Graph Representation Learning for Multi-Task Settings: a Meta-Learning Approach

Davide Buffelli
Department of Information Engineering
University of Padova
Padova, Italy
davide.buffelli@unipd.it

Fabio Vandin
Department of Information Engineering
University of Padova
Padova, Italy
fabio.vandin@unipd.it

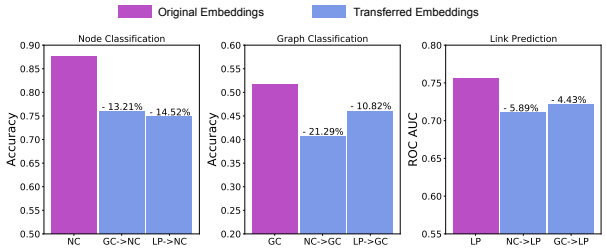
***Abstract*—Graph Neural Networks (GNNs) have become the state-of-the-art method for many applications on graph structured data. GNNs are a model for *graph representation learning*, which aims at learning to generate low dimensional node embeddings that encapsulate structural and feature-related information. GNNs are usually trained in an end-to-end fashion, leading to highly specialized node embeddings. While this approach achieves great results in the single-task setting, the generation of node embeddings that can be used to perform multiple tasks (with performance comparable to single-task models) is still an open problem. We propose the use of meta-learning to allow the training of a GNN model capable of producing *multi-task* node embeddings. In particular, we exploit the properties of optimization-based meta-learning to learn GNNs that can produce general node representations by learning parameters that can quickly (i.e. with a few steps of gradient descent) adapt to multiple tasks. Our experiments show that the embeddings produced by a model trained with our purposely designed meta-learning procedure can be used to perform multiple tasks with comparable or, surprisingly, even higher performance than both single-task and multi-task end-to-end models.**

***Index Terms*—Machine Learning, Representation Learning, Artificial Neural Networks, Graph Neural Networks, Graph Representation Learning**

I. Introduction

Graph Neural Networks (GNNs) are deep learning models for graph structured data, and have become one of the main topics of the deep learning research community. The interest in GNNs is due, in part, to their great empirical performance on many graph-related tasks. Three tasks in particular, with many practical applications, have received the most attention: graph classification, node classification, and link prediction.

GNNs are centered around the concept of *node representation learning*, and typically follow the same architectural pattern with an *encoder-decoder* structure [5, 12, 38]. The encoder produces node embeddings (low-dimensional vectors capturing relevant structural and feature-related information about each node), while the decoder uses the embeddings to carry out the desired downstream task. The model is then trained in an end-to-end manner, leading to highly specialized node embeddings. While this approach can achieve state-of-the-art performance, it also affects the generality and reusability of the embeddings. In fact, taking the node embeddings generated by an encoder trained for a given task, and using them to train a decoder for a different task leads to substantial performance loss (see Fig. 1).



(a) (b) (c)

Fig. 1: Performance drop when transferring node embeddings on the ENZYMES dataset to perform the tasks: (a) Node Classification (NC), (b) Graph Classification (GC), and (c) Link Prediction (LP). "x ->y" indicates that the embeddings obtained from a model trained on task x are used to train a network for task y.

The low transferability of node embeddings requires the use of one specialized encoder and one specialized decoder for each considered task. However, there are many practical scenarios in which *multiple* tasks must be performed on the same graph(s). For example, in a social network, both classification tasks (e.g., classify users as spammer or nonspammer) and link prediction tasks (e.g., suggest new connections between users) must be performed. While a trivial solution might be to deploy a specific model (generating specific node embeddings) for each task, this inevitably leads to significant overhead.

This paper studies the problem of generating node embeddings that can be used for multiple tasks, which is a scenario with many practical applications that, nonetheless, has received little attention from the GNN community. In more detail, we propose a multi-task representation learning procedure, based on optimization-based meta-learning [9], that learns a GNN encoder producing node embeddings that generalize across multiple tasks. Our focus is on the most studied tasks in the GNN literature: graph classification, node classification, and link prediction.

The proposed meta-learning procedure is targeted towards multi-task representation learning and takes advantage of MAML [9] and ANIL [29] to reach a setting of the parameters

where a few steps of gradient descent on a given task lead to good performance on that task. This procedure leads to an encoder-decoder model that can easily be adapted to perform each of the tasks *singularly*, and hence encourages the encoder to learn representations that can be reused across tasks. At the end of the training procedure, the decoder is discarded, and the encoder GNN is used to generate embeddings.

We summarize our contributions as follows:

- We consider the under-studied problem of learning GNN models generating node representations that can be used to perform multiple tasks. In this regard, we design a meta-learning strategy for training GNN models with such capabilities.
- To the best of our knowledge, we are the first to propose a GNN model generating a *single* set of node embeddings that can be used to perform the three most common graph-related tasks (i.e., graph classification, node classification, and link prediction). In particular, the generated embeddings lead to comparable or even higher performance with respect to separate end-to-end trained single-task models.
- We show that the episodic training strategy at the base of our meta-learning procedure leads to a model generating node embeddings that are more effective for downstream tasks, even in single-task settings. This unexpected finding is of interest in itself, and may provide fruitful directions for future research.

II. Preliminaries

This section introduces GNNs (Section II-A), multi-head models (Section II-B), and optimization-based meta-learning techniques (Section II-C), which are at the core of our method. Throughout the paper we use the term "*task*" as in the multi-task learning (MTL) literature, i.e. to refer to a downstream application (e.g. graph classification, node classification, etc.).

