A Meta-Learning Approach for Graph Representation Learning in Multi-Task Settings

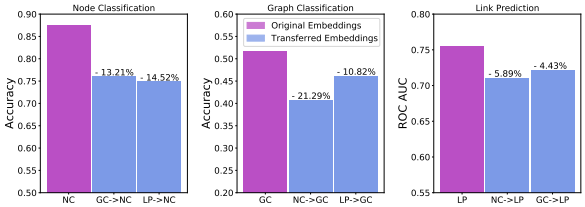
Anonymous Author(s)
Affiliation
Address
email

Abstract

Graph Neural Networks (GNNs) are a framework for *graph representation learning*, where a model learns 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. However, generating node embeddings that can be used to perform multiple tasks (with performance comparable to single-task models) is an open problem. We propose a novel meta-learning strategy capable of producing *multi-task* node embeddings. Our method avoids the difficulties arising when learning to perform multiple tasks *concurrently* by, instead, learning to quickly (i.e. with a few steps of gradient descent) adapt to multiple tasks *singularly*. We show that the embeddings produced by our method can be used to perform multiple tasks with comparable or higher performance than classically trained models. Our method is model-agnostic and task-agnostic, thus applicable to a wide variety of multi-task domains.[1]

1 Introduction

Graph Neural Networks (GNNs) are deep learning models that operate on graph structured data obtaining great empirical performance, and are a very active area of research. Three tasks in particular 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 [10, 4, 31]. The encoder produces node embeddings (low-dimensional vectors capturing 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, giving rise to highly specialized node embeddings. In fact, taking the embeddings from a trained GNN, and using them to train a decoder for a different task, leads to substantial performance loss (see Figure 1).



(a) (b) (c)

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

[1]For the interested reader, Appendix and source code are provided as supplementary material.

Submitted to 34th Conference on Neural Information Processing Systems (NeurIPS 2020). Do not distribute.

The low transferability of node embeddings requires the use of task-specific encoders and decoders. However, many practical machine learning applications operate in resource-constrained environments where being able to share parameters between tasks is of great importance. Learning models that perform multiple tasks is known as *Multi-Task Learning* (MTL), and is an open area of research [29] .

We show that training a multi-head model with the classical procedure, i.e. by performing multiple tasks concurrently on each graph, and updating the parameters with some form of gradient descent to minimize the sum of the single-task losses, can lead to performance loss with respect to single-task models. We then propose a novel optimization-based meta-learning [7] procedure that can generate node embeddings that generalize across tasks. Our meta-learning procedure does not aim at a setting of the parameters that can perform multiple tasks concurrently (like a classical method would do), or to a setting that allows fast multi-task adaptation (like traditional meta-learning), but to a setting that can **easily be adapted to perform each of the tasks singularly**. In fact, our procedure aims at a setting of the parameters where a few steps of gradient descent on a given task, can lead to good performance on that task, hence removing the burden of learning to solve multiple tasks *concurrently*.

We summarize our contributions as follows:

- We propose a novel meta-learning strategy for multi-task representation learning. We apply it on graph MTL, and show that a GNN trained with our method produces higher quality node embeddings with respect to classical training procedures. Our method is *model-agnostic* and *task-agnostic*, thus easily applicable to a wide range of multi-task domains.
- 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. In fact, our embeddings lead to comparable or higher performance with respect to single-task models even when used as input to a simple linear classifier.
- We show that the episodic training strategy in our meta-learning procedure leads to better node embeddings even for single-task models. We believe this finding provides interesting directions for future work on connections between meta-learning and representation learning.

2 Related Work

GNNs, MTL, and meta-learning are very active areas of research. We highlight works that are at the intersections of these subjects, and point the interested reader to comprehensive reviews of each field.

Graph Neural Networks. GNNs have a long history [26], but in the past few years the field has grown exponentially. Seminal works include ChebNet [6], GCN [16], GAT [30], and GIN [32]. For a thorough review of the field we refer the reader to Chami et al. [4] and Wu et al. [31].

Multi-Task Learning. Works at the intersection of MTL and GNNs have focused on multi-head architectures for several applications [22, 12, 32, 2, 17], but no *single* model has been proposed for the three most common tasks on graphs. Other works use GNNs as a tool for MTL: Liu et al. [19] use GNNs to allow communication between tasks, while Zhang et al. [34] use GNNs to estimate the test error of a MTL model. For an exhaustive review of deep MTL we refer to Vandenhende et al. [29].

