Small Molecule Property Prediction via Proxy Labeling and Multi-scale Learning

Gautam Machiraju
Department of Biomedical Data Science
Stanford University
Stanford, CA 94305
gmachi@stanford.edu

Abstract

Predicting molecular functions or properties is a challenging task and largely studied in the drug development communities. This work presents a novel multi-scale learning approach to Graph Neural Networks that leverages weak supervision support from de facto or proxy node-level labels. Preliminary results show promising results despite the challenges of multi-task optimization. Further study of architectural design, optimization strategy, and hyperparameter tuning lends confidence toward state-of-the-art performance.

1 Introduction

Molecular property and functional prediction is one of the biggest challenges in the bio-scientific community due to the numerous applications in *in silico* drug design, as well as the enormous implications from an improved understanding of the fundamental structure-function relationship of molecules in basic science. In order to featurize molecules for machine learning tasks, graphical representations have been used to represent atoms as nodes and molecules as graphs, where graph edges in turn represent bonds between atoms.

Because of the increased work in graph-level predictions in the machine learning community, Graph Neural Networks (GNNs) have more recently been expanding into the molecular property prediction. Because of the limited information carried by graph-level labels, multiple efforts have been made to further support the supervision of graph-level predictions. Because of this multi-task Learning (MTL) approaches have become increasingly popular for encoding richer node-level embeddings. This work explores this space of Multi-scale Learning (MSL), which leverages prediction tasks at multiple levels of structured data in this goal of richer embeddings.

1.1 Related Work

Graph Neural Networks In the last few years, deep learning for applications in networks has led to the creation of multiple Graph Neural Network (GNN) architectures. Using the shared Message Passing formulation and framework [1], multiple GNN architectures have come into the spotlight with superior performance in classification and regression tasks: Graph Convolutional Networks (GCNs) [2], GCN+Virtual Nodes [2], Graph Isomorphism Networks (GINs) [3], GIN+Virtual Nodes [3] are all conveniently provided as baselines by the Open Graph Benchmark (OGB) [4].

Multi-task Learning Multi-task Learning (MTL) is a relatively new paradigm and area of research, especially as it applies to graph neural networks. MTL is often used in practice as a technique to equip models with more robust and flexible training, and to ultimately predict on a wide suite of potential tasks. To this effect of model generalizability, it is also a viable form of regularization

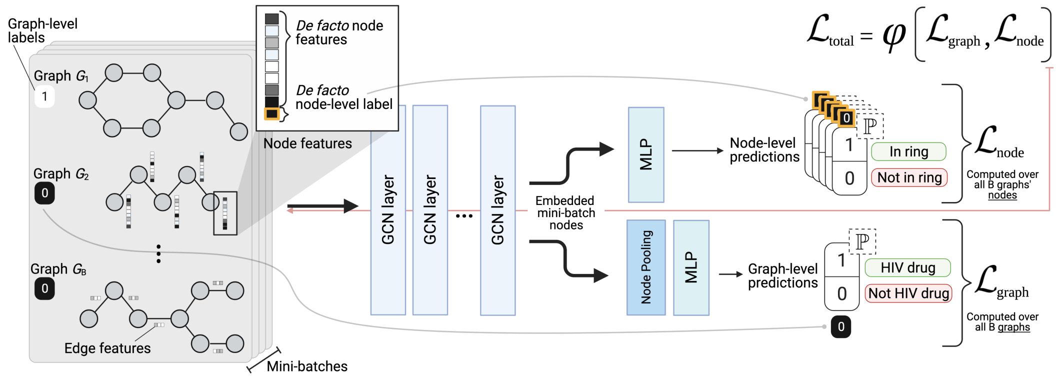
Graph-level labels
Graph G_1
1
Graph G_2
0
Graph G_B
0
Edge features
Mini-batches
De facto node features
De facto node-level label
Node features
GCN layer
GCN layer
GCN layer
Embedded mini-batch nodes
MLP
Node Pooling
MLP
Node-level predictions
Graph-level predictions
1
0
In ring
Not in ring
HIV drug
Not HIV drug
$\mathbb{P}$
$\mathcal{L}_{\text{node}}$
Computed over all B graphs' nodes
$\mathcal{L}_{\text{graph}}$
Computed over all B graphs
$\mathcal{L}_{\text{total}} = \varphi\left[\mathcal{L}_{\text{graph}}, \mathcal{L}_{\text{node}}\right]$