A. Graph Neural Networks

Many popular state-of-the-art GNN models follow the *message-passing* paradigm [11], which we now briefly describe. We represent a graph $\mathcal{G} = (\mathbf{A}, \mathbf{X})$ with an adjacency matrix $\mathbf{A} \in \{0,1\}^{n \times n}$, such that $\mathbf{A}_{ij} = 1$ if and only if there is an edge between the i-th vertex and the j-th vertex, and a node feature matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$, where the v-th row $\mathbf{X}_v$ represents the d dimensional feature vector of node v. Let $\mathbf{H}^{(\ell)} \in \mathbb{R}^{n \times d'}$ be the matrix containing the node representations at layer ℓ. A message passing layer updates the representation of every node v as follows:

$$msg_v^{(\ell)} = \text{AGGREGATE}(\{\mathbf{H}_u^{(\ell)} \ \forall u \in \mathcal{N}_v\})$$
$$\mathbf{H}_v^{(\ell+1)} = \text{UPDATE}(\mathbf{H}_v^{(\ell)}, msg_v^{(\ell)})$$

where $\mathbf{H}^{(0)} = \mathbf{X}$, $\mathcal{N}_v$ is the set of neighbours of node v, AGGREGATE is a permutation invariant function (as it takes a set as input), and UPDATE is usually a neural network. After L message-passing layers, the final node embeddings $\mathbf{H}^{(L)}$ are the representations used to perform a given task (e.g., they are the input to a neural component that performs the given task), and the network is trained end-to-end.

B. Multi-Head Models

In deep learning, the standard approach for performing multiple tasks [35] with the same model is to employ a *multi-head* architecture (see Fig. 2 (a)). A multi-head model is composed of a *backbone* and multiple *heads* (one for each task). The backbone is a neural network which processes the input to extract features. The features extracted by the backbone are then used by the heads (which are also neural networks) to perform the desired tasks (each head performs one task). The whole model is then trained end-to-end to minimize a combination of the single-task losses (e.g. the sum of the losses on each task). We refer to this strategy as the *classical* training procedure for multi-task models.

C. Model-Agnostic Meta-Learning and ANIL

MAML (Model-Agnostic Meta-Learning) is an optimization-based meta-learning strategy proposed by Finn et al. [9]. Let f_θ be a deep learning model, where θ represents its parameters. Let $p(\mathcal{E})$ be a distribution over episodes[1], with an episode $\mathcal{E}_i \sim p(\mathcal{E})$ being defined as a tuple containing a *loss function* $\mathcal{L}_{\mathcal{E}_i}(\cdot)$, a *support set* $\mathcal{S}_{\mathcal{E}_i}$, and a *target set* $\mathcal{T}_{\mathcal{E}_i}$: $\mathcal{E}_i = (\mathcal{L}_{\mathcal{E}_i}(\cdot), \mathcal{S}_{\mathcal{E}_i}, \mathcal{T}_{\mathcal{E}_i})$, where support and target sets are simply sets of labelled examples. MAML's goal is to find a value of θ that can quickly, i.e. in a few steps of gradient descent, be adapted to new episodes. This is done with a nested loop optimization procedure: an *inner loop* adapts the parameters to the support set of an episode by performing some steps of gradient descent, and an *outer loop* updates the initial parameters aiming at a setting that allows fast adaptation. Formally, by defining $\theta'_i(t)$ as the parameters after t adaptation steps on the support set of episode $\mathcal{E}_i$, we can express the computations in the inner loop as

$$\theta'_i(t) = \theta'_i(t-1) - \alpha \nabla_{\theta'_i(t-1)} \mathcal{L}_{\mathcal{E}_i}(f_{\theta'_i(t-1)}, \mathcal{S}_{\mathcal{E}_i}) \quad (1)$$

where $\theta'_i(0) = \theta$, $\mathcal{L}(f_{\theta'_i(t-1)}, \mathcal{S}_{\mathcal{E}_i})$ indicates the loss over the support set $\mathcal{S}_{\mathcal{E}_i}$ for the model $f_{\theta'_i(t-1)}$ with parameters $\theta'_i(t-1)$, and α is the learning rate. The *meta-objective* that the outer loop tries to minimize is defined as $\mathcal{L}_{meta} = \sum_{\mathcal{E}_i \sim p(\mathcal{E})} \mathcal{L}_{\mathcal{E}_i}(f_{\theta'_i(t)}, \mathcal{T}_{\mathcal{E}_i})$, which leads to the following parameter update[2] performed in the outer loop:

$$\theta = \theta - \beta \nabla_\theta \mathcal{L}_{meta} = \theta - \beta \nabla_\theta \sum_{\mathcal{E}_i \sim p(\mathcal{E})} \mathcal{L}_{\mathcal{E}_i}(f_{\theta'_i(t)}, \mathcal{T}_{\mathcal{E}_i}). \quad (2)$$

Raghu et al. [29] showed that feature reuse is the dominant factor in MAML: in the adaptation loop, only the last layer(s) in the network are updated, while the first layer(s) remain almost unchanged. The authors then propose ANIL (Almost No Inner Loop) where they split the parameters in two sets:

[1]The meta-learning literature usually derives episodes from *tasks* (i.e., tuples containing a dataset and a loss function). We focus on episodes to avoid using the term *task* for both a MTL task, and a meta-learning task.

[2]We limit ourself to one step of gradient descent for clarity, but any optimization strategy could be used.

one that is used for adaptation in the inner loop, and one that is only updated in the outer loop. This simplification leads to computational improvements while maintaining performance.

III. SAME: Single-Task Adaptation for Multi-Task Embeddings

We design a meta-learning approach targeted towards representation learning, by building on three insights:

***(i)* optimization-based meta-learning is implicitly learning robust representations.** The findings by Raghu et al. [29] suggest that, in a model trained with MAML, the first layers learn features that are reusable across episodes, while the last layers are set up for fast adaptation. MAML is then *implicitly* learning a model with two components: an *encoder* (the first layers), focusing on learning reusable representations that generalize across episodes, and a *decoder* (the last layers) that can be quickly adapted for different episodes.

***(ii)* meta-learning episodes can be designed to encourage generalization.** By designing support and target sets to mimic the training and validation sets of a classical training procedure, then the meta-learning procedure is effectively optimizing for generalization.

***(iii)* meta-learning can learn to quickly adapt to multiple tasks *singularly*, without having to learn to solve multiple tasks *concurrently*.** The meta-learning procedure can be deisgned so that, for each considered task, the inner loop adapts the parameters to a task-specific support set, and tests the adaptation on a task-specific target set. The outer loop then updates the parameters to allow this fast *multiple single-task adaptation*.

