Meta-Learning with Graph Neural Networks: Methods and Applications

Debmalya Mandal[1], Sourav Medya[2], Brian Uzzi[2], Charu Aggarwal[3]
[1]Data Science Institute, Columbia University, New York
[2]Kellogg School of Management, Northwestern University
[3]IBM T. J. Watson Research Center, Yorktown Heights, New York
dm3557@columbia.edu, {sourav.medya,uzzi}@kellogg.northwestern.edu, charu@us.ibm.com

ABSTRACT

Graph Neural Networks (GNNs), a generalization of deep neural networks on graph data have been widely used in various domains, ranging from drug discovery to recommender systems. However, GNNs on such applications are limited when there are few available samples. Meta-learning has been an important framework to address the lack of samples in machine learning, and in recent years, researchers have started to apply meta-learning to GNNs. In this work, we provide a comprehensive survey of different meta-learning approaches involving GNNs on various graph problems showing the power of using these two approaches together. We categorize the literature based on proposed architectures, shared representations, and applications. Finally, we discuss several exciting future research directions and open problems.

1. INTRODUCTION

The methods of artificial intelligence (AI) and machine learning have found tremendous success in various applications, ranging from natural language processing [17] to cancer screening [66]. Such success of AI systems can be attributed to various architectural innovations, and the ability of deep neural networks (DNN) to extract meaningful representations from Euclidean data (e.g. image, video etc.). However, in many applications, the data is graph-structured. For example, in drug discovery, the goal is to predict whether a given molecule is a potential candidate for a new drug, where the input molecules are represented by graphs. In a recommender system, the interaction between the users and the items are represented by a graph, and such non-Euclidean data is crucial in designing a better system.

The proliferation of graph structured data in various applications has led to Graph Neural Networks (GNNs) which are generalizations of DNN for graph-structured inputs. The main goal of GNNs is to learn effective representations of the graphs. Such representations map the vertices, edges, and/or graphs to a low-dimensional space, so that the structural relationships in the graph are reflected by the geometric relationships in the representations [29]. In recent years, GNNs have been applied in diverse domains, often with surprising positive results like discovery of a new antibiotic [57], accurate traffic forecasting [14], etc.

Despite of recent success of GNNs in various domains, GNN frameworks have their own shortcomings. One of the major challenges in applying GNNs, particularly for large graph-structured datasets, is the limited number of samples. Furthermore, real-world systems like recommender systems often need to handle diverse types of problems, and must adapt to a new problem with very few observations. In recent years, meta-learning has turned out be an important framework to address these shortcomings of deep learning systems. The main idea behind meta-learning is to design learning algorithms that can leverage prior learning experience to adapt to a new problem quickly, and learn a useful algorithm with few samples. Such approaches have been quite successful in diverse applications like natural language processing [41], robotics [48], and healthcare [74].

Recently, several meta learning methods to train GNNs have been proposed for various applications. There are two main challenges in applying meta-learning to graph-structured data. First, an important challenge is to determine the type of representation that is shared across different tasks. As GNNs are used for a wide range of tasks from node classification to graph classification, the learned shared representation needs to consider the type of tasks to be solved and this makes the choice and design of architecture quite important for meta-learning. Second, in a multi-task setting, we usually have few samples from each task. Thus, the support and query examples have often limited overlap in terms of similarity. For example, in node classification tasks, the nodes rarely are similar in the support and query set of a given task. On the other hand, in link prediction, the support and query edges are often located far away from each other in the graph. Therefore, a major challenge in applying meta-learning to GNNs is to model the dependencies among nodes (or edges) that are far apart (both distance-wise and similarity-wise) from each other in the graph. In this survey, we review the growing literature on meta learning with GNNs. There are several thorough individual surveys on GNNs [77, 67] and meta-learning [30], but we believe this survey is the first effort to categorize and comprehensively review the existing papers on meta learning with GNNs.

1.1 Our Contributions

Besides providing background on meta-learning and architectures based on GNNs individually, our major contributions can be summarized as follows.

- **Comprehensive review:** We provide a comprehensive review of meta learning techniques with GNNs on several graph problems. We categorize the literature based on methods, representations and applications and show various scenarios where limitations of GNNs are addressed via meta learning.
- **Future directions:** We discuss how meta learning and GNNs

can address some of the challenges in several areas: (i) combinatorial graph problems, (ii) graph mining problems, and (iii) other emerging applications such as traffic flow prediction, molecular property prediction, and network alignment.

The rest of this paper is organized as follows. Section 2 provides background on a few key graph neural network architectures. Section 3 outlines the background on meta-learning and major theoretical advances. A comprehensive categorization of the papers that use the framework of meta-learning equipped with GNNs on important graph related problems is described in Sections 4 and 5. First, Section 4 covers applications of meta-learning framework for solving some classical graph problems. The problem discussed here doesn't explicitly propose a multi-task setting, rather the meta-learning framework is applied to a fixed graph. In Section 5 we cover the literature on graph meta learning when there are multiple tasks and the graph might change with the tasks. Although various GNNs have been proposed for graph meta-learning, they can be categorized broadly based on the type of shared representation, which can be either at a local level (node/edge based) or at the global level (graph based). Table 1 provides an overview of various papers categorized by the type of shared representation and the application domains. Table 2 presents the papers described in Section 5 based on the corresponding meta-learning approaches. Section 6 covers a broad range of applications of meta-learning on GNNs and Section 7 suggests some exciting future directions.

2. GRAPH NEURAL NETWORKS

Generalizing deep learning on graphs has resulted in an exciting area of Graph Neural networks (GNNs). GNNs embed or represent nodes as points in a vector space with the help of structural and attribute information from the neighbourhood of a node and the node itself. They encode this information via non-linear transformations and aggregation functions into a final representation. The proposed architectures can be broadly categorized into two types: (i) *convolution on neighborhood*, and (ii) *location-aware*.

(i) **Convolution on neighborhood:** The primary examples of architectures that are based on *convolution on neighborhood* include GCN [36], GraphSage [28], and GAT [61]. These architectures mostly create representations of nodes through a *convolution* operation ψ over its neighborhood, i.e., the embedding, $z_{v,G} = \psi\left(N_G^k(v)\right)$ where the (k-hop) neighborhood (set of nodes) of the node v in the graph G is $N_G^k(v)$. Thus, two nodes with similar neighborhoods are likely to have similar embeddings.

(ii) **Location-aware:** The examples of GNNs that are location aware framework include PGNN [71] and GraphReach [49]. In this approach, if two nodes are located close (usually by number of hops) to each other in the graph then they are expected to have similar embeddings. If the graph has a high clustering coefficient, then one-hop neighbors of a node share many other neighbors among them as well. Therefore, if two nodes are close to each other, they have a high likelihood of having similar neighborhoods. Many real graphs have *small-world* and *scale-free* properties and have high clustering coefficients. Next, we briefly describe the key architectures of GNNs.