Figure 1: Model architecture. Mini-batches of graphs (of size B) are fed into our model. After the multiple GNN layers (GCN in our case), the model makes node-level and graph-level predictions, which are then jointly optimized as $\mathcal{L}_{\text{total}}$. The red arrow represents this joint optimization through back-propagation. This graphical abstracts provides a depiction of our specific dataset use case, but can be extended to any dataset without loss of generality. Shared parameters Θ in this work are those arising from the GCN layers.

to avoid overfitting to any task. Lastly, MTL designs can also help with small datasets due to the supportive weak supervision other tasks may provide.

In the single-task scenario, let us suppose a dataset $\mathcal{D} = \{(\mathbf{x}, \mathbf{y})_n\}$ is generated with data-generating distributions $p(\mathbf{x}), p(\mathbf{y}|\mathbf{x})$ for training inputs $\mathbf{x}$ and labels $\mathbf{y}$, respectively. Suppose also that we aim to fit a Θ-parameterized function $f_\Theta(\mathbf{y}|\mathbf{x})$ by minimizing some loss function $\mathcal{L}$: $\min_\Theta \mathcal{L}(\Theta, \mathcal{D})$. Then, a task $\mathcal{T}$ is simply a set of the aforementioned data-generating distributions and its loss function: $\mathcal{T} = \{p(\mathbf{x}), p(\mathbf{y}|\mathbf{x}), \mathcal{L}\}$. The multi-task scenario simply extends this definition to K number of tasks $\mathcal{T}_1, \ldots, \mathcal{T}_K$, where each task $\mathcal{T}_i = \{p_i(\mathbf{x}), p_i(\mathbf{y}|\mathbf{x}), \mathcal{L}_i\}$ has its own associated dataset $\mathcal{D}_i$ (and associated data-generating functions) and loss $\mathcal{L}_i$. As one may envision, the architectural design space of model f_Θ can be vast given all the tasks to consider. For the purposes of this work, we consider Θ the shared set of parameters for all tasks $\mathcal{T}_i \; \forall i = 1 \ldots K$. This is referred to as *hard parameter sharing* [5] and is assumed for the models in this work.

Multi-scale Learning Multi-scale Learning (MSL), sometimes also referred to as Hierarchical Learning, is a proposed regime to learn embeddings and representations that operate on multiple scales of structured data (e.g. both nodes and graphs, both patches and images, etc.) for downstream prediction tasks. The hope of such a regime is to allow for more flexible learning for performing simultaneous tasks, model robustness, and model generalizability to other datasets. An increasingly popular approach to accomplishing this form of learning is through MTL frameworks, where each task is operating at the different data scales. Recent work by Holtz et al [6] explored models that jointly optimized predictions on both the node- and graph-level. The hope with such approaches is that the smaller-scale predictions act as weak supervision for the larger-scale predictions. Successful performance in their experimentation served as inspiration for the work in this paper.

Molecular Function Prediction In recent years, the drug development community has leveraged GNNs for molecular property or functional prediction. Feinberg et al [7] introduced a family of graph convolutions specifically designed for learning protein-ligand binding affinities. Capela et al [8] used a MTL approach to learn multiple graph- or molecule-level targets for more general-use node and graph embeddings. Finally, as discussed previously, Holtz et al [6] use atom-level (i.e. node-level) and molecule-level (i.e. graph-level) features to perform MSL and ultimately outperform many molecular function prediction baselines on larger molecule datasets.

1.2 Contribution

This work builds heavily on the recent work by Holtz et al [6] to design Multi-scale Learning approaches for molecular function prediction. More precisely, our contributions in this work builds off of these ideas by:

- Using proxy node-level labels — in the absence of *a priori* node-level labels — as a form of weak supervision support for the main task of graph classification
- Forgoing the use of computationally intensive approaches to combine loss terms such as GradNorm
- Showing promising results despite model simplicity
- Testing the Multi-scale Learning paradigm for the first time on OGB's hosted `ogbg-molhiv` dataset of small molecules; **such a model design is not currently on the leaderboard**, and to our knowledge has not yet been deployed on a small molecule dataset of this kind

2 Proposed Method

With the goal of creating rich node embeddings that are meaningful at multiple scales, we employ a Multi-scale Learning (MSL) approach through the use of a *de facto* or *proxy* node-level labels. Our approach trains classifiers to perform $K = 2$ tasks: (1) a *graph-level task* $\mathcal{T}_{\text{graph}}$, which is a binary classification of input graph labels (per mini-batch); and (2) a *node-level task* $\mathcal{T}_{\text{node}}$, which is a binary classification of node-level proxy labels (on all the nodes within the mini-batch's graphs). We accomplish this by jointly optimizing our target function f_Θ both tasks.