Based on ***(ii)*** and ***(iii)***, we design the meta-learning procedure such that the inner loop adapts to multiple tasks *singularly*, each time with the goal of *single-task generalization*. Using an encoder-decoder architecture, ***(i)*** suggests that this procedure leads to an encoder that learns features reusable across episodes. As, in each episode, the learner is adapting to multiple tasks, the encoder is learning features that generalize across multiple tasks. After training with our meta-learning strategy, the decoder is discarded, and only the encoder is kept and used to generate representations. Contrary to many applications of meta-learning, there is no adaptation performed at test time, as meta-learning is used only for training the model from which an encoder is exracted.

In the rest of this section, we formally present our meta-learning procedure for training multi-task graph representation learning models. There are three aspects that need to be defined: **(1) Episode Design:** how is a an episode composed, **(2) Model Architecture Design:** what is the architecture of our model, **(3) Meta-Training Design:** how, and which, parameters are adapted/updated.

A. Episode Design

In our case, an episode becomes a *multi-task episode* (Fig. 2 (b)). To formally introduce the concept, let us consider the case where the tasks are graph classification (GC), node classification (NC), and link prediction (LP). We define a *multi-task episode* $\mathcal{E}_i^{(m)} \sim p(\mathcal{E}^{(m)})$ as a tuple $\mathcal{E}_i^{(m)} = (\mathcal{L}_{\mathcal{E}_i}^{(m)}, \mathcal{S}_{\mathcal{E}_i}^{(m)}, \mathcal{T}_{\mathcal{E}_i}^{(m)})$ where

$$\mathcal{L}_{\mathcal{E}_i}^{(m)} = \{\mathcal{L}_{\mathcal{E}_i}^{(\text{GC})}, \mathcal{L}_{\mathcal{E}_i}^{(\text{NC})}, \mathcal{L}_{\mathcal{E}_i}^{(\text{LP})}\},$$
$$\mathcal{S}_{\mathcal{E}_i}^{(m)} = \{\mathcal{S}_{\mathcal{E}_i}^{(\text{GC})}, \mathcal{S}_{\mathcal{E}_i}^{(\text{NC})}, \mathcal{S}_{\mathcal{E}_i}^{(\text{LP})}\},$$
$$\mathcal{T}_{\mathcal{E}_i}^{(m)} = \{\mathcal{T}_{\mathcal{E}_i}^{(\text{GC})}, \mathcal{T}_{\mathcal{E}_i}^{(\text{NC})}, \mathcal{T}_{\mathcal{E}_i}^{(\text{LP})}\}.$$

The meta-objective $\mathcal{L}_{meta}^{(m)}$ of our method is then defined as:

$$\mathcal{L}_{meta}^{(m)} = \sum_{\mathcal{E}_i^{(m)} \sim p(\mathcal{E}^{(m)})} \lambda^{(GC)} \mathcal{L}_{\mathcal{E}_i}^{(\text{GC})} + \lambda^{(NC)} \mathcal{L}_{\mathcal{E}_i}^{(\text{NC})} + \lambda^{(LP)} \mathcal{L}_{\mathcal{E}_i}^{(\text{LP})}. \tag{3}$$

where $\lambda^{(\cdot)}$ are balancing coefficients.

Support and target sets are set up to resemble training and validation sets. This way the outer loop's objective becomes to *maximize the performance on a validation set, given a training set*, hence encouraging generalization. In more detail, given a batch of graphs, we divide it in equally sized splits (one per task), and create support and target sets as follows:

- **Graph Classification:** $\mathcal{S}_{\mathcal{E}_i}^{(\text{GC})}$ and $\mathcal{T}_{\mathcal{E}_i}^{(\text{GC})}$ contain labeled graphs, obtained with a random split.
- **Node Classification:** $\mathcal{S}_{\mathcal{E}_i}^{(\text{NC})}$ and $\mathcal{T}_{\mathcal{E}_i}^{(\text{NC})}$ are composed of the same graphs, with different labelled nodes. We mimic the common semi-supervised setting [19] where feature vectors are available for all nodes, and only a small subset of nodes is labelled.
- **Link Prediction:** $\mathcal{S}_{\mathcal{E}_i}^{(\text{LP})}$ and $\mathcal{T}_{\mathcal{E}_i}^{(\text{LP})}$ are composed of the same graphs, with different query edges. In every graph we randomly remove some edges, used as positive examples together with non-removed edges, and randomly sample pairs of non-adjacent nodes as negative examples.

Notice how we only need labels for *one* task for each graph. The full algorithm for the creation of *multi-task episodes* is provided in Appendix[3] A.

B. Model Architecture Design

We use an encoder-decoder model with a multi-head architecture. The *backbone* (which represents the encoder) is composed of 3 GCN [19] layers with ReLU non-linearities and residual connections [14]. The decoder is composed of three *heads*. The node classification head is a single layer neural network with a *Softmax* activation that is shared across nodes and maps node embeddings to class predictions. In the graph classification head, first a single layer neural network (shared across nodes) performs a linear transformation (followed by a ReLU activation) of the node embeddings. The transformed node embeddings are then averaged and a final single layer neural network with *Softmax* activation outputs the class predictions. The link prediction head is composed of a single layer neural network with ReLU non-linearity that transforms node embeddings, and a single layer neural network that given concatenation of two embeddings outputs the probability of a link between them. We remark that after training the full model

[3]Appendix available at: https://arxiv.org/abs/2201.03326.