Meta-Learning. Meta-Learning has attracted considerable attention (see the review by Hospedales et al. [13]), specially in the area of *few-shot learning*. Some works use GNNs directly for few-shot learning [8], others as a tool for enhancing meta-learning [18, 28], and others use meta-learning to train GNNs in few-shot learning scenarios for graph-related problems [35, 33, 15, 15, 1, 3, 24], Other works combining meta-learning and GNNs involve adversarial attacks [36] and active learning [20].

3 Preliminaries

3.1 Graph Neural Networks

Many GNNs follow the *message-passing* paradigm [9]. Let us represent a graph $\mathcal{G} = (\mathbf{A}, \mathbf{X})$ with an adjacency matrix $\mathbf{A} \in \{0,1\}^{n \times n}$, 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 node representation matrix at layer ℓ. A message passing layer updates the representation of every node v as follows:

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

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

3.2 Model-Agnostic Meta-Learning and ANIL

MAML (Model-Agnostic Meta-Learning)[7] is an optimization-based meta-learning strategy. Let f_θ be a deep learning model, where θ are its parameters. Let $p(\mathcal{E})$ be a distribution over episodes[2]. An episode $\mathcal{E}_i \sim p(\mathcal{E})$ is defined as a tuple containing a *loss function* $\mathcal{L}_{\mathcal{E}_i}$, a *support set* $\mathcal{S}_{\mathcal{E}_i}$, and a *target set* $\mathcal{T}_{\mathcal{E}_i}$: $\mathcal{E}_i = (\mathcal{L}_{\mathcal{E}_i}(\cdot), \mathcal{S}_{\mathcal{E}_i}, \mathcal{T}_{\mathcal{E}_i})$ (support and target sets are 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 to allow fast adaptation. Formally, let $\theta'_i(t)$ be the parameters after t adaptation steps on the support set of episode $\mathcal{E}_i$, then the computations in the inner loop are

$$\theta'_i(t) = \theta'_i(t-1) - \alpha \nabla_{\theta'_i(t-1)} \mathcal{L}_{\mathcal{E}_i}(f_{\theta'_i(t-1)}, \mathcal{S}_{\mathcal{E}_i}), \text{ with } \theta'_i(0) = \theta$$

where $\mathcal{L}(f_{\theta'_i(t-1)}, \mathcal{S}_{\mathcal{E}_i})$ indicates the loss over the support set $\mathcal{S}_{\mathcal{E}_i}$ of the model 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 $\mathcal{L}_{\textit{meta}} = \sum_{\mathcal{E}_i \sim p(\mathcal{E})} \mathcal{L}_{\mathcal{E}_i}(f_{\theta'_i(t)}, \mathcal{T}_{\mathcal{E}_i})$, which leads to the following parameter update[3]

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

Raghu et al. [25] 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: 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.

4 Our Method

Our novel representation learning technique, based on meta-learning, is built on three insights:

***(i)* optimization-based meta-learning is implicitly learning robust representations.** The findings by Raghu et al. [25] suggest that in a model trained with MAML, the first layer(s) learn features that are reusable across episodes, while the last layer(s) are set up for fast adaptation. MAML is then *implicitly* focusing on learning reusable representations that generalize across episodes.

***(ii)* meta-learning episodes can be designed to encourage generalization.** If we design support and target set 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*.** We design the meta-learning procedure 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 fast *multiple* **single-task adaptation**. This strategy is searching for a parameter setting that can be easily adapted for good single-task performance, without learning to solve multiple tasks concurrently. (See Appendix A for a comparison with classical training and meta-learning strategies.)

Based on ***(ii)*** and ***(iii)***, we develop a novel meta-learning procedure where the inner loop adapts to multiple tasks *singularly*, each time with the goal of single-task generalization. Using an encoder-decoder architecture, and episodes that involve adapting to multiple tasks, ***(i)*** suggests that this procedure leads to an encoder that learns features that are reusable across episodes (and hence tasks).

Intuition. Training multi-task models is challenging, as tasks may negatively interfere with each other [27]. We design a meta-learning procedure where the learner does not have to find a configuration of

[2]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.