2.1 Preliminaries

We define our inputs $\mathbf{x}^{(i)}$ as graphs $G_i(V_i, E_i)\ \forall i = 1 \ldots n$, which contain sets of vertices or nodes (V) and edges (E). For expressivity, we represent each input as $\mathbf{x}^{(i)} = \{\mathcal{A}^{(i)}, \mathcal{V}^{(i)}, \mathcal{E}^{(i)}\}$, or a set of its corresponding adjacency matrix ($\mathcal{A}^{(i)}$), node-level feature matrix ($\mathcal{V}^{(i)}$), and edge-level feature matrix ($\mathcal{E}^{(i)}$). Graph-level labels are represented by $\mathbf{y}^{(i)}_{\text{graph}}$ and are labeled *a priori*. In our approach, we simply define our proxy node-level label $\mathbf{y}^{(i)}_{\text{node}j} = \mathcal{V}^{(i)}[*, j]$, where j indicates the jth node in the ith graph, and is node-featurized by the jth column in $\mathcal{V}^{(i)}$. Note also that the $*$ character indicates the user-defined proxy label extracted from *a priori* node-level input features.

2.2 Architecture

The Architecture in this work depends on (1) proxy node-level labels, (2) GCN layers, (3) Node Pooling strategy, and (4) Loss combinations. Each is described in detail below.

Proxy node labels As described in the definition above, we choose an *a priori* defined node-level feature as our proxy node-level label or prediction target. The hope is to accomplish weak supervision support for the main task of graph-level classification. This choice of node-level proxy label (in our case, ring membership of an atom, or node feature (9)) was largely based on (a) its binary nature for ease of prediction and (b) its potential to enforce learning local structure within the molecule — thereby in line with the goals of MSL approaches.

Graph Convolutional Networks As the workhorse model, we use a Graph Convolutional Network (GCN) [2]. We chose this implementation again for its simplicity and to understand how much a MSL approach would boost its baseline performance.

Node Pooling Instead of the SAGPool layer performed by Holtz et al, we perform mean-pooling for simplicity. This was specifically chosen for more straightforward and modular comparison with OGB's baselines.

2.3 Loss Functions

While our reference model [6] uses Gradient Normalization (i.e. GradNorm) [9] (and its associated use of higher-order automated differentiation) to linearly combine loss terms, we instead take a simpler approach to learn multiple convex combinations of our losses. This was design choice was largely pursued due to the computational burdens of higher-order computations needed for GradNorm (e.g. training on multiple GPUs).

To perform multi-scale learning, we define multiple convex combinations or functions, ψ, to combine loss terms from the multiple scales (i.e. $\mathcal{L}_{\text{graph}}$ and $\mathcal{L}_{\text{node}}$). This results in $\mathcal{L}_{\text{total}}$, which is optimized for both tasks. In **Figure 1**, ψ is shown applied to both constituent losses and back-propagated over (as shown by the red backward line).

Unweighted Additive Loss For our simplest approach, we perform unweighted addition of our constituent loss terms. While this simplified and naive approach to multi-task learning has been shown to lead to noisy training and unstable convergence [10], this method was kept in our study as a baseline model. The jointly optimized $\mathcal{L}_{\text{total}}$ is thus simply just expressed as:

$$\mathcal{L}_{\text{total}} = \psi(\mathcal{L}_{\text{graph}}, \mathcal{L}_{\text{node}}) = \mathcal{L}_{\text{graph}} + \mathcal{L}_{\text{node}}. \tag{1}$$

