SMGRL: A Scalable Multi-resolution Graph Representation Learning Framework

Reza Namazi 1  Elahe Ghalebi 2  Sinead Williamson 3  Hamidreza Mahyar 4

Abstract

Graph convolutional networks (GCNs) allow us to learn topologically-aware node embeddings, which can be useful for classification or link prediction. However, by construction, they lack positional awareness and are unable to capture long-range dependencies without adding additional layers—which in turn leads to over-smoothing and increased time and space complexity. Further, the complex dependencies between nodes make mini-batching challenging, limiting their applicability to large graphs.

This paper proposes a Scalable Multi-resolution Graph Representation Learning (SMGRL) framework that enables us to learn multi-resolution node embeddings efficiently. Our framework is model-agnostic and can be applied to any existing GCN model. We dramatically reduce training costs by training only on a reduced-dimension coarsening of the original graph, then exploit self-similarity to apply the resulting algorithm at multiple resolutions. Inference of these multi-resolution embeddings can be distributed across multiple machines to reduce computational and memory requirements further. The resulting multi-resolution embeddings can be aggregated to yield high-quality node embeddings that capture both long- and short-range dependencies between nodes. Our experiments show that this leads to improved classification accuracy, without incurring high computational costs.

1. Introduction

When working with graph-structured data, we often wish to learn latent vector representations for nodes within the graph—often referred to as node embeddings. These representations, which typically aim to capture the topological structure of the graph, can be used for tasks such as node classification and link prediction, or can be combined to obtain a representation of the entire graph. Message passing algorithms, where a node's embedding is updated based on its neighbors' embeddings, are a natural way to capture topological structure. Graph convolutional networks (GCNs, Kipf & Welling, 2017; Hamilton et al., 2017; Veličković et al., 2018) learn the form of these updates using neural networks. In addition to allowing for flexible update rules, this allows GCNs to operate in an inductive fashion, since the message passing algorithm can be applied to new graphs. This gives them a clear advantage over transductive algorithms such as node2vec (Grover & Leskovec, 2016) or DeepWalk (Perozzi et al., 2014), which learn a one-to-one mapping from node to embedding, often based on random walks over the graph, and cannot generalize to nodes not appearing in the training graph.

GCNs have achieved impressive performance on many graph-based classification, prediction, and simulation tasks (Klicpera et al., 2019; Bianchi et al., 2021; Brockschmidt, 2020). However, there are limits to their representational power. By aggregating information from its immediate neighborhood, a single-layer GCN ignores any long-range dependencies across the graph. This can be addressed by adding more layers—a \( K \)-layer GCN incorporates information from a node's \( K \)-hop neighborhood. In practice, however, increasing the number of layers is often accompanied by a decrease in performance. As the reach of the message passing algorithm expands, the inferred representations become increasingly homogeneous—a phenomenon known as oversmoothing (Li et al., 2016; Oono & Suzuki, 2019; Cai & Wang, 2020).

In addition, the computational and memory requirements of GCNs scale poorly when compared with standard neural networks. Memory requirements are inflated due to the connectivity pattern of the graph: the representation of a node relies on all the nodes in its \( K \)-hop neighborhood. This means that, when minibatching, we must store in memory not just the set of nodes selected for the minibatch, but also their \( K \)-hop neighborhood (or an appropriately informative subset thereof).
One way to reduce the computational and memory requirements is to train the GCN on a smaller graph that is “similar” to the original graph. Loukas (2019) showed that a class of coarsening algorithms can be used to construct graphs that have similar spectral properties to the original graph. Huang et al. (2021) show that training a GCN on this coarsened graph can be seen as an approximation to training on the full graph, and that we can use the resulting algorithm on the original graph to obtain node embeddings. Moreover, theoretical and empirical results suggest that this procedure effectively acts as a regularizer, leading to improved generalization performance over the baseline GCN.

Since the GCN deployed on the coarsened graph is ultimately deployed on the original graph to obtain node embeddings, the approach of Huang et al. (2021) does not allow us to capture information at multiple length scales without increasing the number of layers in the GCN. Instead, we construct a hierarchy of graphs, and infer embeddings at each level, using a GCN trained on the coarsest level. These embeddings—each capturing information at a different length-scale—can be combined into a single, multi-resolution embedding for each node. This approach is reminiscent of hierarchical GCNs that learn representations at multiple levels of a hierarchy (Chen et al., 2018a; Akyildiz et al., 2020; Liang et al., 2021; Zhong et al., 2020; Guo et al., 2021; Jiang et al., 2020). However, unlike such methods, we do not incur high computational costs associated with training at each level of the hierarchy.

We find that the resulting algorithm—whose computational and memory requirements are almost identical to those of the coarsened GCN approach of Huang et al. (2021)—typically outperforms both a GCN trained on the original graph, and the coarsened GCN algorithm, which only considers a single resolution. It also outperforms several existing hierarchical GCN algorithms, while having a much lower computational cost.

Our framework, which we denote Scalable, Multi-resolution Graph Representation Learning (SMGRL), consists of three components: a hierarchical graph coarsening algorithm; a GCN learned on a coarsened graph, and a final aggregation step. Each of these components is highly customizable. In particular, we explore two choices of GCN, and in the appendix explore the impact of different aggregation schemes.

2. Related Work

Representing nodes \( v_i \in V \) within a graph \( G = (V, E) \) in terms of embeddings \( h_i \in \mathbb{R}^d \) allows us to distill important information about the graph topology in an easily digestible package. GCNs allow us to learn such embeddings in an inductive manner: rather than directly learn a mapping from \( V \) to \( \mathbb{R}^d \), GCNs learn how to learn such embeddings. Concretely, they parametrize a message-passing algorithm that allows us to update node embeddings by passing messages along the graph’s edges, and aggregating the messages arriving at each node.