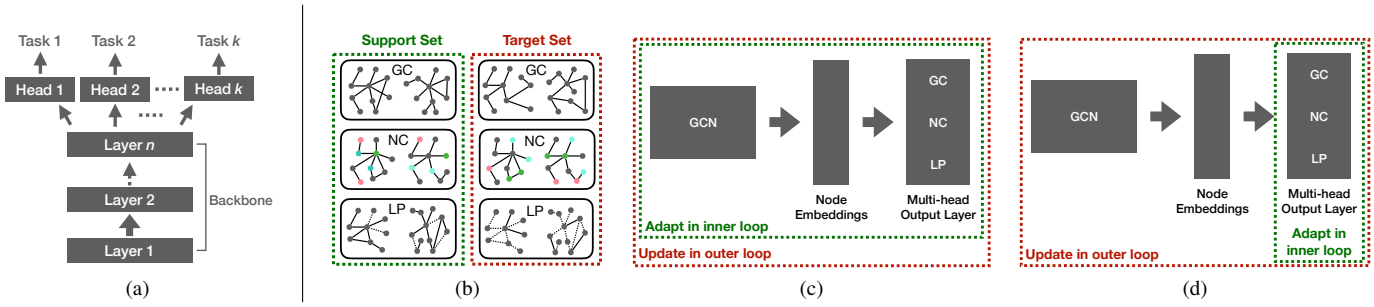


Fig. 2: **Main ingredients of our meta-learning procedure SAME.** (a) Multi-head architecture. (b) Schematic representation of a *multi-task episode*. For each task, support and target set are designed to be as the training and validation sets for single-task training. (c-d) Overview of the parameter updates in SAME's meta-learning procedure. In the inner loop, the model is adapted *separately* to each task in the support set; the outer loop then tests the performance of each adaptation on the corresponding task in the target set, and updates the initial parameters of the network to allow it to rapidly be adapted to each task, by minimizing the meta-objective. In iSAME (c) all parameters are adapted in the inner loop. In eSAME (d) only the task-specific output layers are adapted in the inner loop. Both in iSAME and eSAME, after training the model, only the backbone GCN is kept, and used to generate embeddings.

Algorithm 1 Proposed (meta-learning based) procedure.

Input: Model f_θ; Episodes $\mathcal{E} = \{\mathcal{E}_1, .., \mathcal{E}_n\}$; Meta-Objective Coefficients $\lambda^{(GC)}, \lambda^{(NC)}, \lambda^{(LP)}$.

```
init(θ)
for E_i in E do
    o_loss ← 0
    for τ in (GC, NC, LP) do
        θ'^(τ) ← θ
        θ'^(τ) ← ADAPT(f_θ, S_{E_i}^(τ), L_{E_i}^(τ))
        o_loss ← o_loss + λ^(τ) TEST(f_{θ'^(τ)}, T_{E_i}^(τ), L_{E_i}^(τ))
    end for
    θ ← UPDATE(θ, o_loss, θ'^(GC), θ'^(NC), θ'^(LP))
end for
```

with the proposed meta-learning procedure, only the encoder is kept, and is used to generate node embeddings which can be fed to any machine learning model for downstream tasks.

C. Meta-Training Design

We first present the meta-learning training procedure, and successively describe which parameters are adapted/updated in the inner and outer loops.

Meta-Learning Training Procedure. The meta-learning procedure is designed such that the inner loop adaptation involves a *single task* at a time. Only the parameter update performed to minimize the meta-objective involves multiple tasks, but, crucially, it does not aim at a setting of parameters that can solve, or quickly adapt to, multiple tasks *concurrently*, but to a setting allowing **multiple fast single-task adaptation**.

The pseudocode of our procedure is in Algorithm 1. `init` is a method that initializes the weights of the GNN. `ADAPT` performs a few steps of gradient descent on a *task-specific* loss function and support set (as in eq. 1), `TEST` computes the value of the meta-objective component on a *task-specific* loss function and target set for a model with parameters adapted on that task, and `UPDATE` optimizes the parameters θ by minimizing the meta-objective in eq. 3 (which is contained in `o_loss` in the pseudocode). Notice the multiple heads of the decoder are never used concurrently.

Parameter Update in Inner/Outer Loop. Let us partition the parameters of our model in four sets: $\theta = [\theta_{\text{GCN}}, \theta_{\text{NC}}, \theta_{\text{GC}}, \theta_{\text{LP}}]$ representing the parameters of the backbone (θ_{GCN}), node classification head (θ_{NC}), graph classification head (θ_{GC}), and link prediction head (θ_{LP}). We name our meta-learning strategy SAME (Single-Task Adaptation for Multi-Task Embeddings), and present two variants (Fig. 2 c-d):

- ***Implicit* SAME (iSAME):** all the parameters θ are used for adaptation. This strategy makes use of the *implicit* feature-reuse factor of MAML, leading to parameters θ_{GCN} that are general across *multi-task episodes*.
- ***Explicit* SAME (eSAME):** only the head parameters $\theta_{\text{NC}}, \theta_{\text{GC}}, \theta_{\text{LP}}$ are used for adaptation (as done by ANIL). Contrary to iSAME, this strategy *explicitly* aims at learning the parameters θ_{GCN} to be general across *multi-task episodes* by only updating them in the outer loop.

The pseudocode in Algorithm 1 is the same for both iSAME and eSAME. The difference between the methods is in which subset of the parameters θ is updated by the `ADAPT` function. In iSAME the `ADAPT` function will update the head and the backbone parameters ($\theta_{\text{GCN}}, \theta_{\text{NC}}, \theta_{\text{GC}}, \theta_{\text{LP}}$), while for eSAME only the head parameters ($\theta_{\text{NC}}, \theta_{\text{GC}}, \theta_{\text{LP}}$) will be updated.

D. Connection between SAME and other Optimization-based Meta-Learning Methods

SAME is an instantiation of optimization-based meta-learning (in particular iSAME is an instantiation of MAML [9], while eSAME of ANIL [29]) specially designed for learning multi-task representations. In particular SAME employs the following design choices: **(1)** In SAME each episode is composed of multiple tasks (i.e. downstream applications). **(2)** In SAME each task (both in the inner and in the outer loop)

can involve only a subset of the parameters of the model. **(3)** In SAME's inner loop, *separate* adaptations are performed for each task in the episode. In the outer loop, the meta-objective defines how these multiple adaptations are combined for updating the initial representations of the parameters.

After training a model with SAME, an encoder is extracted and used to generate representations of the input that can then be fed to any machine learning model. As SAME is only used for training, no adaptation is performed at test time, and hence support and target sets are not required at test time.