Weighted Additive Loss The next loss combination examined is a linear combination of weights (w_i) applied to our losses. It has also been shown that using fixed weights w_i, which act as hyper-parameters, leads to extreme model performance sensitivity [10]. We instead use the more flexible approach of defining the weights as learnable parameters for the optimization procedure, which should allow us to observe the near-optimal weights over the searched parameter space and iterated epochs. This combination of losses mirrors Holtz et al's work [6], albeit without gradient normalization [9] and its associated computational costs:

$$\mathcal{L}_{\text{total}} = \psi(\mathcal{L}_{\text{graph}}, \mathcal{L}_{\text{node}}) = [\mathcal{L}_{\text{graph}}, \mathcal{L}_{\text{node}}]^{\top}[w_1, w_2] = w_1\mathcal{L}_{\text{graph}} + w_2\mathcal{L}_{\text{node}} \tag{2}$$

Uncertainty Loss Further model complexity is explored through the use of nonlinear convex combinations of loss terms. Working off recent work in computer vision and scene semantics, we use the Uncertainty Loss proposed by Kendall et al [10]. This loss is fundamentally working off of a Maximum Likelihood estimator of all model task ($\mathcal{T}_1, \ldots, \mathcal{T}_K$) outputs $\mathbf{y}_1, \ldots, \mathbf{y}_K$ given our neural network model output $f_{\mathbf{\Theta}}(\mathbf{x})$ with weights $\mathbf{\Theta}$ and input $\mathbf{x}$. With our two tasks and associated loss terms, this Uncertainty Loss formulation results as:

$$\begin{aligned} \mathcal{L}_{\text{total}} &= \psi(\mathcal{L}_{\text{graph}}, \mathcal{L}_{\text{node}}) = [\mathcal{L}_{\text{graph}}, \mathcal{L}_{\text{node}}] \odot \exp[-w_1, -w_2] + [w_1, w_2] & (3) \\ &= e^{-w_1}\mathcal{L}_{\text{graph}} + e^{-w_2}\mathcal{L}_{\text{node}} + w_1 + w_2 & (4) \end{aligned}$$

where $w_i = 2\log(\sigma_i)$ represents a change of variable for the model's observation noise parameter σ_i derived from the Gaussian likelihood function. Practically, we simply parameterize w_i as learnable weights for the model.

3 Experiments

All experiments were run on Google Colab's GPU resources provided by Stanford CS 224W. Code was developed using the Pytorch and Pytorch-Geometric packages, adapting OGB's baseline models [4]. All model variants were trained for 100 epochs. Due to the noisiness of tasks, multiple optimizers were used to forgo detrimental momentum terms. As a result, RMSProp, Adam, and AdamW were compared with the different loss functions described in the previous section. Additionally, batch sizes (B) were varied per experiment to observe batch stochasticity in model training. Unfortunately due to computational and logistical limitations, a grid search was not performed on the aforementioned model and hyperparameter configurations.

3.1 Dataset

The above specified architecture is applied to the OGB's `ogbg-molhiv` dataset. This dataset, originally adapted from MoleculeNet datasets, contains 41,127 small molecules represented as graphs — where nodes represent atoms and edges represent bonds. Each edge is featurized with 3 features: bond type, stereochemistry, and conjugation. The full featurization of each node is 9-dimensional: (1) atomic number, (2) chirality, (3) degree or number of bonds, (4) formal charge, (5) number of hydrogen bonds, (6) number of free radical electrons, (7) hybridization, (8) aromiticity, and (9) ring membership. More dataset details are outlined in the OGB paper [4] and its associated website.

For the purposes of our model, we remove the last node feature, (9) ring membership, to use as our *de facto* or *proxy* labels in the node-level predictions of the multi-scale setup. This use of the final feature for node-level predictions can be seen in **Figure 1**.

Table 1: Model performance. Configurations are listed by major model specifications: (1) loss type, (2) number of GCN layers (L), (3) number of epochs (E), (4) batch size (B), (5) learning rate (α), and (6) optimizer.

ID	Model configuration	Best validation score	Test score
I	Unweighted-L5-E100-B32-α1e-3-Adam	**0.8283**	0.7448
II	Unweighted-L5-E100-B32-α5e-4-RMSProp	0.8101	**0.7619**
III	Unweighted-L5-E100-B32-α1e-3-AdamW	0.8162	0.7597
IV	Unweighted-L7-E120-B64-α5e-4-RMSProp	0.8025	0.7568
V	Uncertainty-L5-E100-B64-α5e-4-Adam	0.6032	0.5195

3.2 Results & Discussion

Table 1 shows model performance on the models applied to the above dataset. **General performance of Unweigthed Additive Loss, despite lacking rigorous hyperparameter tuning, matches that of results on OGB's leaderboard.** Further testing to create confidence intervals on test-set performance would need to be conducted to verify leaderboard placement.