Consider a node classification task where each node \( v_i \) has feature \( x_i \) and label \( y_i \). Our goal is to find node embeddings \( h_i \), and a corresponding classification function \( g_\theta \), that minimize some loss

\[
\sum_{i=1}^{|V|} \mathcal{L}(g_\theta(h_i), y_i). \tag{1}
\]

We do so by learning a message passing update rule \( f_\phi \), such that

\[
h_i \leftarrow f_\phi(x_i, \{x_j : v_j \in \text{Ne}(v_i; G)\}), \tag{2}
\]

where \( \text{Ne}(v_i; G) \) indicates the neighborhood of \( v_i \) in \( G \). Multiple layers of the form in (2) can be stacked so that the embedding is based on a multi-hop neighborhood.

While this approach has proved powerful, it does have a number of drawbacks. We describe several such drawbacks below, and discuss existing attempts to mitigate them.

2.1. GCNs have high time and space complexity

Training a GCN involves updating each node’s embedding based on its neighborhood once for each layer, meaning that the computational complexity of training a \( K \)-layer GCN scales as \( O(K|V| \cdot |E|) \). If our training graph is dense, \( |E| \sim O(|V|^2) \), implying a time complexity of \( O(K|V|^3) \); even in the best-case scenario of sparse graphs, the computational cost will scale quadratically with the number of vertices. Further, dependency between nodes makes minibatching challenging: if a GCN has \( K \) layers, then updating the embedding of a single node requires knowledge of the \( K \)-hop neighborhood of that node. This leads to increased memory requirements when minibatching, since we must augment a size-\( m \) minibatch with its \( K \)-hop neighborhood. In dense graphs, this can easily envelope a high proportion of the full graph.

We can reduce the computational and memory requirements by replacing our graph \( G \) with a smaller graph \( G' \) that shares similar properties, and then using the message passing algorithm trained on \( G' \) to obtain embeddings for \( G \). In the simplest case, \( G' \) may be an appropriate subset of the graph. A more sophisticated approach is to construct a lower-dimensional graph that is designed to behave similar to \( G \). Spectral coarsening algorithms (Loukas & Vanderheynst, 2018; Loukas, 2019; Bravo Hermsdorff & Gunderson, 2019; Cai et al., 2021) have been proposed that reduce the number of nodes in a graph, while maintaining spectral properties of the original graph. Along a similar vein, spectral sparsification algorithms aim to reduce the number of edges, while maintaining a similar spectral profile (Spielman & Teng, 2011).
These spectrally-approximating representations are well-matched to the task of learning GCNs, which can be viewed as approximate spectral convolution over the graph (Kipf & Welling, 2017). It is intuitive that a GCN trained on a spectrally coarsened or sparsified graph would also perform well on the original graph. This is seen in practice: Huang et al. (2021) show that a GCN trained on a spectrally coarsened graph can be seen as an approximation to a GCN trained on the full graph. In addition to reducing computation and memory requirements, training a GCN on a spectrally coarsened graph often leads to improved generalization, likely due to the approximation mechanism acting as a regularizer. Similarly, training a GCN on spectrally sparsified graphs has been shown to reduce computational costs while maintaining stable performance (Srinivasa et al., 2020).

Another way to reduce computational and memory requirements is to make approximations to the graph that facilitate minibatching. For example, we can replace the full $K$-hop neighborhood with a (possibly weighted) sub-sample (Hamilton et al., 2017; Chen et al., 2018b, c; Zou et al., 2019). Alternatively, we can make approximations informed by the graph topology (Chiang et al., 2019; Zheng et al., 2020; Bojchevski et al., 2020). Other works aim to deploy GCNs in a distributed setting: for example, Tripathy et al. (2020) reformulate the (exact, full-batch) GCN update equations in order to minimize communication between parallel workers, and Scardapane et al. (2020) combines locally-learned gradients using global constraints on the parameter values.

### 2.2. Embeddings depend only on local neighborhood

In a single-layer GCN, as described in Eq. (2), the embedding of a node depends only on the node features of its immediate neighborhood. To capture long-range dependencies, we must stack multiple layers, increasing the sphere of influence. However, adding more layers leads to over-smoothing—all embeddings become increasingly similar—which in turn leads to degradation in prediction performance (Li et al., 2016; Oono & Suzuki, 2019; Cai & Wang, 2020). A related problem is that GCNs learn inherently local representations: If the local graph topologies of two nodes $v, v'$ are similar, they will be given similar representations even if they are not close within the graph. One way to avoid these problems is to explicitly incorporate positional information. Position-aware graph neural networks (P-GNNs, You et al., 2019; Qin et al., 2021; Dwivedi et al., 2022) select a set of “anchor nodes”, and use the distance to these anchor nodes to calculate a node’s embedding.