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

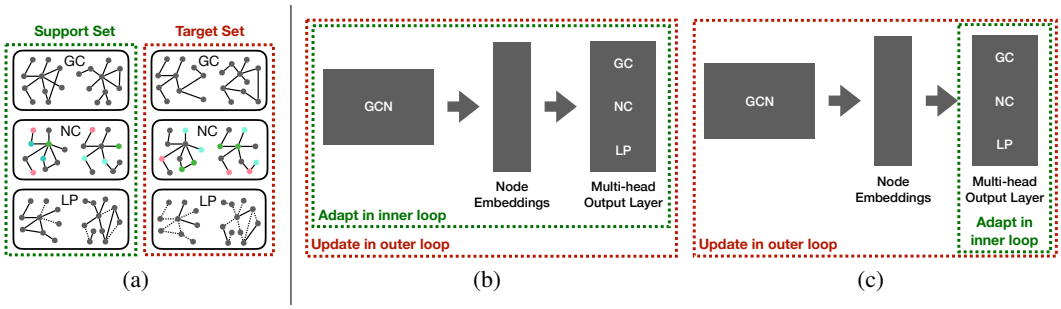


Figure 2: (a) *Multi-task episode*: for each task, support and target sets mimic training and validation sets. (b) iSAME: both backbone and task-specific output layers are adapted (one at a time) in the inner loop. (c) eSAME: only task-specific output layers are adapted (one at a time) in the inner loop.

the parameters that *concurrently* performs all tasks, but a configuration that can **easily be adapted to perform each of the tasks singularly**. Finally, leveraging the robust representation learning that happens with MAML and ANIL, we can extract an encoder generating node representations that generalize across tasks.

We now formally present our novel meta-learning procedure in three steps: **(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.

4.1 Episode Design

In our case, an episode becomes a *multi-task episode* (Figure 2 (a)). 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 $\mathcal{E}_i^{(m)} = (\mathcal{L}_{\mathcal{E}_i}^{(m)}, \mathcal{S}_{\mathcal{E}_i}^{(m)}, \mathcal{T}_{\mathcal{E}_i}^{(m)})$, with

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

where $\lambda^{(\cdot)}$ are balancing coefficients. The meta-objective of our method then becomes:

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

Support and target sets are set up to resemble a training and a validation set. Therefore the outer loop's objective becomes to *maximize the performance on a validation set, given a training set*, hence pushing towards generalization (additional details are provided in Appendix B).

4.2 Model Architecture Design

We use an encoder-decoder model with a multi-head architecture. The *backbone* (which represents the encoder) is a 3 layer GCN [16], while the decoder is composed of three *heads* (one per task) with standard architectures. For additional information we refer the interested reader to Appendix C.

4.3 Meta-Training Design

To avoid the problems arising from training a model that performs multiple tasks concurrently, we design a meta-learning procedure where the inner loop adaptation and the meta-objective computation involve 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 that allows *multiple* **fast single-task adaptation**. The pseudocode of our procedure is in Algorithm 1. `ADAPT` performs a few steps of gradient descent on a task specific loss function and support set, `TEST` computes the value of a meta-objective component on a task specific loss function and target set, and `UPDATE` optimizes the parameters by minimizing the meta-objective. Notice how the multiple *heads* of the decoder in our model are never used concurrently.

Table 1: Results for a single-task model trained in a classical supervised manner (Cl), and a **linear** classifier trained on the embeddings produced by our meta-learning strategies (iSAME, eSAME).

Task	Model	Dataset			
		ENZYMES	PROTEINS	DHFR	COX2
NC	Cl	87.5 ± 1.9	72.3 ± 4.4	97.3 ± 0.2	96.4 ± 0.3
	iSAME	87.3 ± 0.8	81.8 ± 1.6	96.6 ± 0.3	96.1 ± 0.4
	eSAME	87.8 ± 0.7	82.4 ± 1.6	96.8 ± 0.2	96.5 ± 0.6
GC	Cl	51.6 ± 4.2	73.3 ± 3.6	71.5 ± 2.3	76.7 ± 4.7
	iSAME	50.8 ± 2.9	73.5 ± 1.2	73.2 ± 3.2	76.3 ± 4.6
	eSAME	52.1 ± 5.0	72.6 ± 1.6	71.6 ± 2.4	75.6 ± 4.1
LP	Cl	75.5 ± 3.0	85.6 ± 0.8	98.8 ± 0.7	98.3 ± 0.8
	iSAME	81.7 ± 1.7	84.0 ± 1.1	99.2 ± 0.4	99.1 ± 0.5
	eSAME	80.1 ± 3.4	84.1 ± 0.9	99.2 ± 0.3	99.2 ± 0.7