IV. Experiments

Our goal is to assess the quality of the representations learned by models trained with SAME, and to study the impact of SAME's underlying components. In more detail, we aim to answer the following questions:

- **Q1:** *Do iSAME and eSAME lead to node embeddings that can be used to perform multiple downstream tasks with comparable (or better) performance than end-to-end single-task models?*
- **Q2:** *Can node embeddings learned by a model trained with iSAME and eSAME be used for multiple tasks with comparable or better performance than classically trained (i.e., see Section II-B) multi-task models?*
- **Q3:** *Do iSAME and eSAME extract information that is not captured by the classical training procedure (i.e., see Section II-B)?*
- **Q4** ***(Ablation Study)*:** *What are the contributions of the different components of SAME's meta-learning procedure?*

Unless otherwise stated, accuracy (%) is used for NC and GC, while ROC AUC (%) is used for LP. (As a reminder, we use GC to refer to graph classification, NC for node classification, and LP for link prediction.)

Datasets. To perform multiple tasks, we consider datasets with graph labels, node attributes, and node labels from the TUDataset library [27]: ENZYMES [32], PROTEINS [8], DHFR and COX2 [34]. ENZYMES is a dataset of protein structures belonging to six classes. PROTEINS is a dataset of chemical compounds with two classes (enzyme and non-enzyme). DHFR, and COX2 are datasets of chemical inhibitors which can be active or inactive.

Experimental Setup. We perform a 10-fold cross validation, and average results across folds. To ensure a fair comparison, the same architecture is used for all training strategies. We set $\lambda^{(GC)} = \lambda^{(NC)} = \lambda^{(LP)} = 1$ as we noticed that weighting the losses did not provide significant benefits. Loss balancing techniques (e.g. *uncertainty weights* [17], and *gradnorm* [6]) were tested, both with SAME and with the classical training procedure, but they did not result effective. This is in accordance with recent works [20, 35] which observe that, when appropriate tuning is done, no method is significantly better than minimizing the sum of the task losses. For more information we refer to Appendix B, and we publicly release source code[4].

[4]https://github.com/DavideBuffelli/SAME

TABLE I: Results for a single-task model trained in a classical supervised manner, a fine-tuned model (trained on all, and fine-tuned on two tasks), and a **linear** classifier trained on node embeddings generated by a model trained with our strategies (iSAME, eSAME) in a multi-task setting.

Task			Dataset											
GC	NC	LP	ENZYMES			PROTEINS			DHFR			COX2		
			GC	NC	LP	GC	NC	LP	GC	NC	LP	GC	NC	LP
Classical End-to-End Training														
✓			51.6			73.3			71.5			76.7		
	✓			87.5			72.3			97.3			96.4	
		✓			75.5			85.6			98.8			98.3
Fine-Tuning														
✓	✓		48.3	85.3		73.6	72.0		66.4	92.4		80.0	92.3	
✓		✓	49.3		71.6	69.6		80.7	65.3		58.9	80.2		50.9
	✓	✓		87.7	73.9		80.4	81.5		80.7	56.6		87.4	52.3
iSAME (ours)														
✓	✓		50.1	86.1		73.1	76.6		71.6	94.8		75.2	95.4	
✓		✓	50.7		83.1	73.4		85.2	71.6		99.2	77.5		98.9
	✓	✓		86.3	83.4		79.4	87.7		96.5	99.3		95.5	99.0
✓	✓	✓	50.0	86.5	82.3	71.4	76.6	87.3	71.2	95.5	99.5	75.4	95.2	99.2
eSAME (ours)														
✓	✓		51.7	86.1		71.5	79.2		70.1	95.7		75.6	95.5	
✓		✓	51.9		80.1	71.7		85.4	70.1		99.1	77.5		98.8
	✓	✓		86.7	82.2		80.7	86.3		96.6	99.4		95.6	99.1
✓	✓	✓	51.5	86.3	81.1	71.3	79.6	86.8	70.2	95.3	99.5	77.7	95.7	98.8

Q1: We train a model with SAME, on all multi-task combinations, and use the embeddings produced by the learned encoder as the input for a **linear classifier**. We compare against models with the same task-specific architecture trained in a classical supervised manner on a single task, and with a fine-tuning baseline. The latter is a model that has been trained on all three tasks, and then fine-tuned on two specific tasks. The idea is that the initial training on all tasks should lead the model towards the extraction of features that it would otherwise not consider (by only seeing 2 tasks). The fine-tuning process should then allow the model to use these features to target the specific tasks of interest. Results are shown in Table I. The embeddings produced by the model learned with SAME in a multi-task setting achieve performance comparable to, and frequently even better than, end-to-end single-task models. In fact, the embeddings from SAME are never outperformed by more than 3%, and in 50% of the cases actually achieve higher performance. Moreover, the fine-tuning baseline is almost always outperformed by both single-task models, and our proposed methods. These results confirm that meta-learning is a powerful solution for multi-task representation learning on graphs.

Q2: We train the same multi-task model, both in the classical supervised manner (see Section II-B), and with our proposed approaches, on all multi-task combinations. For our approaches, a **linear classifier** is then trained on top of the node embeddings produced by the learned encoder. We further consider the fine-tuning baseline introduced in **Q1**. The multi-task performance (Δ_m) metric [25] is used, defined as the average per-task drop with respect to the single-task baseline: $\Delta_m = \frac{1}{T}\sum_{i=1}^{T}\left(M_{m,i} - M_{b,i}\right)/M_{b,i}$, where $M_{m,i}$ is the value of a task's metric for the multi-task model, and $M_{b,i}$ is the value for the baseline. Results are shown in Table II. Multi-task models usually achieve lower performance than specialized single-task ones. Moreover, **linear** classifiers trained on the embeddings generated by a model trained with SAME are not only comparable, but in many cases

TABLE II: Δ_m (%) results for a classically trained multi-task model (Cl), a fine-tuned model (FT; trained on all three tasks and fine-tuned on two) and a **linear** classifier trained on the node embeddings generated by a model trained with our meta-learning strategies (iSAME, eSAME) in a multi-task setting.