An alternative family of approaches elect to make explicit use of hierarchical representations to learn from multiple levels of resolution. In such a representation, we have a sequence of graphs $G_0, \ldots, G_L$, such that $G_0 := G$ and nodes in $G_\ell$ have a single parent in $G_{\ell+1}$. For example, Liang et al. (2021) constructs a hierarchy that retains as much global structure as possible at each level. Embeddings are learned on $G_L$ (the authors do not use a GCN for this step, but it would be possible to do so), and projected to the level below, where they are refined using a GCN. Chen et al. (2018a) and Akylidiz et al. (2020) use a similar hierarchical refinement strategy, but do not make use of GCNs. Hu et al. (2019) use both a sequence of GCNs to learn a hierarchical coarsening, plus a sequence of GCNs that refine representations based on this hierarchy. Li et al. (2020) use an attention-based GCN to learn an initial embedding, and use a hierarchical representation to coarsen and then refine this embedding, allowing information to be shared at multiple resolutions. Other approaches jointly train GCNs at multiple levels in a hierarchy (Guo et al., 2021; Jiang et al., 2020), or learn a single GCN that spans all levels (Zhong et al., 2020).

In addition to hierarchical methods for learning node embeddings, hierarchies have been successfully exploited to learn informative representations of entire graphs. For example, hierarchical pooling methods have been used to combine representations from multiple levels in a hierarchy, resulting in a multi-resolution embedding of the graph (Ying et al., 2018; Xin et al., 2021; Huang et al., 2019; Bandyopadhyay et al., 2020; Liu et al., 2021).

### 3. The Proposed Framework: SMGRL

Hierarchical GCNs, such as those discussed in Section 2.2, allow us to aggregate information at multiple scales, incorporating both long- and short-term dependencies. However, jointly learning GCNs at multiple levels of granularity, or recursively refining embeddings at each level of a hierarchy, increases the computational cost over a single GCN learned on the original graph, making these approaches impractical for large graphs.

Rather than learning multiple message passing algorithms at differing levels of granularity, we note that graphs tend to exhibit self-similarity: like fractals, they appear similar at multiple levels of resolution (Dill et al., 2002). Moreover, graph coarsening algorithms such as those proposed by Loukas (2019) are explicitly designed to preserve spectral properties of the original graph. These two observations suggest that a message passing algorithm learned at one level of granularity might be appropriate at multiple levels. This intuition is made concrete by the coarsened GCN framework of Huang et al. (2021), who train on the coarsest level of a spectrally-coarsened hierarchy, and use the resulting algorithm on the original graph.

We go further: we deploy a GCN learned on the coarsest graph at all levels of the hierarchy. If a GCN trained on level $L$ is a good approximation to one trained on level 0, as shown by Huang et al. (2021), then it must also be a
We begin by constructing hierarchically coarsened representations \( \mathcal{G}_0, \ldots, \mathcal{G}_L \) of our graph, where \( \mathcal{G}_L = (\mathcal{V}_L, \mathcal{E}_L) \) and \( \mathcal{G}_0 := \mathcal{G} \), using the “variation neighborhoods” method introduced by Loukas (2019). Loosely, this algorithm considers collapsing the neighborhood of a single node in \( \mathcal{G}_{\ell-1} \) into a “super-node”, and selects which neighborhoods to collapse in order to minimize a spectral distance between the coarsened graph \( \mathcal{G}_\ell \) and the previous level \( \mathcal{G}_{\ell-1} \). This is repeated in a hierarchical manner—i.e., collapsing the neighborhoods of supernodes from the previous level—until the desired reduction ratio has been achieved. At each level \( \ell > 0 \), we associate each node \( v_{\ell,i} \in \mathcal{V}_\ell \) with a feature \( x_{\ell,i} \) and a label \( y_{\ell,i} \), each calculated as the average of the features and labels of \( v_{\ell,i} \)’s children in the hierarchy.

3.2. Learn message passing rules and use to obtain multi-resolution embeddings

We use the coarsest graph \( \mathcal{G}_L \), and the corresponding features \( X_L \) and labels \( Y_L \), to learn a message passing algorithm \( f_\theta \) and a classification function \( g_\theta \), as described in Eq. (1) and Eq. (2). We note that any GCN can be used here. Once training is complete, we use \( f_\theta \) to learn embeddings \( h_{\ell,i} \) for each node \( v_{\ell,i} \) at each level of the hierarchy, and discard the classification function \( g_\theta \).

3.3. Aggregate the embeddings and learn a new classification function

Each node \( v_i \in \mathcal{V} \) is now associated with a sequence \( h_{0,i}^0, \ldots, h_{L,i}^L \) of embeddings, where \( h_{\ell,i}^\ell \) is the embedding of the ancestor of node \( v_i \) at level \( \ell \). We can then aggregate these embeddings to obtain a final representation \( \tilde{h}_i \) for each node \( v_i \), and learn a new classification function \( g_{\tilde{\theta}} \) based on this new embedding.

A number of choices can be made here. A lightweight option is to either concatenate the embeddings, or take a simple mean. Alternatively, if computational resources allow, we can learn a weighted average alongside the classification function, i.e. \( \tilde{h}_i = \sum_{\ell=0}^L w_\ell^\ell h_{\ell,i}^\ell \). Unless otherwise stated, we use this weighted average approach; we explore alternatives in Appendix C.

Algorithm 1 SMGRL

input Graph \( \mathcal{G} = (\mathcal{V}, \mathcal{E}) \), features \( X \), labels \( Y \), hierarchical coarsening algorithm \( \text{COARSE} \), aggregation method \( \text{AGG} \), GCN \( f_\theta \), classification function \( g_\theta \).

output Node embeddings \( \tilde{h}_i \), trained GCN \( f_{\tilde{\theta}} \), trained classification function \( g_{\tilde{\theta}} \).