GCN [36]: A primary contribution in applying neural architectures on graphs has been made by [36] with the introduction of Graph Convolutional Networks (GCNs). GCNs are analogous version of convolutional neural networks (CNNs) on graphs. Inspired by the idea of representing a pixel with information from its nearby pixels (filter in CNNs), graph convolutions also apply the key idea of aggregating feature information from a node's local neighborhood. More formally, GCNs are neural network architectures that produces a d-dimensional embeddings for each node by taking as input adjacency matrix A and node features X; $\mathbf{GCN}(A, X) : \mathbb{R}^{n\times n} \times \mathbb{R}^{n\times p} \rightarrow \mathbb{R}^{n\times d}$. The idea is to aggregate feature information from a node's neighborhood (can be generalized to multiple hops) and its own features to produce the final embedding. A 2-layer (neighbourhood is 2-hops) GCN can be defined as follows:

$$\mathbf{GCN}(A, X) = \sigma(\hat{A}\sigma(\hat{A}XW^{(1)})W^{(2)})$$

where $\hat{A} = \widetilde{D}^{-\frac{1}{2}}\widetilde{A}\widetilde{D}^{-\frac{1}{2}}$ is the normalized adjacency matrix with $\widetilde{D}$ as weighted degree matrix and $\widetilde{A} = I_n + A$ with I_n being an $n \times n$ identity matrix and σ is an activation function. Moreover, $W^{(i)}$ is a weight matrix for the i-th layer to be learned during training, with $W^{(1)} \in \mathbb{R}^{p\times d'}$, $W^{(2)} \in \mathbb{R}^{d'\times d}$, and d (d') being the number of neural network nodes in the output (hidden) layer.

GraphSage [28]: Hamilton et al. [28] propose an inductive framework with an aggregation function that is able to share weight parameters ($\mathbb{W}^k$) across nodes, can be generalized to unseen nodes and scale to large datasets. To learn representation h_v^k of a node v, it iterates over all nodes which are in their K-hop neighborhood. While iterating over node v, it *aggregates* (with AGGREGATE_k) the current representations of v's neighbors ($\mathbf{h}_N^k(v)$) and *concatenate* with the current representation of v (h_v^{k-1}), which is then fed through a fully connected layer with an activation function. Intuitively, with more iterations, nodes incrementally receive information from neighbors of higher depth (i.e., distance). More specifically for k-th iteration,

$$\mathbf{h}_N^k(v) = \text{AGGREGATE}_k\left(\left\{h_u^{k-1}, \forall u \in N(v)\right\}\right)$$
$$\mathbf{h}_v^k = \sigma\left(\mathbb{W}^k \cdot \text{CONCAT}\left(\mathbf{h}_N^k(v), h_v^{k-1}\right)\right)$$

GAT [61]: Graph Attention Networks (GATs) [61] learn edge weights using attention mechanisms. GAT does not assume that the contributions of neighbouring nodes are all equal unlike in GraphSage [28]. GAT learns the relative importance/weights between two connected nodes. The graph convolutional operation (k-th iteration) is defined as follows:

$$\mathbf{h}_v^k = \sigma\left(\sum_{u\in N(v)\cup v} \alpha_{v,u}^k \mathbb{W}^k h_v^{k-1}\right)$$

where $\alpha_{v,u}$ measures the strength between the node v and its neighbour $u \in N(v)$. GAT has been shown to outperform both GCN and GraphSage in node classification task both in transductive as well as inductive settings in benchmark datasets.

PGNN [71]: Unlike in GraphSage where the representation of a node depends on its k-hop neighborhood, PGNN follows a different paradigm and aims to incorporate positional information of a node with respect to the nodes in the entire network. The key idea is that the position of a node can be captured via a low-distortion embedding by quantifying the distance between that node and a set of anchor nodes. The framework first samples multiple sets of anchor nodes. It also learns a non-linear aggregation scheme to combine the features of the nodes in each anchor set. The aggregation is normalized by the distance between the node and the anchor-set.

Other variations: There are several other variations and improvements of GNNs that are based on different mechanisms: GAT is further extended by Gated Attention Network (GAAN) [72] through a self-attention mechanism which computes an additional attention score for each attention head. Graph Autoencoders [9, 37] encode nodes/graphs into a latent vector space and further reconstruct the graph related data depending on the application from this encoding in an unsupervised fashion; Recurrent GNNs [53, 39] apply the same set of parameters recurrently over nodes to extract high-level node representations. For a comprehensive survey on GNNs, please refer to [67].

2.1 Applications

GNNs outperform traditional approaches for semi-supervised learning tasks (e.g. node classification) on graphs. The high level applications of GNNs can be categorized in three major tasks: node classification, link prediction, and graph classification. For node classification and link prediction tasks, traditionally four benchmark datasets are used: Cora, Citeseer, Pubmed, and protein-protein interaction (PPI) dataset. Shchur et al. [56] and Errica et al. [22] provide a detailed comparison of performances of the key architectures on node and graph classification tasks. GNNs are also used in the link prediction task that has applications in many domains such as friend or movie recommendation, knowledge graph completion, and metabolic network reconstruction [73].

3. BACKGROUND ON META-LEARNING

Meta-learning has turned out to be an important framework to address the problem of limited data in various machine learning applications. The main idea behind meta-learning is to design learning algorithms that can leverage prior learning experience to adapt to a new problem quickly, and learn a useful algorithm with few samples [55]. Such approaches have been quite successful in diverse applications like natural language processing [41], robotics [48], and healthcare [74].

3.1 Framework

In standard supervised learning, we are given a training dataset $\mathcal{D} = \{\boldsymbol{x}_i, y_i\}_{i=1}^{n}$, a loss function ℓ, and we aim to find a predictive model of the form $\hat{y} = f_\theta(\boldsymbol{x})$.

$$\theta^* = \text{argmin}_\theta \mathcal{L}(\mathcal{D}, \theta) = \text{argmin}_\theta \sum_{i=1}^{n} \ell(f_\theta(\boldsymbol{x}_i), y_i)$$

In meta-learning, we are given samples from a number of different tasks and the goal is to learn an algorithm that generalizes across tasks. In particular, the tasks are drawn from a distribution $p(\mathcal{T})$, and the meta-objective is to find a common parameter that works across the distribution of tasks.

$$\omega^* = \text{argmin}_\omega \sum_{\substack{\mathcal{T}_i \sim p(\mathcal{T}) \\ \mathcal{D}_i \sim \mathcal{T}_i}} \mathcal{L}_i(\mathcal{D}_i, \omega) \qquad (1)$$

In the meta-test phase, we are given a target task (say task 0) and we use the meta-knowledge ω^* to obtain the best parameter for the target with few samples.

$$\theta_0^* = \text{argmin}_\theta \mathcal{L}_0(\mathcal{D}_0, \theta | \omega^*)$$

3.2 Training

Many popular meta-learning algorithms are based on gradient descent on the meta-parameter ω [23, 52]. In order to understand how to perform gradient descent with respect to ω, it is insightful to frame Equation (1) as a bi-level optimization problem.

$$\omega^* = \text{argmin}_\omega \sum_{\substack{\mathcal{T}_i \sim p(\mathcal{T}) \\ \mathcal{D}_i \sim \mathcal{T}_i}} \mathcal{L}(\mathcal{D}_i, \theta_i^*(\omega), \omega)$$
$$\text{s.t. } \theta_i^*(\omega) = \text{argmin}_\theta \mathcal{L}_i(\theta, \omega, \mathcal{D}_i) \ \forall i$$

If we have a model for the inner-optimization method, then a gradient of the objective with respect to ω can be computed by using the chain rule e.g.

$$\nabla_\omega \mathcal{L}(\mathcal{D}_i, \theta_i^*(\omega), \omega) = \nabla_{\theta_i^*(\omega)} \mathcal{L}(\mathcal{D}_i, \theta_i^*(\omega), \omega) \frac{d\theta_i^*(\omega)}{d\omega}$$

However, often the inner objective function is non-convex, and hard to solve. So model agnostic meta learning (MAML), introduced by Finn et al. [23] suggests to first take a gradient step for each task i as follows:

$$\theta_i' = \theta_i(\omega) - \alpha \nabla_\theta \mathcal{L}_i(\theta_i(\omega), \omega, \mathcal{D}_i)$$