Task			Model	Dataset			
GC	NC	LP		ENZYMES	PROTEINS	DHFR	COX2
✓	✓		Cl	-0.1 ± 0.5	4.0 ± 1.0	-0.3 ± 0.2	0.5 ± 0.1
			FT	-4.5 ± 1.2	0.1 ± 0.5	-7.4 ± 1.4	0.1 ± 0.4
			iSAME	-2.3 ± 0.9	2.7 ± 1.5	-1.2 ± 0.4	-1.6 ± 0.2
			eSAME	-0.8 ± 0.8	3.2 ± 1.4	-1.8 ± 0.3	-1.2 ± 0.3
✓		✓	Cl	-25.3 ± 3.2	-5.3 ± 1.2	-28.3 ± 4.3	-21.4 ± 3.4
			FT	-5.1 ± 1.9	-5.4 ± 1.5	-24.5 ± 3.7	-22.6 ± 3.8
			iSAME	4.1 ± 0.5	-0.2 ± 0.9	0.2 ± 3.2	0.2 ± 0.5
			eSAME	3.2 ± 0.4	-1.2 ± 1.1	-0.7 ± 3.4	-0.8 ± 0.7
	✓	✓	Cl	7.2 ± 2.7	6.8 ± 0.9	-29.1 ± 7.7	-28.2 ± 4.5
			FT	-1.0 ± 0.3	3.1 ± 1.2	-28.9 ± 6.4	-28.3 ± 4.2
			iSAME	4.4 ± 1.1	6.1 ± 1.0	-0.1 ± 6.2	-0.6 ± 2.5
			eSAME	3.9 ± 1.3	6.1 ± 1.1	0.1 ± 6.4	-0.6 ± 2.6
✓	✓	✓	Cl	1.6 ± 1.3	2.9 ± 0.3	-18.9 ± 2.3	-16.9 ± 3.1
			iSAME	1.5 ± 1.0	2.2 ± 0.2	-0.5 ± 1.4	-0.9 ± 1.3
			eSAME	1.8 ± 0.9	2.8 ± 0.2	-1.0 ± 1.7	-0.4 ± 1.2

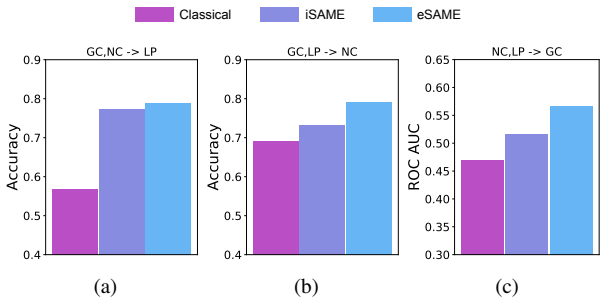


Fig. 3: Results for model, trained on the embeddings generated by a multi-task model, performing a task that was not seen during training. "x, y->z" indicates that x, y are the tasks used for training the multi-task model, and z is the new task.

significantly superior to classically trained multi-task models. In fact, a multi-task model trained in a classical manner is highly sensible to the tasks that are being learned (e.g. GC and LP negatively interfere with each other in every dataset), while our methods are much less sensible. For instance, the former has a worst-case average drop in performance of 29%, while our method has a worst-case average drop of less than 3%. Finally, the fine-tuning baseline generally performs worse than classically trained models, confirming that transferring knowledge in multi-task settings is not easy.

Q3: We train a multi-task model, and then train a new simple network (with the same architecture as the heads described in Section III-B), which is refer to as *classifier*, on the embeddings generated by the multi-task model to perform a task that was not seen during training. We compare the performance of the classifier on the embeddings generated by a model trained in a classical manner, and with SAME. Intuitively, this tests gives us a way to analyse if the embeddings generated by a model trained with SAME contain "more information" than embeddings generated by a model trained in a classical manner. Results on the ENZYMES dataset are shown in Fig. 3. Interestingly, the embeddings generated by a model trained with SAME lead to at least 10% higher performance. We observe an analogous trend on the other datasets (full results are in Appendix C).

Q4 ***(Ablation Study)*****:** SAME's meta-learning procedure has two main ingredients:

- ***(1)*** the design of support and target sets, to encourage generalization by mimicking training and validation sets (see Section III-A).
- ***(2)*** the separate multiple single-task adaptations performed in the inner loop, which relieve the model from having to learn to solve all the tasks *concurrently* (Section III-C).

To better understand the importance and contribution of each component we perform two experiments, for which results are presented below (the full results can be found in Appendix E).

First, we isolate the contribution of ***(1)*** by applying iSAME and eSAME in a *single-task* setting (i.e., the same *single* task is performed in both inner and outer loops), with episodes following the generalization-encouraging design proposed in Section III-A. Notice that this is like applying the original MAML and ANIL training procedures with our design of support and target sets. In this experiment, for every task, we train a **linear classifier** on top of the embeddings produced by a model trained with "single-task" iSAME and eSAME, and compare against a network with the same architecture trained in a classical end-to-end manner. Results are shown in Fig. 4. For all three tasks, a **linear** classifier on the embeddings produced by a model trained with our methods achieves comparable, if not superior, performance to an end-to-end model. In fact, the linear classifier is never outperformed by more than 2%, and it can outperform the classical end-to-end model by up to 12%. We believe this unexpected outcome is particularly interesting, and hints that episodic training procedures can be used to learn better representations.

Second, we investigate the benefits of ***(2)*** by removing the separate multiple single-task adaptations of SAME and performing *all* tasks (i.e., GC, NC, and LP) concurrently both in the inner and outer loop. This leads to a simple *concurrent multi-task version* of the conventional training procedure of MAML and ANIL, but with our support and target set design. For this experiment, we evaluate the ablated versions of SAME on the same procedure of **Q2** and **Q3**, and compare against the results of iSAME and eSAME. The results from the ablated version are not significantly different from those of *non-ablated* iSAME and eSAME (see Appendix D).

From these experiments we draw two conclusions. ***(i)*** The *generalization-encouraging* design of support and target sets is what allows SAME to reach performance on multiple tasks that are comparable to specialised single-task models trained in a classical manner. ***(ii)*** The separate *multiple single-task adaptations* that are performed in the inner loop of iSAME and eSAME allow the models to reach the same performance of a version of SAME where all tasks are performed concurrently on all graphs, hence increasing the learning efficiency by not requiring labels for each task on every graph.