1: \( ((\mathcal{G}_\ell, X_\ell, Y_\ell))_{\ell=0}^L = \text{COARSE}(\mathcal{G}, X, Y) \)
2: for \( \ell \in \{1, \ldots, L\} \) do
3:    for \( v_{\ell,i} \in \mathcal{V}_\ell \) do
4:        \( x_{\ell,i} = \frac{\sum_{j : \text{pa}(v_{\ell-1,j}) = v_{\ell,i}} x_{\ell-1,j}}{|\{j : \text{pa}(v_{\ell-1,j}) = v_{\ell,i}\}|}, \) where \( \text{pa}(v) \) is the parent of \( v \) in the hierarchy.
5:        \( y_{\ell,i} = \frac{\sum_{j : \text{pa}(v_{\ell-1,j}) = v_{\ell,i}} y_{\ell-1,j}}{|\{j : \text{pa}(v_{\ell-1,j}) = v_{\ell,i}\}|} \)
6:    end for
7: end for
8: Learn parameters \( \phi \) and \( \theta \) to minimize
9: \( \sum_{i=1}^{\mid \mathcal{V}_L \mid} \mathcal{L}(g_{\theta}(x_{L,i}, \{x_{L,j} : (v_{L,i}, v_{L,j}) \in \mathcal{E}_L\})) \)
10: for \( \ell = 0, \ldots, L \) do
11:    Obtain embeddings \( h_{\ell,i} \) by applying \( f_\theta \) to \( \mathcal{G}_\ell \).
12:    end for
13: for \( i = 1, \ldots, \mid \mathcal{V}_L \mid \) do
14:    for \( \ell = 0, \ldots, L \) do
15:        \( h_{\ell,i}^\ell \leftarrow h_{\ell,j} : v_{\ell,j} \text{ is an ancestor of } v_i \)
16:    end for
17:    Learn parameters \( \tilde{\theta} \) (and if appropriate, parameters of \( \text{AGG}(\cdot) \)) to minimize
18: \( \sum_{i=1}^{\mid \mathcal{V}_L \mid} \mathcal{L}(g_{\tilde{\theta}}(\text{AGG}(h_{0,i}^0, \ldots, h_{L,i}^L), y_i)) \)
19: for \( i = 1, \ldots, \mid \mathcal{V}_L \mid \) do
20:    \( \tilde{h}_i \leftarrow \text{AGG}(h_{0,i}^0, \ldots, h_{L,i}^L) \)
21: end for

The computational cost of our framework scales similarly to that of the coarsened GCN algorithm of Huang et al. (2021)—the coarsening and training scheme are the same; the inference in both cases is dominated by inference on the original graph; and learning the aggregation method in SMGRL is fairly lightweight and only linear in the number of labeled vertices. It is worth noting that the hierarchical coarsening can be pre-computed on CPU: further, since the algorithm is deterministic, we only need compute it once if we are using multiple random seeds or exploring multiple hyperparameter settings for our GCN. We provide a complexity analysis in Appendix A.

Our framework, SMGRL, consists of three independent steps: a hierarchical coarsening algorithm; a GCN trained on the coarsest graph and deployed at all levels; and a final aggregation of the resulting embeddings. We provide details of each step below, and summarize the framework in Algorithm 1.
4. Experiments

SMGRL is designed to improve upon existing GCN-based representation-learning algorithms by learning and aggregating multi-resolution embeddings, capturing long- and short-range variation, without incurring excessive computational costs. In Section 4.2, we provide empirical evidence that using SMGRL with a given GCN can outperform both that GCN on the original graph, and its coarsened version, in addition to outperforming several recent, more complex hierarchical graph models. We go on to show, in Section 4.3, how SMGRL can be used in an inductive setting, and in Section 4.4 we explore how SMGRL can be used in settings where inference on the full graph is not feasible.

4.1. Datasets and comparison methods

We consider four multi-label classification datasets, summarized in Table 1: Cora (Sen et al., 2008), a Machine Learning citation graph labeled by subfield; CiteSeer (Sen et al., 2008), a scientific citation graph labeled by research area; PubMed (Sen et al., 2008), a citation graph between papers on diabetes, classified into three topics; and DBLP (Tang et al., 2008), a Computer Science citation network, labeled by subfield. To assess classification performance on these datasets, we look at the macro $F_1$ score on test-set labels; considering alternative metrics such as accuracy showed similar trends. Unless otherwise stated, we use the default train/validation/test split associated with each dataset for evaluating the macro $F_1$ score, and use the full graph topology when training.

We apply SMGRL to two popular GCNs: GraphSAGE (Hamilton et al., 2017), a GCN that uses skip connections to construct a generalized neighborhood aggregation scheme; and APPNP (Klicpera et al., 2019), a GCN that uses an aggregation scheme inspired by personalized PageRank. For space reasons, results on APPNP are included in the Appendix. We used existing Pytorch implementations of GraphSAGE and APPNP\footnote{https://pytorch-geometric.readthedocs.io} and APPNP\footnote{https://github.com/rezanmz/SMGRL}. To transform embeddings into class predictions, we used a single-layer perceptron neural network. To construct our hierarchy, we used the variation neighborhood coarsening method introduced by Loukas (2019), using the original author’s implementation\footnote{https://github.com/loukasa/graph-coarsening}. This approach automatically determines the appropriate hierarchical depth, based on the desired reduction ratio. Unless otherwise stated, we learn per-dimension weights for each embedding, in order to aggregate the multi-resolution embeddings as described in Section 3.3. These weights are learned jointly with our final classification algorithm. In Appendix C, we explore some alternative aggregation schemes.