Then MAML replaces $\theta_i^*(\omega)$ in the outer objective, i.e.,

$$\omega = \omega - \beta \nabla_\omega \sum_i \mathcal{L}(\mathcal{D}_i, \theta_i', \omega)^{1}$$

We now instantiate the MAML algorithm for the task of classifying nodes of a graph. Recall the GCN framework from Section 2. Here the t-th task is classification of nodes of a graph G_t with adjacency matrix A_t and node-feature matrix X_t. Then a standard two-layer GCN for node classification problem is given as follows:

$$f(X_t, A_t, W_t) = \text{softmax}\left(\hat{A}_t \text{ReLU}\left(\hat{A}_t X_t W_t^{(1)}\right) W_t^{(2)}\right) \qquad (2)$$

Given labels of the nodes Y_t, such a network is often trained with the cross-entropy loss:

$$\mathcal{L}_t(X_t, A_t, W_t) = -\sum_\ell \sum_f Y_{\ell f} \ln f(X_t, A_t, W_t)_{\ell f}$$

Usually, the parameters W_t are trained by stochastic gradient descent. Here, we wish to identify a meta parameter vector $W_\star$, which is close to the parameters of different tasks (i.e. $\|W_t - W_\star\|_F \leq \delta$ for some $\delta > 0$). The benefit of learning such meta-parameters $W_\star$ is that, on a new task s, we can initialize task-parameter W_s as $W_\star$ and the new task would require very few samples to train. Algorithm 1 describes the MAML algorithm instantiated for the case of node classification with GCN based representation.

ALGORITHM 1: Model Agnostic Meta-Learning for GCN

```
Input: Step sizes $\alpha$ and $\beta$.
Initialize $W_\star$.
do
    Sample a batch of $T$ tasks $\{G_i\} \sim p(\cdot)$.
    Sample a batch of $T$ datasets $\{\mathcal{D}_i = (A_i, X_i, Y_i)\}$ where
      $\mathcal{D}_i \sim G_i$.
    for each task t in T do
        Update $W_t = W_\star - \alpha \nabla_W \mathcal{L}_t(X_t, A_t, W)\Big|_{W=W_\star}$.
    Update $W_\star = W_\star - \beta \nabla_W \sum_t \mathcal{L}(X_t, A_t, W)\Big|_{W=W_t}$
while Not Convergence
return Meta-parameter $W_\star$.
```

3.3 Representation Learning

Another perspective of meta-learning, which will be particularly important for the context of graph neural networks, is learning a shared representation across different tasks. Here we assume that, given an input x, the training data from the t-th task is generated as $y_t = f_t \circ h(x) + \eta_t$, where η_t is some iid noise. Effectively, the function h maps input x to a shared representation and then a task-specific function f_t is applied to generate the task-specific representation.

[1]We write $\theta_i(\omega)$ to denote the meta-parameter ω adapted to task i.

During the meta-training phase, we attempt to learn the shared function h. Suppose we are given T datasets $\mathcal{D}_t = \{(x_{ti}, y_{ti}\}_{i=1}^{n_t}$ for $t = 1, \ldots, T$. Then we solve the following optimization problem to recover h.

$$\underset{h,\{f_t\}_{t=1}^{T}}{\operatorname{argmin}} \sum_{t=1}^{T} \sum_{i=1}^{n_t} \mathcal{L}_t\left(y_{ti}, f_t(h(x_{ti}))\right) + \mathcal{R}(h) + \sum_{t} \mathcal{R}(f_t) \quad (3)$$

Here $\mathcal{R}(\cdot)$ is some regularization function, and let $\hat{h}, \left\{\hat{f}_t\right\}_{t=1}^{T}$ be its solution. In general, the optimization problem defined in Equation (3) is hard to solve unless we make specific assumption about the types of functions. For example, even if we assume f_t is same across the tasks and in fact an identity function, the problem defined in Equation (3) can involve learning a general neural network based shared representation h. For the special case of linear models, this problem can be solved efficiently (e.g. by using matrix regression [59]). In this survey, we focus on gradient based methods for learning the shared representation h in equation (3), which has been quite successful in practice. In the meta-test phase, we are given samples from a new task s e.g. $\{(x_{si}, y_{si})\}_{i=1}^{n_s}$. We substitute $\hat{h}$, the estimate of the common representation function h, and learn the new task-specific function f_s.

$$\hat{f}_s \leftarrow \underset{f_s}{\operatorname{argmin}} \sum_{i=1}^{n_s} \mathcal{L}_s\left(y_{si}, f_s(\hat{h}(x_{si}))\right) + \mathcal{R}(f_s)$$

We now instantiate this framework for the task of classifying nodes of a graph. As before, we use two-layer GCN where the model is defined in Equation (2). However, we now assume that the first layer is shared across different tasks and only the second layer is trained for a new task. In particular, we assume $W_t = [W^\star; W_t^{(2)}]$. Although, the optimization problem in Equation (3) is NP-hard to solve with this particular type of representation, we can write down an algorithm to solve for the meta-parameter $W^\star$ using gradient descent. Algorithm 2 describes this algorithm and returns the meta-parameter $W^\star$.

ALGORITHM 2: Shared Representation Learning for GCN

```
Input: Step sizes $\alpha$ and $\beta$, datasets $\mathcal{D}_t = \{(x_{ti}, y_{ti}\}_{i=1}^{n_t}$ for
    $t = 1, \ldots, T$.
Initialize $W^\star$.
Initialize $W_t^{(2)}$ for $t = 1, \ldots, T$.
Set $W_t = [W^\star, W_t^{(2)}]$.
do
    for each task t in $[T]$ do
        Update
        $W_t^{(2)} = W_t^{(2)} - \alpha \nabla_W \mathcal{L}_t\left(\mathcal{D}_t, [W^\star; W]\right)\Big|_{W=W_t^{(2)}}$.
    Update $W^\star = W^\star - \beta \nabla_W \sum_t \mathcal{L}\left(\mathcal{D}_t, [W; W_t^{(2)}]\right)\Big|_{W=W^\star}$
while Not Convergence
return Meta-parameter $W^\star$.
```

3.4 Theory

Despite immense success, we are yet to fully understand the theoretical foundations of meta-learning algorithms. Baxter [5] first prove generalization bound for multitask learning problem, by considering a model where tasks with shared representation are sampled from a generative model. Pontil et al. [51], and Maurer et al. [46] develop general uniform-convergence based framework to analyze multitask representation learning. However, they assume oracle access to a global empirical risk minimizer. Recently, there have been promising attempts to understand meta learning from representation learning. The main idea is that the tasks share a common shared representation and a task-specific representation [60, 59, 21]. If the shared representation is learned from the training tasks, then the task-specific representation for the new task can be learned with only a few samples. Finally, there have been interesting recent work trying to understand gradient-based meta-learning. [24, 4, 35, 16] analyze gradient based meta-learning in the framework of online convex optimization (OCO). They assume that the parameters of the tasks are close to a shared parameter to bound regret in the OCO framework.

4. META-LEARNING ON FIXED GRAPHS

In this section, we review applications of meta-learning for solving some classical problems on graphs. Here we consider the setting when the underlying graph is fixed and the node/edge features do not change with different tasks. In fact, we are not in a multitask framework where there are a number of tasks and few samples are available from each task. Rather, the framework of meta-learning is applied to various graph problems by creating multiple tasks either considering the nodes or the edges.