However, one can quickly notice the strikingly poor performance of the Uncertainty Loss run. Because of poor training runs for both the Uncertainty and Weighted Additive Losses in our work's experimentation (seemingly due to struggling to leave local minima), coupled with logistical constraints, many runs were aborted early. Nevertheless, we show the most promising run of the Uncertainty Loss configuration (Model Configuration V), which is far below the simple Unweighted Loss Additive Loss. While the Uncertainty Loss has been met with promising results in computer vision applications, it did not lead to successful model training in our experimentation — despite manually varying hyperparameters such as batch size, optimizer, learning rate, etc. It is still unclear as to why this is the case since the Maximum Likelihood formulation of the Uncertainty Loss is data agnostic.

We posit superior MSL model performance in large molecule functional prediction. Because of an increased number of nodes (i.e. atoms) per graph (i.e. molecule), as well as more complex and diverse graph topologies in a large molecule dataset, we would expect these richer node embeddings to be more informative for the graph-level classification. It should also be considered if the tasks within this study are similar enough to leverage joint learning — the performance of MTL approaches has been shown to be highly dependent on *task similarity*, which as a concept is still relatively non-rigorous and is an area of active research. It empirically has been made clear, however, that without high task similarity, the evaluation of a single task can show sub-par results [11].

4 Conclusions & Future Work

Future work would employ other node-level labels such as aromiticity, or node feature (8). Additionally, investigations will be carried out to understand the failings of the Uncertainty Loss in this application despite all its success in computer vision. Other weighted loss functions will be explored to smoothen gradient steps, including weighting by learning speed and performance [5]. Finally, after the choice and tuning of loss functions, this work will further explore hyperparameter tuning (e.g. learning rate, optimizer, batch size, pooling strategy, etc.) through grid searches. In particular, the many ideas on pooling strategy [12] would serve as quick design changes. Future experimentation would be explored on OGB's larger hosted molecular function prediction dataset, `ogbg-molpcba`. Ultimately the MSL approach would be extended to all off-the-shelf GNN layer architectures such as GIN and associated virtual node implementations.

We would like to acknowledge and thank the CS 224W teaching team for support and computational resources provided for this work.

References

[1] William Hamilton et al. Representation Learning on Graphs: Methods and Applications. *ArXiv*, 2018.

[2] Thomas Kipf et al. Semi-supervised Classification with Graph Convolutional Networks. *International Conference on Learning Representations (ICLR)*, 2017.

[3] Keyulu Xu et al. How Powerful are Graph Neural Networks? *ArXiv*, 2018.

[4] Weihua Hu et al. Open Graph Benchmark: Datasets for Machine Learning on Graphs. *Neural Information Processing Systems (NeurIPS)*, 2020.

[5] Michael Crawshaw. Multi-task Learning with Deep Neural Networks: A Survey. *ArXiv*, 2020.

[6] Chester Holtz et al. Multi-Task Learning on Graphs with Node and Graph Level Labels. *Neural Information Processing Systems (NeurIPS)*, 2019.

[7] Evan Feinberg et al. PotentialNet for Molecular Property Prediction. *ACS Cent. Sci.*, 2018.

[8] Fabio Capela et al. Multitask Learning On Graph Neural Networks Applied To Molecular Property Predictions. *ArXiv*, 2019.

[9] Zhao Chen et al. GradNorm: Gradient Normalization for Adaptive Loss Balancing in Deep Multitask Networks. *International Conference on Machine Learning (ICML)*, 2018.

[10] Alex Kendall et al. Multi-Task Learning Using Uncertainty to Weigh Losses for Scene Geometry and Semantics. *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[11] Trevor Standley et al. Which Tasks Should Be Learned Together in Multi-task Learning? *International Conference on Machine Learning (ICML)*, 2020.

[12] Zonghan Wu et al. A Comprehensive Survey on Graph Neural Networks. *ArXiv*, 2019.