In addition to comparing against non-hierarchical versions of GraphSAGE and APPNP, we compare our model against two popular hierarchical node embedding methods: MILE (Liang et al., 2021) and HARP (Chen et al., 2018a), implemented using the authors’ original code\footnote{https://github.com/jiongqian/MILE} with default settings. We also include results using just the top layer of our hierarchy, which is equivalent to Huang et al. (2021). For all experiments, we use the same single-layer perceptron classifier, replacing any pre-existing classifier if necessary.

Hyperparameter settings and implementation details

For both GraphSAGE and APPNP, we use additive aggregation of messages. For APPNP, we used parameters $k = 3$ and $\alpha = 0.5$; for all other methods, we used the default parameters suggested by the authors. For all GCN-based methods, we use a single-layer GCN, and optimized using RMSProp with default hyperparameters. We carried out early stopping if validation loss did not improve in five epochs. For each experiment, we averaged over 200 runs with random seeds. Code to reproduce our experiments are available at https://github.com/rezanmz/SMGRL.

| Name | Vertices | Edges | Classes | Features |
|------|----------|-------|---------|----------|
| Cora | 2,708    | 10,556| 7       | 1,433    |
| CiteSeer | 3,327   | 9,104 | 6       | 3,703    |
| PubMed | 19,717   | 88,648| 3       | 500      |
| DBLP | 17,716   | 105,734| 4      | 1639     |

Table 1. Dataset overview

4.2. Our hierarchical representation leads to high-quality embeddings

To explore the quality of the embeddings obtained using SMGRL, we first compare with the two closest comparison methods: a single-layer GCN, and a coarsened GCN (Huang et al., 2021). Figure 1 shows test set macro $F_1$ score on our four datasets, using a reduction ratio of 0.4, and with a variety of embedding dimension. In addition to the full SMGRL approach, we show results obtained by training a classifier on the embeddings from each layer in isolation; note that the embedding for layer 0 corresponds to a coarsened GCN. In addition, we include GraphSAGE which trained and deployed on the original graph. In Appendix B, we show comparable results holding embeddings dimension fixed and adjusting the reduction ratio (Figure 5), plus analogous results using APPNP instead of GraphSAGE.

We see several interesting patterns here. First, we note that, as found by Huang et al. (2021), learning a GCN on a coarsened graph and then deploying it on the full graph (Level 0)
SMGRL: A Scalable Multi-resolution Graph Representation Learning Framework

![GraphSAGE, SMGRL, Level 0 embeddings, Level 1 embeddings](a) Cora

![GraphSAGE, SMGRL, Level 0 embeddings, Level 1 embeddings](b) CiteSeer

![GraphSAGE, SMGRL, Level 0 embeddings, Level 1 embeddings](c) PubMed

![GraphSAGE, SMGRL, Level 0 embeddings, Level 1 embeddings](d) DBLP

**Figure 1.** Macro $F_1$ score on four different graphs, using GraphSAGE, with varying embedding dimension. In each case, we use a reduction ratio of 0.4, which results in a two-level hierarchy. Yellow indicates basic GraphSAGE on the original graph. Green indicates embeddings obtained using the coarsened graph. Blue indicates embeddings obtained on the original graph, using the GCN trained on the top level – equivalent to Huang et al. (2021). Black indicates a combined embedding obtained using a weighted average.

typically does better than the baseline GCN (GraphSAGE); they suggested this is because using the coarsened graph acts as a regularizer, due to limiting the space of possible convolution operators. However, we find that using just the top-level embeddings—i.e., assigning to each node in $G$ the embeddings obtained for its ancestor in the coarsened graph—actually perform better than the embeddings obtained using the same algorithm on the original graph. We hypothesise that this is due to two reasons. First, since the GCN is trained on the coarsest layer, we would expect the embeddings obtained at this resolution to be highly predictive. This alone does not explain why these embeddings outperform embeddings trained on the original graph, leading to our second hypothesis: the lengthscale associated with the coarser graph may be better aligned with the natural variation in labels.

We also find that combining the embeddings from multiple layers using learned weights tends to lead to better predictive performance than any single layer. This is not surprising: we are able to aggregate information across multiple length-scales. In principle, if a given level of the hierarchy does not contain relevant information, we can learn to downweight embeddings from that level in the final aggregation.

Clearly, incorporating information obtained at multiple levels of granularity can improve performance over a single level of granularity—whether that level is the original graph, or a single coarsened layer. However, only one level in the hierarchy has embeddings obtained using a GCN trained on that level. A reasonable question to ask might be whether training one GCN per layer, and combining the resulting embedding, does better than our approach. In Table 2 we do exactly that: on the Cora dataset, we compare the SMGRL framework with a single GCN learned at the top level, with a variant that learns a separate GCN for each level. When using our standard weighting aggregation scheme, we perform comparably to this variant, for a significantly lower training cost. Note that, when using a simple mean aggregation, we uniformly outperform the separate GCNs. This is because using a single GCN means that the embeddings at the two levels are compatible—they aggregate information in the same manner, so each dimension carries similar information.
Figure 2. Macro $F_1$ score on four different graphs, with varying embedding dimensions. SMGRL uses GraphSAGE, with a reduction ratio of 0.4. HARP uses node2vec as the underlying network embedding model with default parameters $p = 1$ and $q = 1$. MILE uses NetMF as the base embedding model and GCN as the refinement model.

Table 2. Macro $F_1$ score on Cora dataset, with a reduction ratio of 0.4 (leading to a two-level hierarchy) and various embedding dimensions, on two variants of SMGRL. Top: the standard SMGRL algorithm, where a single GraphSAGE model is learned on the top level; Bottom: a variant where a separate GraphSAGE model is learned at each level. We consider three aggregation methods, plus the embeddings obtained on the original graph (equivalent to the coarsened GCN model). For each embedding type, we bold the better of the two results across the two models. Note that the stated embedding dimension is that of the per-level embeddings; the concatenated embeddings are twice this length.