Let us partition the parameters θ of our model in four sets: one representing the backbone (θ_{GCN}), and one for each head ($\theta_{NC}, \theta_{GC}, \theta_{LP}$). We name our meta-learning strategy SAME (Single-Task Adaptation for Multi-Task Embeddings), and present two variants (Figure 2 (b)-(c)): *implicit* SAME (iSAME), and *explicit* SAME (eSAME). In iSAME all the parameters θ are used for adaptation. iSAME makes use of the *implicit* feature-reuse factor of MAML, leading to parameters θ_{GCN} that are general across *multi-task episodes*. In eSAME only the head parameters $\theta_{\text{NC}}, \theta_{\text{GC}}, \theta_{\text{LP}}$ are used for adaptation. eSAME *explicitly* aims at parameters θ_{GCN} that are general across *multi-task episodes* by only updating them in the outer loop.

Algorithm 1: Proposed Meta-Learning Procedure

Input : Model f_θ; Episodes $\mathcal{E} = \{\mathcal{E}_1, .., \mathcal{E}_n\}$
`init`(θ)
for $\mathcal{E}_i$ *in* $\mathcal{E}$ **do**
  `o_loss` $\leftarrow 0$
  for t *in (GC, NC, LP)* **do**
    $\theta'^{(t)} \leftarrow \theta$
    $\theta'^{(t)} \leftarrow$ `ADAPT`$(f_\theta, \mathcal{S}^{(t)}_{\mathcal{E}_i}, \mathcal{L}^{(t)}_{\mathcal{E}_i})$
    `o_loss` $\leftarrow$ `o_loss` + `TEST`$(f_{\theta'^{(t)}}, \mathcal{T}^{(t)}_{\mathcal{E}_i}, \mathcal{L}^{(t)}_{\mathcal{E}_i})$
  end
  $\theta \leftarrow$ `UPDATE`$(\theta,$ `o_loss`$, \theta'^{(GC)}, \theta'^{(NC)}, \theta'^{(LP)})$
end

5 Experiments

Our goal is to assess the quality of the representations learned by our proposed method by answering four questions (**Q1**-**Q4**). We use GC to refer to graph classification, NC for node classification, and LP for link prediction. Unless otherwise stated, accuracy (%) is used for NC and GC, while ROC AUC (%) is used for LP.

Experimental Setup. We consider datasets from the TUDataset library [23] that allow multi-task settings, and perform a 10-fold cross validation. To ensure a fair comparison, we use the same architecture for all training strategies. For more information we refer to Appendix D.

Q1: ***Do iSAME and eSAME lead to high quality node embeddings in the single-task setting?*** For every task, we train a **linear classifier** on top of the embeddings produced by a model trained using our proposed methods, and compare against a network with the same architecture trained in a classical manner. Results are shown in Table 1. For all tasks, the **linear** classifier achieves comparable, if not superior, performance to the 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%.

Q2: ***Do iSAME and eSAME lead to high quality node embeddings in the multi-task setting?*** We train a model with our proposed methods, on all multi-task combinations, and use the embeddings as input for a **linear classifier**. We compare against single-task models trained in the classical manner. We notice that the **linear** classifier, achieves comparable performance to the end-to-end models, as it is never outperformed by more than 3%, and in 50% of the cases it actually performs better (see Appendix E), confirming the high quality of the node embeddings learned with iSAME and eSAME.

Table 2: Δ_m (%) results for a classical multi-task model (Cl), and a **linear** classifier trained on the node embeddings learned using our meta-learning strategies (iSAME, eSAME).