4.1 Node Embedding

The goal of node embedding is to learn representations for the nodes in the graph so that any downstream application can directly work with these representations, without considering the original graph. This problem is often challenging in practice because the degree distributions of most graphs follow a power law distribution and there are many nodes with very few connections. Liu et al. [43] address this issue by applying meta-learning to the problem of node embedding of graphs. They set up a regression problem with a common prior to learn the node embeddings. Since the base representations of high-degree nodes are accurate, they are used as meta training set to learn the common prior. The low degree nodes have only a few neighbors (samples), the regression problem for learning their representations is formulated as a meta-testing problem, and the common prior is adapted with a small number of samples for learning the embeddings of such nodes.

4.2 Node Classification

The node classification task aims to infer the missing labels of nodes of a given partially labeled graph. This problem often appears in diverse contexts such as document categorization and protein classification [58, 6], and has received significant attention in recent years. However, often many classes are novel i.e., they have a small number of labeled nodes. This makes meta-learning or few-shot learning particularly suitable for this problem.

Zhou et al. [76] have applied a meta-learning framework for the node classification problem on graphs by learning a transferable representation using data from classes that have many labeled examples. Then, during the meta-test phase, this shared representation is used to make predictions for novel classes with few labeled samples. Ding et al. [19] improve upon the previous method by considering a prototype representation of each class and meta-learning the prototype representation as an average of weighted representations of each class. Lan et al. [38] address the same problem via meta-learning but in a different setting where the nodes do not have attributes. Their method only uses the graph structure to obtain latent representation of nodes for the task. Subsequently, Liu et al. [42] point out that it is important to also learn the dependencies among the nodes in a task, and propose to use nodes with high centrality scores (or hub nodes) to update the representations learned by a GNN. This is done by selecting a small set of hub

	Graph applications		
Representation	Node classification	Link Prediction	Graph Classification
Node/Edge Level	Meta-GNN [76], GPN [19], RALE [42], AMM-GNN [62]	MetaR [12], GEN [1]	
	SAME [8], SELAR [32] GFL [70], Meta-GDN [20]	SAME [8], SELAR [32]	SAME [8]
Graph Level	MI-GNN [64]	Meta-graph [7]	AS-MAML [44], Spectral [11]

Table 1: Organization of the papers on Meta-learning and GNNs based on applications and underlying graph-related representations. The abbreviations of the frameworks (methods) are as follows. GPN: Graph Prototypical Networks, MetaR: Meta Relational learning, GEN: Graph Extrapolation Networks, RALE: Relative and Absolute Location Embedding, AMM-GNN: Attribute Matching Meta-learning Graph Neural Networks, SAME: Single-task Adaptation for Multi-task Embeddings, SELAR: SELf-supervised Auxiliary LeaRning, GFL: Graph Few-shot Learning, GDN: Graph Deviation Networks, MI-GNN: Meta-Inductive framework for Graph Neural Network, AS-MAML: Adaptive Step Model Agnostic Meta Learning.

nodes and for each node v, considering all the paths to the node v from the set of hub nodes. It helps to encode the absolute location in the graph. Parallel to these developments, Yao et al. [70] consider a metric-learning based approach where the label of a node is predicted to be the nearest class-prototype in a transferable metric space. They first learn a class-specific representation using a GNN, and then learn a task-specific representation using hierarchical graph representations.

Finally, the few-shot node classification task has also been used in the presence of noisy or inaccurate labels in the support sets of different tasks. Ding et al. [18] propose a method (Graph Hallucination Network) that creates a set by taking a specified number of samples from a class. Then the method learns to produce a confidence score on the accuracy of the label of each node in the set. By using these weights/scores, the final cleaner (i.e., less noisy) node representations are generated. The rest of the algorithm follows the standard MAML framework.

4.3 Link Prediction

The objective of the link prediction problem is to identify pairs of nodes that will either form a link or not. Meta-learning has been shown to be useful for learning new relationship via edges/links especially in multi-relational graphs.

In multi-relational graphs, an edge is represented by a triple of two end points and a relation. Such graphs appear in many important domains such as drug-drug interaction prediction. The goal of link prediction in multi-relation graphs is to predict new triples given one end point of a relation r with observing a few triples about r. This problem is challenging as only few associative triples are usually available. Chen et al. [12] use meta-learning to solve the link prediction problem in two steps. First, they design a Relation-Meta Learner which learns shared structure across a number of relations. Such a meta-learner generates relation meta from heads' and tails' embeddings in the support set. Second, they use an embedding learner that calculates the truth values of triples in support set via end points' embeddings and relation meta.

Multi-relational graphs are even more difficult to manage with their dynamic nature (addition of new nodes) over time and the learning is even more difficult when these newly evolved nodes have only few links among them. Baek et al. [1] introduce a few-shot out-of-graph link prediction technique, where they predict the links between the seen and unseen nodes as well as between the unseen nodes. The main idea is to randomly split the entities in a given graph into the meta-training set for simulated unseen entities, and the meta-test set for real unseen entities.

Finally, Hwang et al. [32] show the effectiveness of graph neural networks on downstream tasks such as node classification and link prediction via a self-supervised auxiliary learning framework combined with meta-learning. The auxiliary task such as meta-path prediction does not need labels and thus the method becomes self-supervised. In the meta learning framework, various auxiliary tasks are used to improve generalization performance of the underlying primary task (e.g., link prediction). The proposed method effectively combines the auxiliary tasks and automatically balances them to improve performance on the primary task. The method is also flexible to work with any graph neural network architecture without additional data.

5. META-LEARNING ON GRAPH NEURAL NETWORKS

We now discuss the growing and exciting literature on graph meta learning where there are multiple tasks and the underlying graph can change across the tasks. The changes in graphs occur when either the node/edge features change, or the underlying network structure changes with the tasks. In the context of meta-learning, several architectures have been proposed in recent years. However, a common thread underlying all of them is a shared representation of the graph, either at a local node/edge level, or at a global graph level. Based on the type of shared representation, we categorize the existing works into two groups. Most of the existing literature adopt the MAML algorithm [23] to train the proposed GNNs. The outer loop of MAML updates the shared parameter, whereas the inner loop updates the task-specific parameter for the current task. Table 2 lists the shared and the task-specific parameters for all the papers in this section.

5.1 Node/Edge Level Shared Representation

First, we consider the setting where the shared representation is local i.e. node or edge based. Huang et al. [31] consider the node classification problem where the input graphs as well as the labels can be different across tasks. They learn a representation for each node u in two steps. First, the method extracts a subgraph S_u corresponding to the set of nodes $\{v : d(u, v) \leq h\}$ where $d(u, v)$ is the distance of the shortest path between nodes u and v. Then it feeds the subgraph S_u through a GCN to learn a representation for node u. The theoretical motivation behind considering the graph S_u is that the influence of a node v on u decreases exponentially as the shortest-path distance between them increases. Once the nodes are encoded, one can learn any function f_θ that maps the encodings to class labels. Huang et al. [31] use MAML to learn this function with very few samples on a new task, enjoying the benefits of node-

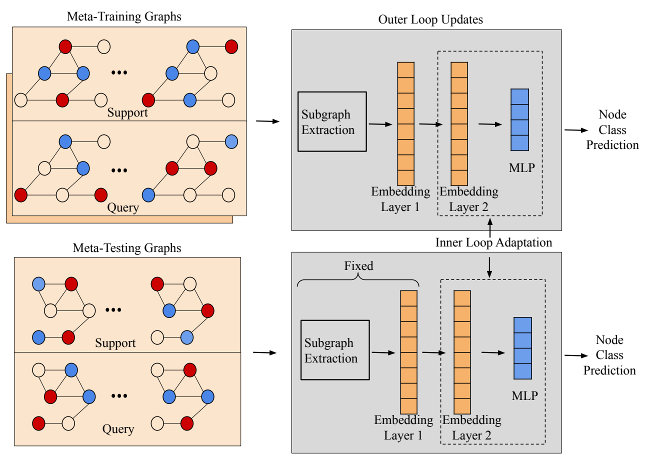

Figure 1: A prototype of the meta learning framework with GNNs for solving node classification problem. This is based on the architectures proposed by [31] and [62]. Following [31], the neighborhoods of each node are used for node embedding. Embedding layer 1 is trained in the outer loop of MAML, whereas the other layers are adapted for particular tasks.