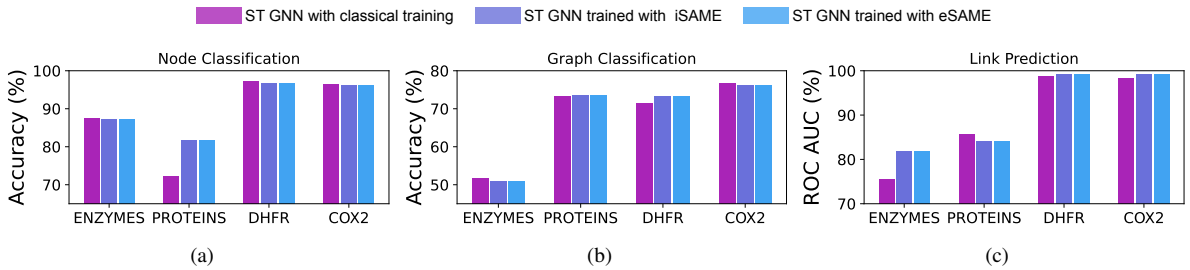


Fig. 4: Results for a Single-Task GNN model (ST GNN) trained with the classical procedure, and a **linear** classifier trained on the embeddings generated by a model trained with an ablated "single-task" version of SAME.

V. Related Work

GNNs, MTL, and meta-learning are very active areas of research. We highlight works that are at the intersection of these subjects, and point the interested reader to comprehensive reviews of each field. To the best of our knowledge there is no work using meta-learning to train a model for graph MTL, or proposing a GNN performing graph classification, node classification, and link prediction *concurrently*.

Graph Neural Networks. GNNs have a long history [31], but in the past few years the field has grown exponentially [5, 38]. The first popular GNN approaches were based on filters in the graph spectral domain [4], and presented many challenges including high computational complexity. Defferrard et al. [7] introduced ChebNet, which uses Chebyshev polynomials to produce localized and efficient filters in the graph spectral domain. Graph Convolutional Networks [19] introduced a localized first-order approximation of spectral graph convolutions which was then extended to include attention mechanisms [36]. Recently, Xu et al. [40] provided theoretical ground for the expressivity of GNNs.

Multi-Task Learning. Works at the intersection of MTL and GNNs have mostly focused on multi-head architectures. These models are composed of a series of GNN layers followed by multiple heads (i.e. independent neural network layers) that perform the desired downstream tasks. In this category, Montanari et al. [26] propose a model for the prediction of physico-chemical properties. Holtz et al. [15] and Xie et al. [39] propose multi-task models for concurrently performing node and graph classification. Finally, Avelar et al. [2] introduce a multi-head GNN for learning multiple graph centrality measures, and Li and Ji [21] propose a MTL method for the extraction of multiple biomedical relations. Other related work includes [13] which introduces a model that can be trained for several tasks singularly, hence, unlike the previously mentioned approaches and our proposed method, it can not perform multiple tasks concurrently. There are also some works that use GNNs as a tool for MTL: Liu et al. [23] use GNNs to allow communication between tasks, while Zhang et al. [42] use GNNs to estimate the test error of a MTL model. In summary, the current literature on graph MTL has focused on multi-head architectures that are trained end-to-end. In this work we tackle the graph representation learning scenario in which the node embeddings are used for multiple tasks, and propose the use of meta-learning for training a GNN for this setting. We further mention the work by Wang et al. [37] that considers the task of generating "general" node embeddings, however their method is not based on GNNs, does not consider node attributes (unlike our method), and is not focused on the three most common graph related tasks. For an exhaustive review of deep MTL techniques we refer the reader to Vandenhende et al. [35].

Meta-Learning. Meta-Learning consists in *learning to learn*. Many methods have been proposed (see the review by Hospedales et al. [16]), specially in the area of *few-shot learning*. Garcia and Bruna [10] frame the few-shot learning problem with a partially observed graphical model and use GNNs as an inference algorithm. Liu et al. [22] use GNNs to propagate messages between class prototypes and improve existing few-shot learning methods, while Suo et al. [33] use GNNs to introduce domain-knowledge in the form of graphs. There are also several works that use meta-learning to train GNNs in few-shot learning scenarios with applications to node classification [41, 43], edge labelling [18], link prediction [1, 3], and graph regression [28]. Finally, other combinations of meta-learning and GNNs involve adversarial attacks [44] and active learning [24].

VI. Conclusions

This work introduces the use of meta-learning as a training strategy for graph representation learning in multi-task settings. We find that our method leads to models that produce "more general" node embeddings. In fact, our results show that the embeddings produced by a model trained with our technique can be used to perform graph classification, node classification, and link prediction, with comparable or better performance than separate single-task end-to-end supervised models. Furthermore, we find that the embeddings generated by a model trained with our procedure lead to higher performance on downstream tasks that were not seen during training, and that the episodic training procedure leads to better embeddings even in the single-task setting. We believe this work can be of interest to the community as it explores the under-studied area of multi-task representation learning on graphs, and further introduces a method built on optimization-based meta-learning (inheriting the properties of being *model-agnostic*), which can be adapted to other domains as future

work. Another interesting direction is to incorporate more advanced meta-learning strategies like [30].

ACKNOWLEDGMENT

This work is supported, in part, by the Italian Ministry of Education, University and Research (MIUR), under PRIN Project n. 20174LF3T8 "AHeAD" and the initiative "Departments of Excellence" (Law 232/2016), and by University of Padova under project "SID 2020: RATED-X".

REFERENCES

[1] F. Alet, E. Weng, T. Lozano-Perez, and L. Kaelbling. Neural relational inference with fast modular meta-learning. In *NeurIPS*, 2019.

[2] P. Avelar, H. Lemos, M. Prates, and L. Lamb. Multitask learning on graph neural networks: Learning multiple graph centrality measures with a unified network. In *ICANN Workshop and Special Sessions*. 2019.

[3] A. J. Bose, A. Jain, P. Molino, and W. L. Hamilton. Meta-graph: Few shot link prediction via meta learning. *arXiv*, 2019.

[4] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst. Geometric deep learning: Going beyond euclidean data. *IEEE Signal Processing Magazine*, 2017.