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
✓	✓		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
✓		✓	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
	✓	✓	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

Q3: ***Do iSAME and eSAME extract information that is not captured by classically trained multi-task models?*** We train a network, which we refer to as *classifier*, on the embeddings generated by a multi-task model, to perform a task that was not seen during the training of the latter. We compare the performance of the classifier on the embeddings learned by a model trained in a classical manner, and with our proposed methods. This test allows us to quantify if our approaches lead to "more informative" node embeddings. Results on the ENZYMES dataset are shown in Figure 3. We notice that embeddings learned by our proposed approaches lead to at least 10% higher performance. We observe an analogous trend on the other datasets (as reported in Appendix F).

Q4: ***Can the node embeddings learned by iSAME and eSAME be used to perform multiple tasks with comparable or better performance than classical multi-task models?*** We train the same multi-task model, both in the classical supervised manner, and with our proposed approaches, on all multi-task combinations. For our approaches, we then train a **linear classifier** on top of the node embeddings. We use the Δ_m metric [21] (average per-task drop with respect to the single-task baseline). Results are shown in Table 2. We first notice that usually multi-task models achieve lower performance than specialized single-task ones. We then highlight that **linear** classifiers trained on the embeddings produced by our procedures are comparable, and in many cases superior, to end-to-end models. In fact, the latter are highly sensible to the tasks that are being learned (e.g. GC and LP), with a worst-case average drop in performance of 29%. Our methods seem much less sensible, with a worst-case average drop of less than 3%.

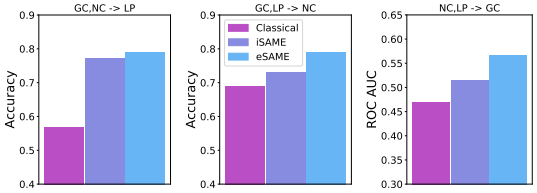


Figure 3: Results for neural network, trained on the embeddings generated by a multi-task model, performing a task that was not seen by the multi-task model. "x, y->z" indicates that x, y are the tasks for training the multi-task model, and z is the new task.

6 Conclusions

We propose a novel meta-learning strategy for representation learning in multi-task settings. Our method overcomes the problems that arise when learning to solve multiple tasks concurrently by optimizing for a parameter setting that can quickly, i.e. with few steps of gradient descent, be adapted for high *single-task* performance on multiple tasks. We apply our method to graph representation learning, and find that it leads to higher quality node embeddings, both in the multi-task and in the single-task setting. We believe this work draws new interesting connections between meta-learning, representation learning, and multi-task learning, providing many directions for future research.

References

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

[2] Pedro Avelar, Henrique Lemos, Marcelo Prates, and Luis Lamb. Multitask learning on graph neural networks: Learning multiple graph centrality measures with a unified network. In *Artificial Neural Networks and Machine Learning – ICANN 2019: Workshop and Special Sessions*, pages 701–715. Springer International Publishing, 2019. doi: 10.1007/978-3-030-30493-5_63. URL `https://doi.org/10.1007%2F978-3-030-30493-5_63`.

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

[4] Ines Chami, Sami Abu-El-Haija, Bryan Perozzi, Christopher Ré, and K. Murphy. Machine learning on graphs: A model and comprehensive taxonomy. *ArXiv*, abs/2005.03675, 2020.

[5] Zhao Chen, Vijay Badrinarayanan, Chen-Yu Lee, and Andrew Rabinovich. Gradnorm: Gradient normalization for adaptive loss balancing in deep multitask networks. In *ICML*, 2018.

[6] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems*, 2016. URL `https://arxiv.org/abs/1606.09375`.

[7] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, pages 1126–1135. JMLR.org, 2017.

[8] Victor Garcia and Joan Bruna. Few-shot learning with graph neural networks. *ICLR*, 2018.

[9] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, pages 1263–1272. JMLR.org, 2017.

[10] William L Hamilton, Rex Ying, and Jure Leskovec. Representation learning on graphs: Methods and applications. *IEEE Data Engineering Bulletin*, 2017.

[11] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[12] Chester Holtz, Onur Atan, Ryan Carey, and Tushit Jain. Multi-task learning on graphs with node and graph level labels. *NeurIPS Workshop on Graph Representation Learning*, 2019.

[13] Timothy Hospedales, Antreas Antoniou, Paul Micaelli, and Amos Storkey. Meta-learning in neural networks: A survey, 2020.