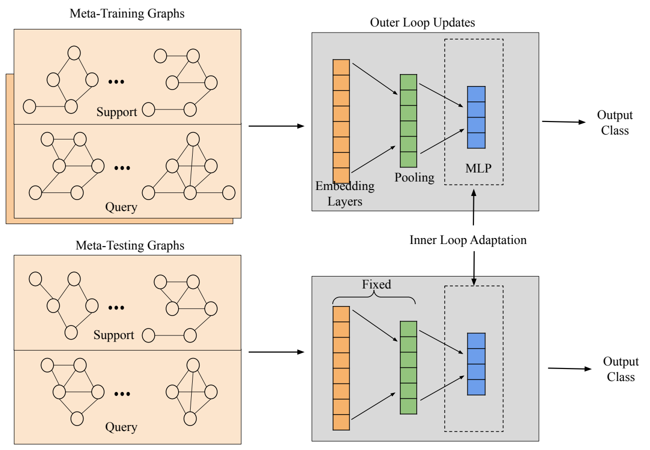

Figure 2: A prototype of the meta learning framework with GNNs for solving graph classification problem. This is based on the architectures proposed by [44], and [8]. The embedding and pooling layers learn global representation of the input graph, and are trained in the outer loop of MAML. The final multi-layer perceptron (MLP) is used for the classification task and is adapted to the particular task at meta-test.

Papers	Meta-learning parameters	
	Inner Loop (Task-Specific)	Outer Loop (Shared)
[31]	Node embeddings	Classification
[62]	Node embeddings	Feature matrix
[11]	Graph feature, Super-class	graph label/ actual class
[44]	Graph feature, Graph embedding	graph embedding/ Classification
[8]	Node Embedding	Output Layer
[7]	VGAE Initialization	Graph Signature (GCN + MLP)
[43]	High-degree node embedding	node specific embedding

Table 2: Organization of the papers in Section 6 based on the corresponding meta-learning approaches.

level shared representations in node classification.

Wang et al. [62] also consider the few-shot node classification problem for a setting where the network structure is fixed, but the features of the nodes change with tasks. In particular, given a base graph with node feature matrix $X \in \mathbb{R}^{n \times d}$, the proposed model learns a new feature matrix $X_t = X \odot \alpha_t(\phi) + \beta_t(\phi)$ for the t-th task, and then use a GNN $f_\theta(X_t)$ to learn the node representations for the t-th task. During training, the outer loop updates the ϕ parameters, whereas the inner loop of MAML only updates the θ-parameter. This enables quick adaptation to the new task.

Wen et al. [64] study the problem of node classification in an inductive setting, where the graph instances in testing and training do not overlap. Their method involves computing a task prior given a graph (i.e., its representation) using multi-layer perceptron (MLP). These representations are useful for the graph-level adaptation. They used the traditional MAML paradigm in their approach for the task-level adaptation.

5.2 Graph Level Shared Representation

In this subsection, we discuss the setting when the shared representation is global i.e. graph-level. A canonical application of this representation is the *graph classification* problem, where the goal is to classify a given graph to one of many possible classes. This problem appears in many applications, ranging from bioinformatics to social network analysis [69]. However, in many settings, the number of samples/graphs available for a particular task is few and the graph classification task often requires a large number of samples for high quality prediction. These challenges can be addressed via meta-learning. The existing papers on using meta-learning for graph classification usually learn an underlying shared representation and adapt the representation for a new task.

Chauhan et al. [11] propose the few-shot graph classification task based on graph spectral measures. In particular, they train a feature-extractor $F_\theta(\cdot)$ to extract features from the graphs in meta-training. For classification, they first use a unit C^{sup} to first predict the super-class probability of a graph which is a clustering of abundant base class labels. Then they use C^{att}, an attention network to predict the actual class label. During the meta-test phase, the weights of the networks $F_\theta(\cdot)$ and C^{sup} are fixed, and the network C^{att} is retrained on the new test classes. As the feature extractor F_θ is the common shared structure, and is not retrained on the test tasks, this approach requires few samples from new classes.

Although Chauhan et al. [11] propose a novel meta-learning architecture for graph classification, there are several limitations. First, the architecture assumes significant overlap between the super-class structure of the test and the training set. Second, the fixed feature extractor cannot be updated for the new tasks. Ma et al. [44] design a better meta-learning technique by allowing the feature extractor to adapt efficiently for new tasks. They apply two networks – embedding layers (θ_e), followed by classification layers (θ_c) to classify a given graph. However, for a new task, both θ_e and θ_c are updated. In particular, the authors use MAML [23] to update the parameters and use a reinforcement learning based controller to determine how the inner loop is run i.e., what is the optimal adaptation step for a new task. The parameters of the controller is updated using the graph's embedding quality and the meta-learner's training state.

Jiang et al. [33] solve the problem of few-shot graph classification via a paradigm in meta learning called metric learning approach [63] that is different from MAML. In the training phase, the idea is to get a mean representations of the instances in each class in the support set. The prediction for query is based on the nearest neighbour. Here the graph representations were obtained by the Graph Isomorphism Network (GIN) model. To capture the global structure of the graph, they used different weights for different GIN layers in the final aggregation scheme. To encode the crucial local structures that might have importance in deciding the graph label, the paper embeds subgraphs and includes their representations with different attention weights.

Finally, Buffelli et al. [8] attempt to develop a framework that can adapt to three different tasks – graph classification, node classification, and link prediction. Like [11, 44] they use two different layers; one generates node embeddings and converts the graph to a representation, and another is a multi-head output layer for the three types of tasks. The node embedding layer is trained during the initialization phase of MAML and the multi-head output layer is updated in the inner loop of MAML based on the type of task.

Bose et al. [7] consider the few-shot link prediction problem, where the goal is to predict labels of links/edges that contain only a small fraction of their true labels. They assume that the graphs are generated from a common distribution $p(\cdot)$ and learn a meta link prediction model that can be quickly adapted to a new graph $G \sim p(\cdot)$. In particular, the authors use Variational Graph Autoencoder (VGAE) [37] to model the base link prediction model. There are two sets of parameters – global initialization parameters for the VGAE, and local graph signature $s_G = \psi(G)$ which is obtained by passing the graph G through GCN and then using a k-layer MLP. The training is done using MAML and only the graph signature is updated for the test graph.

6. OTHER APPLICATIONS

We have discussed applications of meta-learning equipped with GNNs on node classification, link prediction, and graph classification. In fact, this framework is quite general and can be applied to many other relevant important problems.