In Figure 2, we compare against two popular hierarchical graph embedding methods, HARP (Chen et al., 2018a) and MILE (Liang et al., 2021). Both these methods are much more complex than SMGRL, and accordingly had notably higher runtimes. Despite this, we see that SMGRL outperforms both approaches for most settings.
4.3. SMGRL can be used in an inductive manner

A key advantage of GCNs over other graph-based neural network models is that they are inductive. Rather than learn a direct mapping from node to embedding, GCNs learn message passing algorithms that can be applied to arbitrary graphs. Indeed, we directly make use of this when we apply an algorithm trained at one level of the hierarchy, to get embeddings at other levels.

We consider a mixed inductive/transductive setting, where a randomly selected subset of test set nodes are not included in the training graph. We refer to the held-out test set nodes as our inductive test set, and the test set nodes present at training time as our transductive test set. At inference time, we reincorporate the held-out nodes at each level in our hierarchy. At the bottom level, we simply reintroduce the nodes and any missing edges. At higher levels, we assign the new nodes to the partitions with which they have the highest number of edges.

In Figure 3, we show macro $F_1$ on Cora scores for a reduction ratio of 0.4, 8-dimensional embeddings, and a variety of held-out percentages. Error bars show one standard deviation under different random held-out sets. While we do perform better on the transductive test set than the inductive test set, the difference is small relative to the variation across partitions, indicating that SMGRL continues to perform well when a moderate number of new nodes are added to the graph.

4.4. Distributing inference for large graphs

SMGRL dramatically reduces training time of a GCN, since training is carried out over a smaller graph. However, inferring the final embeddings occurs on the full graph (plus any additional, smaller layers in the hierarchy). While, unlike training, inference only requires a single pass through the graph, this inference cost—and the associated memory requirements—may be infeasible if our graph is large.

In such settings, we propose partitioning the graph at each level of the hierarchy, based on their parents in the graph, to approximate the full graph with a sequence of independent subgraphs. We can then infer the embeddings on each subgraph in parallel, making use of distributed resources if available. This leads to lower computational complexity and memory requirements, but will tend to reduce the quality of the embeddings, since we are ignoring many edges in the original graph. In Figure 4, we compare the macro $F_1$ score of embeddings obtained using the disjoint subgraphs at lower levels, and the embeddings obtained using the full graph, on the Cora dataset. We see that, while there is a drop in performance due to using the disjoint subsets, it is small and may be a worthwhile trade-off in practice.

5. Discussion

Many recent graph representation methods in the literature aim to generate a representation of the graph that preserves nodes’ local proximity; however doing so often adds significant complexity to the model. In this work we introduced SMGRL, an extremely lightweight framework for learning multi-resolution node embeddings. SMGRL is highly customizable and can be applied on top of any GCN. Moreover, its computational cost is approximately equivalent to that of the recently proposed, coarsening-based scalable GCN framework of Huang et al. (2021), while introducing increased expressive power due to incorporating information at multiple lengthscales. Our experiments on real-world datasets show impressive empirical performance.

Due to its modularity, SMGRL suggests a number of extensions. Future directions might be to explore alternative coarsening algorithms, or incorporate pre-existing partitioning information, such as partitioning users of a social network by employer or country.
SMGRL: A Scalable Multi-resolution Graph Representation Learning Framework

References

Akyildiz, T. A., Aljundi, A. A., and Kaya, K. GOSH: Embedding big graphs on small hardware. In International Conference on Parallel Processing-ICPP, 2020.

Bandyopadhyay, S., Aggarwal, M., and Murty, M. N. Self-supervised hierarchical graph neural network for graph representation. In IEEE International Conference on Big Data, 2020.

Bianchi, F. M., Grattarola, D., Livi, L., and Alippi, C. Graph neural networks with convolutional ARMA filters. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2021.

Bojchevski, A., Klicpera, J., Perozzi, B., Kapoor, A., Blais, M., Rózemberczki, B., Lukasik, M., and Günnemann, S. Scaling graph neural networks with approximate PageRank. In Knowledge Discovery and Data Mining, 2020.

Bravo Hermsdorff, G. and Gunderson, L. A unifying framework for spectrum-preserving graph sparsification and coarsening. In Advances in Neural Information Processing Systems, 2019.

Brockschmidt, M. GNN-film: Graph neural networks with feature-wise linear modulation. In International Conference on Machine Learning, 2020.

Cai, C. and Wang, Y. A note on over-smoothing for graph neural networks. arXiv preprint arXiv:2006.13318, 2020.

Cai, C., Wang, D., and Wang, Y. Graph coarsening with neural networks. In International Conference on Learning Representations, 2021.

Chen, H., Perozzi, B., Hu, Y., and Skiena, S. HARP: Hierarchical representation learning for networks. In AAAI Conference on Artificial Intelligence, 2018a.

Chen, J., Ma, T., and Xiao, C. FastGCN: Fast learning with graph convolutional networks via importance sampling. In International Conference on Learning Representations, 2018b.

Chen, J., Zhu, J., and Song, L. Stochastic training of graph convolutional networks with variance reduction. In International Conference on Machine Learning, 2018c.

Chiang, W.-L., Liu, X., Si, S., Li, Y., Bengio, S., and Hsieh, C.-J. Cluster-GCN: An efficient algorithm for training deep and large graph convolutional networks. In Knowledge Discovery and Data Mining, 2019.