[5] I. Chami, S. Abu-El-Haija, B. Perozzi, C. Ré, and K. Murphy. Machine learning on graphs: A model and comprehensive taxonomy. *arXiv*, 2020.

[6] Z. Chen, V. Badrinarayanan, C.-Y. Lee, and A. Rabinovich. Gradnorm: Gradient normalization for adaptive loss balancing in deep multitask networks. In *ICML*, 2018.

[7] M. Defferrard, X. Bresson, and P. Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *NeurIPS*, 2016.

[8] P. D. Dobson and A. J. Doig. Distinguishing enzyme structures from non-enzymes without alignments. *J. of Mol. Bio.*, 2003.

[9] C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *ICML*, 2017.

[10] V. Garcia and J. Bruna. Few-shot learning with graph neural networks. In *ICLR*, 2018.

[11] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. Neural message passing for quantum chemistry. In *ICML*, 2017.

[12] W. L. Hamilton, R. Ying, and J. Leskovec. Representation learning on graphs: Methods and applications. *IEEE Data Engineering Bulletin*, 2017.

[13] L. Haonan, S. H. Huang, T. Ye, and G. Xiuyan. Graph star net for generalized multi-task learning. *arXiv*, 2019.

[14] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[15] C. Holtz, O. Atan, R. Carey, and T. Jain. Multi-task learning on graphs with node and graph level labels. In *NeurIPS Workshop on Graph Representation Learning*, 2019.

[16] T. Hospedales, A. Antoniou, P. Micaelli, and A. Storkey. Meta-learning in neural networks: A survey. *arXiv*, 2020.

[17] A. Kendall, Y. Gal, and R. Cipolla. Multi-task learning using uncertainty to weigh losses for scene geometry and semantics. In *CVPR*, 2018.

[18] J. Kim, T. Kim, S. Kim, and C. Yoo. Edge-labeling graph neural network for few-shot learning. In *CVPR*, 2019.

[19] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017.

[20] V. Kurin, A. D. Palma, I. Kostrikov, S. Whiteson, and M. P. Kumar. In defense of the unitary scalarization for deep multi-task learning, 2022.

[21] D. Li and H. Ji. Syntax-aware multi-task graph convolutional networks for biomedical relation extraction. In *LOUHI*, 2019.

[22] L. Liu, T. Zhou, G. Long, J. Jiang, and C. Zhang. Learning to propagate for graph meta-learning. In *NeurIPS*, 2019.

[23] P. Liu, J. Fu, Y. Dong, X. Qiu, and J. Cheung. Learning multi-task communication with message passing for sequence learning. In *AAAI*, 2019.

[24] K. Madhawa and T. Murata. Active learning on graphs via meta learning. In *ICML Workshop on Graph Representation Learning and Beyond, ICML*, 2020.

[25] K.-K. Maninis, I. Radosavovic, and I. Kokkinos. Attentive single-tasking of multiple tasks. In *CVPR*, 2019.

[26] F. Montanari, L. Kuhnke, A. T. Laak, and D.-A. Clevert. Modeling physico-chemical ADMET endpoints with multitask graph convolutional networks. *Molecules*, 2019.

[27] C. Morris, N. M. Kriege, F. Bause, K. Kersting, P. Mutzel, and M. Neumann. Tudataset: A collection of benchmark datasets for learning with graphs. In *ICML Workshop on Graph Representation Learning and Beyond*, 2020.

[28] C. Q. Nguyen, C. Kreatsoulas, and B. K. M. Meta-learning gnn initializations for low-resource molecular property prediction. In *ICML Workshop on Graph Representation Learning and Beyond, ICML*, 2020.

[29] A. Raghu, M. Raghu, S. Bengio, and O. Vinyals. Rapid learning or feature reuse? towards understanding the effectiveness of maml. In *ICLR*, 2020.

[30] A. Rajeswaran, C. Finn, S. M. Kakade, and S. Levine. Meta-learning with implicit gradients. *NeurIPS*, 2019.

[31] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 2009.

[32] I. Schomburg, A. Chang, C. Ebeling, M. Gremse, C. Heldt, G. Huhn, and D. Schomburg. Brenda, the enzyme database: updates and major new developments. *Nucleic acids res.*, 2004.

[33] Q. Suo, J. Chou, W. Zhong, and A. Zhang. Tadanet: Task-adaptive network for graph-enriched meta-learning. In *ACM SIGKDD*, 2020.

[34] J. J. Sutherland, L. A. O'Brien, and D. F. Weaver. Spline-fitting with a genetic algorithm: A method for developing classification structure-activity relationships. *Journal of Chemical Information and Computer Sciences*, 2003.

[35] S. Vandenhende, S. Georgoulis, W. Van Gansbeke, M. Proesmans, D. Dai, and L. Van Gool. Multi-task learning for dense prediction tasks: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–1, 2021. doi: 10.1109/TPAMI.2021.3054719.

[36] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio. Graph Attention Networks. In *ICLR*, 2018.

[37] S. Wang, Q. Wang, and M. Gong. Multi-task learning based network embedding. *Frontiers in Neuroscience*, 2020.

[38] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[39] Y. Xie, M. Gong, Y. Gao, A. K. Qin, and X. Fan. A multi-task representation learning architecture for enhanced graph classification. *Frontiers in Neuroscience*, 2020.

[40] K. Xu, W. Hu, J. Leskovec, and S. Jegelka. How powerful are graph neural networks? In *ICLR*, 2019.

[41] H. Yao, C. Zhang, Y. Wei, M. Jiang, S. Wang, J. Huang, N. V. Chawla, and Z. Li. Graph few-shot learning via knowledge transfer. In *AAAI*, 2020.

[42] Y. Zhang, Y. Wei, and Q. Yang. Learning to multitask. In *NeurIPS*, 2018.

[43] F. Zhou, C. Cao, K. Zhang, G. Trajcevski, T. Zhong, and J. Geng. Meta-gnn: On few-shot node classification in graph meta-learning. In *CIKM*, 2019.

[44] D. Zügner and S. Günnemann. Adversarial attacks on graph neural networks via meta learning. In *ICLR*, 2019.