Anomaly Detection: The problem of anomaly detection often suffers from scarcity of labels, as obtaining labels for anomalies is usually labor intensive. Ding et al. [20] study anomaly detection when there are scarcity of labels, and different tasks involve different graphs. The proposed method used traditional architectures of GNNs to embed nodes and predict the anomaly score by adding another layer after the embedding is obtained. Finally it exploits the traditional MAML framework to deploy the meta-learner. The inner loop optimizes the parameters for a specific task, i.e., graph. The outer-loop optimizes the generic parameter for all graphs.

Network Alignment (NA): NA aims to map or link entities from different networks and relevant in many application domains such as cross-domain recommendation and advertising. Zhou et al. [75] address this alignment problem via meta-learning. If two different networks share some common nodes or anchors, then these networks are partially aligned networks. A virtual link between two anchors is called anchor link. In NA, given a set of networks and some known anchor nodes (or links), the goal is to identify all the other (unknown) potential anchor nodes (or links). The main idea in [75] is to frame this problem as one shot classification problem and use the meta-metric learning from known anchor nodes to obtain latent priors for linking unknown anchor nodes.

Traffic Prediction: Recently, the traffic prediction problem [50] has been addressed via meta-learning. In traffic prediction, the main challenges are modeling complex spatio-temporal correlations of traffic and capturing the diversity of such correlations varying locations. Pan et al. [50] address these challenges with a meta-learning based model. Their method predicts traffic in all locations at the same time. The proposed framework consists of a sequence-to-sequence architecture that uses an an encoder to learn traffic history and a decoder to make predictions. For the encoder and decoder components a combination of graph attention networks and recurrent neural networks is used to model diverse spatial and temporal correlations respectively.

7. FUTURE DIRECTIONS

The application of meta-learning using GNNs for graph specific applications is a growing and exciting area of research. In this section, we suggest several future directions for research.

7.1 Combinatorial Optimization Problems on Graphs

Combinatorial optimization problems appearing in graphs have applications in many domains such as viral marketing in social networks [34], health-care [65], and infrastructure development [47], and several architectures based on GNNs have been proposed for solving them [15, 40, 26, 45]. These optimization problems are often NP-hard, and polynomial-time algorithms, with or without approximation guarantees, are often desirable and used in practice. However, some techniques [40, 45] based on GNNs need to generate candidate solution nodes/edges before generating the actual solution set. Note that, labels in the form of importance of each node in a solution set of these problems are often difficult to get. Meta-learning can be used when there are scarcity of labels. Furthermore, these combinatorial problems often share similar structures. For instance, the influence maximization problem [34] have similarity with the Max Cover problem. However, even performing a greedy iterative algorithm to generate solutions/labels for influence maximization problem is computationally expensive. The idea of using meta-learning in solving a harder combinatorial problem (unseen task) with a fewer node labels will be to learn on the easier problems (seen tasks) where labels can be generated at a lower cost. Solving combinatorial optimization problems on graphs via neural approaches has recently gained a lot of attention and we refer the readers to [10] for further reading.

7.2 Graph Mining Problems

There has been recent attempt to solve classical graph mining problems with GNNs. For instance, a popular problem is to learn similarity between two graphs, i.e., to find graph edit distance (similarity) between two graphs [2]. When the notion of similarity changes and there are not enough data to learn via a standard supervised learning method, can meta-learning be helpful? Another popular graph mining problem is detecting the Maximum Common Subgraph (MCS) between two input graphs with applications in biomedical analysis and malware detection. In drug design, common substructures in compounds can reduce the number of human-conducted experiments. However, MCS computation is NP-hard, and state-of-the-art exact MCS solvers are not scalable to large graphs. Designing learning based models [3] for the MCS problem while utilizing as few labeled MCS instances as possible remains to be a challenging task and meta-learning could be helpful in mitigating this challenge.

7.3 Theory

We point out several important theoretical questions in the context of meta learning with GNNs. The most natural question is understanding the benefits of transfer learning in GNNs. Garg et al. [25] and Scarselli et al. [54] have recently established generalization bounds for GNNs. On the other hand, in the context of meta-learning, Tripuraneni et al. [60] consider functions of the form $f_j \cdot h$, where $f_j \in \mathcal{F}$ is the task-specific function and h is the shared function. Then the number of samples required in the meta-test phase grows as $C(\mathcal{F})$, which can be significantly lower than learning $f_j \cdot h$ from scratch. It would be interesting to see if one can prove similar speedup results for GNNs by generalizing the results of [25] and [54]. Another interesting question is determining the right level of shared representation and figuring out the expressiveness of such structures. The seminal work of Xu et al. [68] proves that variants of GNNs such as GCN and GraphSAGE are no more discriminative than the Weisfeiler-Lehman (WL) test. Since GNNs for meta-learning further limit the type of architecture used, an interesting question is whether it comes with any additional cost on expressiveness. Finally, the methods discussed in Section 5 differ in one crucial way – whether they fine-tune and update the shared meta-parameter on a new task or whether they keep the shared meta-parameter fixed. Recently, Chua et al. [13] show that fine-tuning the meta-parameter could be beneficial in some situations, particularly when the number of samples on the new task is large. In the context of meta learning on GNNs, it would be interesting to understand when such fine-tuning helps to improve the performance on a new task.

7.4 Applications

We have already discussed a few applications of meta-learning with frameworks of GNNs in Section 6. This generic framework is quite relevant for many important problems in the field.

Network alignment: A potential problem where meta-learning could be helpful is network alignment (NA) [75]. In NA, the main goal is to map or link entities from different networks and the existing approaches is quite difficult to scale. An interesting direction of research would consider meta-learning to overcome this scalability challenge.

Molecular property prediction: GNNs have been also used in predicting molecular properties. However, one of the main challenges is that molecules are heterogeneous structure where each atom has connection with different neighboring atoms via different types of bonds. Secondly, often a limited amount of data on labeled molecular property are available; and thus, to predict new molecular properties, meta-learning techniques [27] can be relevant and effective.

Dynamic graphs: In many applications, graphs arise with their dynamic nature, i.e., nodes and edges along with their attributes can change (addition or deletion) over time. Most of the papers discussed above use frameworks that are built on meta-learning and GNNs for static graphs. An interesting direction would be to extend this framework for dynamic graphs. Dynamic nature brings new challenges such as difficulty in obtaining labels for newly added nodes or edges. For instance, in knowledge graphs, newly added edges introduces new relationships. The other challenge is efficiency as managing and making predictions on evolving networks are difficult tasks as its own. Meta-learning would be useful to address these challenges.

8. CONCLUSION

In this survey, we have performed a comprehensive review of the works that are combination of graph neural networks (GNNs) and meta-learning. Besides outlining backgrounds on GNNs and meta-learning, we have organized the past research in an organized manner in multiple categories. We have also provided a thorough review, summary of methods, and applications in these categories. Furthermore, we have described several future research directions where meta learning with GNNs can be useful. The application of meta-learning to GNNs is a growing and exciting field and we believe many graph problems will benefit immensely from the combination of the two approaches.

References

[1] Jinheon Baek, Dong Bok Lee, and Sung Ju Hwang. "Learning to extrapolate knowledge: Transductive few-shot out-of-graph link prediction". In: *NeurIPS* (2020).

[2] Yunsheng Bai, Hao Ding, Song Bian, Ting Chen, Yizhou Sun, and Wei Wang. "SimGNN: A Neural Network Approach to Fast Graph Similarity Computation". In: *WSDM*. 2019.

[3] Yunsheng Bai, Derek Xu, Alex Wang, Ken Gu, Xueqing Wu, Agustin Marinovic, Christopher Ro, Yizhou Sun, and Wei Wang. "Fast detection of maximum common subgraph via deep q-learning". In: *arXiv preprint arXiv:2002.03129* (2020).