Dill, S., Kumar, R., McCurley, K. S., Rajagopalan, S., Sivakumar, D., and Tomkins, A. Self-similarity in the web. ACM Transactions on Internet Technology, 2(3): 205–223, 2002.

Dwivedi, V. P., Luu, A. T., Laurent, T., Bengio, Y., and Bresson, X. Graph neural networks with learnable structural and positional representations. In International Conference on Learning Representations, 2022.

Grover, A. and Leskovec, J. node2vec: Scalable feature learning for networks. In Knowledge Discovery and Data Mining, 2016.

Guo, K., Hu, Y., Sun, Y., Qian, S., Gao, J., and Yin, B. Hierarchical graph convolution networks for traffic forecasting. In AAAI Conference on Artificial Intelligence, 2021.

Hamilton, W. L., Ying, R., and Leskovec, J. Inductive representation learning on large graphs. In Advances in Neural Information Processing Systems, 2017.

Hu, F., Zhu, Y., Wu, S., Wang, L., and Tan, T. Hierarchical graph convolutional networks for semi-supervised node classification. International Joint Conference on Artificial Intelligence, 2019.

Huang, J., Li, Z., Li, N., Liu, S., and Li, G. Attpool: Towards hierarchical feature representation in graph convolutional networks via attention mechanism. In International Conference on Computer Vision, 2019.

Huang, Z., Zhang, S., Xi, C., Liu, T., and Zhou, M. Scaling up graph neural networks via graph coarsening. In Knowledge Discovery and Data Mining, 2021.

Jiang, H., Cao, P., Xu, M., Yang, J., and Zaiane, O. Hi-GCN: A hierarchical graph convolution network for graph embedding learning of brain network and brain disorders prediction. Computers in Biology and Medicine, 127: 104096, 2020.

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. In International Conference on Learning Representations, 2017.

Klicpera, J., Bojchevski, A., and Günnemann, S. Predict then propagate: Graph neural networks meet personalized PageRank. In International Conference on Learning Representations, 2019.

Li, K., Feng, Y., Gao, Y., and Qiu, J. Hierarchical graph attention networks for semi-supervised node classification. Applied Intelligence, 50(10):3441–3451, 2020.

Li, Y., Tarlow, D., Brockschmidt, M., and Zemel, R. Gated graph sequence neural networks. In International Conference on Learning Representations, 2016.

Liang, J., Gurukar, S., and Parthasarathy, S. MILE: A multi-level framework for scalable graph embedding. In International AAAI Conference on Web and Social Media, 2021.
Liu, N., Jian, S., Li, D., Zhang, Y., Lai, Z., and Xu, H. Hierarchical adaptive pooling by capturing high-order dependency for graph representation learning. *arXiv preprint arXiv:2104.05960*, 2021.

Loukas, A. Graph reduction with spectral and cut guarantees. *Journal of Machine Learning Research*, 20(116):1–42, 2019.

Loukas, A. and Vandergheynst, P. Spectrally approximating large graphs with smaller graphs. In *International Conference on Machine Learning*, 2018.

Oono, K. and Suzuki, T. Graph neural networks exponentially lose expressive power for node classification. In *International Conference on Learning Representations*, 2019.

Perozzi, B., Al-Rfou, R., and Skiena, S. Deepwalk: Online learning of social representations. In *Knowledge Discovery and Data Mining*, 2014.

Qin, Z., Anwar, S., Kim, D., Liu, Y., Ji, P., and Gedeon, T. Position-sensing graph neural networks: Proactively learning nodes relative positions. *arXiv preprint arXiv:2105.11346*, 2021.

Scardapane, S., Spinelli, I., and Di Lorenzo, P. Distributed training of graph convolutional networks. *IEEE Transactions on Signal and Information Processing over Networks*, 2020.

Sen, P., Namata, G., Bilgic, M., Getoor, L., Galligher, B., and Eliassi-Rad, T. Collective classification in network data. *AI Magazine*, 29(3):93, 2008.

Spielman, D. A. and Teng, S.-H. Spectral sparsification of graphs. *SIAM Journal on Computing*, 40(4):981–1025, 2011.

Srinivasa, R. S., Xiao, C., Glass, L., Romberg, J., and Sun, J. Fast graph attention networks using effective resistance based graph sparsification. *arXiv preprint arXiv:2006.08796*, 2020.

Tang, J., Zhang, J., Yao, L., Li, J., Zhang, L., and Su, Z. Arnetminer: extraction and mining of academic social networks. In *Knowledge Discovery and Data Mining*, 2008.

Tripathy, A., Yelick, K., and Buluç, A. Reducing communication in graph neural network training. In *International Conference for High Performance Computing, Networking, Storage and Analysis*, 2020.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., and Bengio, Y. Graph attention networks. In *International Conference on Learning Representations*, 2018.
A. Computational complexity of SMGRL

We discuss the cost of each step of SMGRL below, noting that the first two steps are identical to the coarsened GCN framework of Huang et al. (2021).

- Constructing a coarsened graph using the neighborhood-based coarsening algorithm of Loukas (2019) scales (up to polylog terms) linearly with both the number of edges, and the product of the number of nodes and the average degree – both of which are upper bounded by $|V|^2$.

- Using a coarsening method with reduction ratio $r$, the coarsened graph $G_L$ has $|V_L| = (1 - r)|V|$ vertices. The number of edges in $G_L$ is upper bounded by $\min(\lvert E \rvert, (1 - r)^2 |V|^2)$. This implies that training on the coarsened graph will have a worst case computational complexity $(1 - r)$ times the complexity of the original GCN. In a more realistic case, the number of edges will be lower than this upper bound, and the computational savings greater. Both these steps also take place in the coarsened GCN algorithm.