[14] Alex Kendall, Yarin Gal, and Roberto Cipolla. Multi-task learning using uncertainty to weigh losses for scene geometry and semantics. In *Conference on Computer Vision and Pattern Recognition*, pages 7482–7491, 2018.

[15] Jongmin Kim, Taesup Kim, S. Kim, and C. Yoo. Edge-labeling graph neural network for few-shot learning. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11–20, 2019.

[16] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)*, 2017.

[17] Diya Li and Heng Ji. Syntax-aware multi-task graph convolutional networks for biomedical relation extraction. In *Proceedings of the Tenth International Workshop on Health Text Mining and Information Analysis (LOUHI 2019)*, pages 28–33, Hong Kong, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-6204. URL `https://www.aclweb.org/anthology/D19-6204.pdf`.

[18] Lu Liu, Tianyi Zhou, Guodong Long, Jing Jiang, and Chengqi Zhang. Learning to propagate for graph meta-learning. In *Neural Information Processing Systems (NeurIPS)*, 2019.

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

[20] Kaushalya Madhawa and Tsuyoshi Murata. Active learning on graphs via meta learning. *ICML Workshop on Graph Representation Learning and Beyond, ICML*, 2020.

[21] K. Maninis, I. Radosavovic, and I. Kokkinos. Attentive single-tasking of multiple tasks. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1851–1860, 2019.

[22] Floriane Montanari, Lara Kuhnke, Antonius Ter Laak, and Djork-Arné Clevert. Modeling physico-chemical ADMET endpoints with multitask graph convolutional networks. *Molecules*, 25(1):44, dec 2019. doi: 10.3390/molecules25010044. URL `https://doi.org/10.3390%2Fmolecules25010044`.

[23] Christopher Morris, Nils M. Kriege, Franka Bause, Kristian Kersting, Petra Mutzel, and Marion Neumann. Tudataset: A collection of benchmark datasets for learning with graphs. In *ICML 2020 Workshop on Graph Representation Learning and Beyond (GRL+ 2020)*, 2020. URL `www.graphlearning.io`.

[24] Cuong Q. Nguyen, Constantine Kreatsoulas, and Branson Kim M. Meta-learning gnn initializations for low-resource molecular property prediction. *ICML Workshop on Graph Representation Learning and Beyond, ICML*, 2020.

[25] Aniruddh Raghu, Maithra Raghu, Samy Bengio, and Oriol Vinyals. Rapid learning or feature reuse? towards understanding the effectiveness of maml. *ICLR*, 2020.

[26] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.

[27] Trevor Standley, Amir R. Zamir, Dawn Chen, Leonidas Guibas, Jitendra Malik, and Silvio Savarese. Which tasks should be learned together in multi-task learning? In *ICML*, 2020.

[28] Qiuling Suo, Jingyuan Chou, Weida Zhong, and Aidong Zhang. Tadanet: Task-adaptive network for graph-enriched meta-learning. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, page 1789–1799. Association for Computing Machinery, 2020. ISBN 9781450379984. doi: 10.1145/3394486.3403230. URL `https://doi.org/10.1145/3394486.3403230`.

[29] Simon Vandenhende, Stamatios Georgoulis, Marc Proesmans, Dengxin Dai, and Luc Van Gool. Revisiting multi-task learning in the deep learning era, 2020.

[30] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks. *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=rJXMpikCZ`.

[31] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–21, 2020. doi: 10.1109/tnnls.2020.2978386. URL `https://doi.org/10.1109%2Ftnnls.2020.2978386`.

[32] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=ryGs6iA5Km`.

[33] Huaxiu Yao, Chuxu Zhang, Ying Wei, Meng Jiang, Suhang Wang, Junzhou Huang, Nitesh V. Chawla, and Zhenhui Li. Graph few-shot learning via knowledge transfer. In *AAAI*, 2020.

[34] Yu Zhang, Ying Wei, and Qiang Yang. Learning to multitask. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, page 5776–5787, Red Hook, NY, USA, 2018. Curran Associates Inc.

[35] 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 *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, CIKM '19, page 2357–2360, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450369763. doi: 10.1145/3357384.3358106. URL https://doi.org/10.1145/3357384.3358106.

[36] Daniel Zügner and Stephan Günnemann. Adversarial attacks on graph neural networks via meta learning. In *International Conference on Learning Representations (ICLR)*, 2019.