[4] Maria-Florina Balcan, Mikhail Khodak, and Ameet Talwalkar. "Provable guarantees for gradient-based meta-learning". In: *ICML*. 2019, pp. 424–433.

[5] Jonathan Baxter. "A model of inductive bias learning". In: *JAIR* 12 (2000), pp. 149–198.

[6] Karsten M Borgwardt, Cheng Soon Ong, Stefan Schönauer, SVN Vishwanathan, Alex J Smola, and Hans-Peter Kriegel. "Protein function prediction via graph kernels". In: *Bioinformatics* 21 (2005), pp. i47–i56.

[7] Avishek Joey Bose, Ankit Jain, Piero Molino, and William L Hamilton. "Meta-graph: Few shot link prediction via meta learning". In: *arXiv preprint arXiv:1912.09867* (2019).

[8] Davide Buffelli and Fabio Vandin. "A Meta-Learning Approach for Graph Representation Learning in Multi-Task Settings". In: *arXiv preprint arXiv:2012.06755* (2020).

[9] Shaosheng Cao, Wei Lu, and Qiongkai Xu. "Deep neural networks for learning graph representations". In: *AAAI* 30.1 (2016).

[10] Quentin Cappart, Didier Chételat, Elias Khalil, Andrea Lodi, Christopher Morris, and Petar Veličković. "Combinatorial optimization and reasoning with graph neural networks". In: *arXiv preprint arXiv:2102.09544* (2021).

[11] Jatin Chauhan, Deepak Nathani, and Manohar Kaul. "Few-Shot Learning on Graphs via Super-Classes based on Graph Spectral Measures". In: *ICLR* (2020).

[12] Mingyang Chen, Wen Zhang, Wei Zhang, Qiang Chen, and Huajun Chen. "Meta Relational Learning for Few-Shot Link Prediction in Knowledge Graphs". In: *EMNLP-IJCNLP*. 2019, pp. 4208–4217.

[13] Kurtland Chua, Qi Lei, and Jason D Lee. "How fine-tuning allows for effective meta-learning". In: *arXiv:2105.02221* (2021).

[14] Zhiyong Cui, Kristian Henrickson, Ruimin Ke, and Yinhai Wang. "Traffic graph convolutional recurrent neural network: A deep learning framework for network-scale traffic learning and forecasting". In: *IEEE TITS* (2019), pp. 4883–4894.

[15] Hanjun Dai, Elias B Khalil, Yuyu Zhang, Bistra Dilkina, and Le Song. “Learning combinatorial optimization algorithms over graphs”. In: *NeurIPS* (2017).

[16] Giulia Denevi, Carlo Ciliberto, Riccardo Grazzi, and Massimiliano Pontil. “Learning-to-learn stochastic gradient descent with biased regularization”. In: *ICML*. 2019, pp. 1566–1575.

[17] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. “Bert: Pre-training of deep bidirectional transformers for language understanding”. In: *NAACL-HLT* (2019).

[18] Kaize Ding, Jianling Wang, Jundong Li, James Caverlee, and Huan Liu. “Weakly-supervised Graph Meta-learning for Few-shot Node Classification”. In: *arXiv:2106.06873* (2021).

[19] Kaize Ding, Jianling Wang, Jundong Li, Kai Shu, Chenghao Liu, and Huan Liu. “Graph prototypical networks for few-shot learning on attributed networks”. In: *CIKM*. 2020, pp. 295–304.

[20] Kaize Ding, Qinghai Zhou, Hanghang Tong, and Huan Liu. “Few-Shot Network Anomaly Detection via Cross-Network Meta-Learning”. In: *Proceedings of the Web Conference 2021*. 2021, 2448–2456.

[21] Simon S Du, Wei Hu, Sham M Kakade, Jason D Lee, and Qi Lei. “Few-shot learning via learning the representation, provably”. In: *arXiv preprint arXiv:2002.09434* (2020).

[22] Federico Errica, Marco Podda, Davide Bacciu, and Alessio Micheli. “A fair comparison of graph neural networks for graph classification”. In: *arXiv preprint arXiv:1912.09893* (2019).

[23] Chelsea Finn, Pieter Abbeel, and Sergey Levine. “Model-agnostic meta-learning for fast adaptation of deep networks”. In: *ICML*. 2017, pp. 1126–1135.

[24] Chelsea Finn, Aravind Rajeswaran, Sham Kakade, and Sergey Levine. “Online meta-learning”. In: *ICML*. 2019, pp. 1920–1930.

[25] Vikas Garg, Stefanie Jegelka, and Tommi Jaakkola. “Generalization and representational limits of graph neural networks”. In: *ICML*. PMLR. 2020, pp. 3419–3430.

[26] Maxime Gasse, Didier Chételat, Nicola Ferroni, Laurent Charlin, and Andrea Lodi. “Exact combinatorial optimization with graph convolutional neural networks”. In: *NeurIPS* (2019).

[27] Zhichun Guo, Chuxu Zhang, Wenhao Yu, John Herr, Olaf Wiest, Meng Jiang, and Nitesh V Chawla. “Few-Shot Graph Learning for Molecular Property Prediction”. In: *The Web Conference* (2021).

[28] Will Hamilton, Zhitao Ying, and Jure Leskovec. “Inductive representation learning on large graphs”. In: *NeurIPS*. 2017, pp. 1024–1034.

[29] William L Hamilton, Rex Ying, and Jure Leskovec. “Representation learning on graphs: Methods and applications”. In: *arXiv preprint arXiv:1709.05584* (2017).

[30] Timothy Hospedales, Antreas Antoniou, Paul Micaelli, and Amos Storkey. “Meta-learning in neural networks: A survey”. In: *arXiv preprint arXiv:2004.05439* (2020).

[31] Kexin Huang and Marinka Zitnik. “Graph meta learning via local subgraphs”. In: *NeurIPS* (2020).

[32] Dasol Hwang, Jinyoung Park, Sunyoung Kwon, Kyung-Min Kim, Jung-Woo Ha, and Hyunwoo J Kim. “Self-supervised Auxiliary Learning for Graph Neural Networks via Meta-Learning”. In: *arXiv preprint arXiv:2103.00771* (2021).

[33] Shunyu Jiang, Fuli Feng, Weijian Chen, Xiang Li, and Xiangnan He. “Structure-Enhanced Meta-Learning For Few-Shot Graph Classification”. In: *arXiv preprint arXiv:2103.03547* (2021).

[34] David Kempe, Jon Kleinberg, and Éva Tardos. “Maximizing the spread of influence through a social network”. In: *KDD*. 2003.

[35] M Khodak, M Balcan, and A Talwalkar. “Adaptive Gradient-Based Meta-Learning Methods”. In: *Neural Information Processing Systems*. 2019.

[36] Thomas N Kipf and Max Welling. “Semi-supervised classification with graph convolutional networks”. In: *ICLR* (2017).

[37] Thomas N Kipf and Max Welling. “Variational graph auto-encoders”. In: *arXiv preprint arXiv:1611.07308* (2016).

[38] Lin Lan, Pinghui Wang, Xuefeng Du, Kaikai Song, Jing Tao, and Xiaohong Guan. “Node classification on graphs with few-shot novel labels via meta transformed network embedding”. In: *NeurIPS* (2020).

[39] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. “Gated graph sequence neural networks”. In: *ICLR* (2016).