- Since the coarsened graphs are smaller than the original graph, for both SMGRL and the coarsened algorithm the inference step is dominated by the cost of inference on the original graph, which scales as $O(K|V| \cdot |E|^2)$. Since our hierarchy allows us to extract information at multiple resolutions by design, we choose to let the number of layers, $K$, to be one. In practice the number of iterations required for convergence of the GCN means that in the graphs considered in this paper, the cost of inference is smaller than the cost of training. If desired, inference can be approximated by partitioning the graphs at levels $\ell < L$ based on their parents in level $\ell + 1$, and independently inferring embeddings on the resulting subgraph. This has the additional advantage of allowing parallelization across subgraphs. We empirically explore the resulting error in Section 4.4.

- Aggregating the embeddings and learning the final classification algorithm is linear in the number of labeled vertices.

B. Additional experiments comparing SMGRL to alternative schemes

![Graphs showing Macro F1 score on four different graphs, using GraphSAGE, with varying reduction ratio and embedding dimension of 8.](image)

Figure 5. Macro $F_1$ score on four different graphs, using GraphSAGE, with varying reduction ratio and an embedding dimension of 8. Blue indicates embeddings obtained on the original graph, using the GCN trained on the top level – equivalent to Huang et al. (2021). Black indicates a combined embedding obtained using a weighted average. Other colors represent single layer embeddings. Note that a reduction ratio of zero corresponds to GraphSAGE on the original graph.
In Section 4.2, we compared how macro $F_1$ scores between GraphSAGE on a single graph, SMGRL with GraphSAGE, and the per-level embeddings obtained using SMGRL with GraphSAGE, for a fixed reduction ratio and varying embedding dimensions. Note that our Level 0 embeddings are equivalent to coarsened GraphSAGE. In Figure 5, we show equivalent plots holding the embedding dimension fixed at 8 and with varying reduction ratios. Note that the number of levels of the hierarchy are determined by the reduction ratio. We see a similar pattern: For all but the highest reduction ratio, the GCN learned on the coarsest graph performs well at all layers, typically leading to embeddings that perform better than those obtained using a GCN trained on the original graph (indicated by a reduction ratio of zero). And in almost all cases, an aggregated embedding which considers all levels of the hierarchy outperforms any single level’s embeddings.

![Figure 5](image)

Figure 5. Equivalent plots holding the embedding dimension fixed at 8 and with varying reduction ratios. Note that the number of levels of the hierarchy are determined by the reduction ratio. We see a similar pattern: For all but the highest reduction ratio, the GCN learned on the coarsest graph performs well at all layers, typically leading to embeddings that perform better than those obtained using a GCN trained on the original graph (indicated by a reduction ratio of zero). And in almost all cases, an aggregated embedding which considers all levels of the hierarchy outperforms any single level’s embeddings.

In Figures 6 and 7, we perform the same experiments, using APPNP instead of GraphSAGE as our underlying GCN. We see a similar pattern: Using embeddings trained on a coarsened graph leads to improved performance over the baseline APPNP model, and aggregating such embeddings yields a further performance boost.

![Figure 6](image)

Figure 6. Macro $F_1$ score on four different graphs, using APPNP, with varying embedding dimension. In each case, we use a reduction ratio of 0.4, which results in a two-level hierarchy. Yellow indicates basic APPNP on the original graph. Green indicates embeddings obtained using the coarsened graph. Blue indicates embeddings obtained on the original graph, using the GCN trained on the top level – equivalent to Huang et al. (2021). Black indicates a combined embedding obtained using a weighted average.

In Figures 6 and 7, we perform the same experiments, using APPNP instead of GraphSAGE as our underlying GCN. We see a similar pattern: Using embeddings trained on a coarsened graph leads to improved performance over the baseline APPNP model, and aggregating such embeddings yields a further performance boost.

C. Exploration of alternative final aggregation schemes

As we have seen in the experiments so far, learning post-hoc weights to combine the embeddings works well in practice. However, this does add additional computational complexity, and limits our choice of classifier to one that can be jointly trained with the post-hoc weightings. In Figures 8 and 9, we compare our post-hoc weighting scheme with two alternatives: taking a simple mean, and directly concatenating the embedding. All experiments use single-layer GraphSAGE, with a variety of reduction rations and embedding dimensions as described in the captions. Note that embedding dimension here refers to the per-layer embedding dimension; the concatenated dimension will be multiplied by the number of dimensions in the hierarchy. We see that in general, the learned weighted mean performs better than the alternative approaches; however the difference is fairly small. This leads us to suggest using a simple mean if computational resources are limited.
Figure 7. Macro $F_1$ score on four different graphs, using APPNP, with varying reduction ratio and an embedding dimension of 8. Blue indicates embeddings obtained on the original graph, using the GCN trained on the top level – equivalent to Huang et al. (2021). Black indicates a combined embedding obtained using a weighted average. Other colors represent single layer embeddings. Note that a reduction ratio of zero corresponds to APPNP on the original graph.

Figure 8. Macro $F_1$ score on four different graphs, for three different aggregation schemes, using SMGRL + GraphSAGE. The reduction ratio is fixed at 0.4, and a variety of embedding dimensions are shown.
Figure 9. Macro $F_1$ score on four different graphs, for three different aggregation schemes, using SMGRL + GraphSAGE. The embedding dimension is fixed at 8, and a variety of reduction ratios are shown.