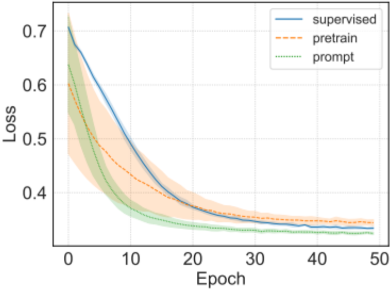


Figure 7: Training losses with epochs. Mean values and 65% confidence intervals by 5 repeats with different seeds.