[40] Zhuwen Li, Qifeng Chen, and Vladlen Koltun. “Combinatorial optimization with graph convolutional networks and guided tree search”. In: *NeurIPS* (2018).

[41] Xiaodong Liu, Pengcheng He, Weizhu Chen, and Jianfeng Gao. “Multi-Task Deep Neural Networks for Natural Language Understanding”. In: *ACL*. 2019.

[42] Zemin Liu, Yuan Fang, Chenghao Liu, and Steven CH Hoi. “Relative and Absolute Location Embedding for Few-Shot Node Classification on Graph”. In: *AAAI* (2021).

[43] Zemin Liu, Wentao Zhang, Yuan Fang, Xinming Zhang, and Steven CH Hoi. “Towards locality-aware meta-learning of tail node embeddings on networks”. In: *CIKM*. 2020, pp. 975–984.

[44] Ning Ma, Jiajun Bu, Jieyu Yang, Zhen Zhang, Chengwei Yao, Zhi Yu, Sheng Zhou, and Xifeng Yan. “Adaptive-Step Graph Meta-Learner for Few-Shot Graph Classification”. In: *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. 2020, 1055–1064.

[45] Sahil Manchanda, Akash Mittal, Anuj Dhawan, Sourav Medya, Sayan Ranu, and Ambuj Singh. “GCOMB: Learning Budget-constrained Combinatorial Algorithms over Billion-sized Graphs”. In: *NeurIPS* (2020).

[46] Andreas Maurer, Massimiliano Pontil, and Bernardino Romera-Paredes. “The benefit of multitask representation learning”. In: *Journal of Machine Learning Research* 17.81 (2016), pp. 1–32.

[47] Sourav Medya, Jithin Vachery, Sayan Ranu, and Ambuj Singh. “Noticeable network delay minimization via node upgrades”. In: *VLDB* (2018).

[48] Anusha Nagabandi, Kurt Konolige, Sergey Levine, and Vikash Kumar. “Deep dynamics models for learning dexterous manipulation”. In: *CoRL*. 2020.

[49] Sunil Nishad, Shubhangi Agarwal, Arnab Bhattacharya, and Sayan Ranu. “GraphReach: Locality-Aware Graph Neural Networks using Reachability Estimations”. In: *IJCAI*. 2021.

[50] Zheyi Pan, Wentao Zhang, Yuxuan Liang, Weinan Zhang, Yong Yu, Junbo Zhang, and Yu Zheng. “Spatio-Temporal Meta Learning for Urban Traffic Prediction”. In: *TKDE* (2020).

[51] Massimiliano Pontil and Andreas Maurer. “Excess risk bounds for multitask learning with trace norm regularization”. In: *COLT*. 2013, pp. 55–76.

[52] Sachin Ravi and Hugo Larochelle. “Optimization as a model for few-shot learning”. In: *ICLR*. 2017.

[53] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. “The graph neural network model”. In: *IEEE transactions on neural networks* 20.1 (2008), pp. 61–80.

[54] Franco Scarselli, Ah Chung Tsoi, and Markus Hagenbuchner. “The Vapnik–Chervonenkis dimension of graph and recursive neural networks”. In: *Neural Networks* 108 (2018), pp. 248–259.

[55] Jürgen Schmidhuber. “Evolutionary principles in self-referential learning, or on learning how to learn: the meta-meta-... hook”. PhD thesis. Technische Universität München, 1987.

[56] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. “Pitfalls of graph neural network evaluation”. In: *arXiv preprint arXiv:1811.05868* (2018).

[57] Jonathan M Stokes, Kevin Yang, Kyle Swanson, Wengong Jin, Andres Cubillos-Ruiz, et al. “A deep learning approach to antibiotic discovery”. In: *Cell* (2020), pp. 688–702.

[58] Jie Tang, Jing Zhang, Limin Yao, Juanzi Li, Li Zhang, and Zhong Su. “Arnetminer: extraction and mining of academic social networks”. In: *KDD*. 2008.

[59] Nilesh Tripuraneni, Chi Jin, and Michael I Jordan. “Provable meta-learning of linear representations”. In: *arXiv preprint arXiv:2002.11684* (2020).

[60] Nilesh Tripuraneni, Michael Jordan, and Chi Jin. “On the Theory of Transfer Learning: The Importance of Task Diversity”. In: *NeurIPS* 33 (2020).

[61] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. “Graph attention networks”. In: *ICLR* (2018).

[62] Ning Wang, Minnan Luo, Kaize Ding, Lingling Zhang, Jundong Li, and Qinghua Zheng. “Graph Few-shot Learning with Attribute Matching”. In: *CIKM*. 2020, pp. 1545–1554.

[63] Yan Wang, Wei-Lun Chao, Kilian Q Weinberger, and Laurens van der Maaten. “Simpleshot: Revisiting nearest-neighbor classification for few-shot learning”. In: *arXiv:1911.04623* (2019).

[64] Zhihao Wen, Yuan Fang, and Zemin Liu. “Meta-Inductive Node Classification across Graphs”. In: *arXiv:2105.06725* (2021).

[65] Bryan Wilder, Han Ching Ou, Kayla de la Haye, and Milind Tambe. “Optimizing Network Structure for Preventative Health”. In: *AAMAS*. 2018.

[66] Nan Wu, Jason Phang, Jungkyu Park, Yiqiu Shen, et al. “Deep neural networks improve radiologists’ performance in breast cancer screening”. In: *IEEE transactions on medical imaging* 39.4 (2019), pp. 1184–1194.

[67] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. “A comprehensive survey on graph neural networks”. In: *IEEE transactions on neural networks and learning systems* (2020).

[68] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. “How powerful are graph neural networks?” In: *ICLR* (2018).

[69] Pinar Yanardag and SVN Vishwanathan. “Deep graph kernels”. In: *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*. 2015, pp. 1365–1374.

[70] Huaxiu Yao, Chuxu Zhang, Ying Wei, Meng Jiang, Suhang Wang, Junzhou Huang, Nitesh Chawla, and Zhenhui Li. “Graph few-shot learning via knowledge transfer”. In: *AAAI*. 2020, pp. 6656–6663.

[71] Jiaxuan You, Rex Ying, and Jure Leskovec. “Position-aware Graph Neural Networks”. In: *ICML*. 2019, pp. 7134–7143.

[72] Jiani Zhang, Xingjian Shi, Junyuan Xie, Hao Ma, Irwin King, and Dit-Yan Yeung. “Gaan: Gated attention networks for learning on large and spatiotemporal graphs”. In: *UAI* (2018).

[73] Muhan Zhang and Yixin Chen. “Link prediction based on graph neural networks”. In: *NeurIPS* (2018).

[74] Xi Sheryl Zhang, Fengyi Tang, Hiroko H Dodge, Jiayu Zhou, and Fei Wang. “Metapred: Meta-learning for clinical risk prediction with limited patient electronic health records”. In: *KDD*. 2019.

[75] Fan Zhou, Chengtai Cao, Goce Trajcevski, Kunpeng Zhang, Ting Zhong, and Ji Geng. “Fast network alignment via graph meta-learning”. In: *INFOCOM*. 2020, pp. 686–695.

[76] Fan Zhou, Chengtai Cao, Kunpeng Zhang, Goce Trajcevski, Ting Zhong, and Ji Geng. “Meta-Gnn: On Few-Shot Node Classification in Graph Meta-Learning”. In: *CIKM*. 2019, pp. 2357–2360.

[77] Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. “Graph neural networks: A review of methods and applications”. In: *arXiv preprint arXiv:1812.08434* (2018).