Applying GraphSAGE to Large-Scale Node Classification

Programming Machine Learning Models for HPC
Final Report

Thomas Gantz Alberto Finardi Tommaso Crippa Jan Marxen

December 2025

Abstract

This report documents our study and application of GraphSAGE (Graph SAmple and aggreGatE) [2], an inductive framework for learning node embeddings on large-scale graphs. We provide background on graph neural networks and the message passing paradigm, then describe how GraphSAGE addresses the scalability limitations of earlier approaches by learning aggregator functions rather than fixed node embeddings. We apply GraphSAGE to the ogbn-products benchmark dataset for node classification, detailing our implementation choices, hyperparameter tuning process, and experimental results.

Contents

1 Introduction **2**
- 1.1 Graph Prediction Tasks . . . 2
- 1.2 What is a Graph Neural Network? . . . 2
- 1.3 Message Passing: The Core Mechanism . . . 2
- 1.4 Limitations of Existing Approaches . . . 3

2 The Key Insight **3**

3 GraphSAGE Framework **3**
- 3.1 Algorithm Overview . . . 4
- 3.2 Neighbor Sampling . . . 4
- 3.3 Aggregator Functions . . . 4
- 3.4 Training Objective . . . 4

4 Implementation **5**

5 Hyperparameters **5**

6 Results **5**

7 Conclusion **5**

1 Introduction

Graph-structured data appears in many domains, including social networks, citation graphs, biological protein interaction networks, and recommendation systems. A central challenge in machine learning on graphs is learning representations (embeddings) of nodes that capture both their features and their structural role in the graph.

1.1 Graph Prediction Tasks

Machine learning on graphs encompasses several prediction tasks at different granularities:

- **Node-level prediction:** Predict properties of individual nodes using their features and neighborhood information. Examples include classifying users in social networks or predicting protein functions.
- **Edge-level prediction:** Predict relationships between pairs of nodes, such as link prediction (will two users become friends?) or edge classification (what type of relationship exists?).
- **Graph-level prediction:** Predict properties of entire graphs, commonly used in molecular property prediction where each molecule is represented as a graph.

This report focuses primarily on **node-level prediction**, specifically node classification, which is the task addressed by GraphSAGE.

1.2 What is a Graph Neural Network?

A Graph Neural Network (GNN) is a learnable transformation on graph-structured data that:

1. Updates node, edge, and/or graph features using neural networks
2. Respects the graph structure by aggregating information from neighboring nodes
3. Is permutation invariant: the output does not depend on the ordering of nodes

The key insight behind GNNs is that a node's representation should be informed not only by its own features but also by the features and structure of its local neighborhood. This is achieved through iterative message passing between connected nodes.

1.3 Message Passing: The Core Mechanism

The message passing paradigm forms the foundation of most modern GNN architectures. At each layer k, every node updates its representation by:

1. **Gather:** Collect embeddings from neighboring nodes
2. **Aggregate:** Combine neighbors' information using a permutation-invariant function
3. **Update:** Apply a learned transformation to produce the new embedding

Formally, the message passing equation can be written as:

$$h_v^{(k)} = \sigma\left(W^{(k)} \cdot \left[h_v^{(k-1)}, \text{AGG}\left(\{h_u^{(k-1)} : u \in \mathcal{N}(v)\}\right)\right]\right) \tag{1}$$

where:

- $h_v^{(k)}$ is the representation of node v at layer k

- $\mathcal{N}(v)$ is the set of neighbors of node v
- $\text{AGG}(\cdot)$ is a permutation-invariant aggregation function (e.g., mean, sum, max)
- $W^{(k)}$ is a learnable weight matrix for layer k
- σ is a non-linear activation function (e.g., ReLU)

An intuitive analogy: while CNNs aggregate over fixed local pixel neighborhoods (e.g., 3×3 patches), GNNs aggregate over variable-size neighbor sets defined by the graph topology.

1.4 Limitations of Existing Approaches

Before GraphSAGE, existing approaches for node representation learning had significant limitations:

- **DeepWalk / node2vec:** These methods [4, 1] learn embeddings via random walks on the graph. However, they are *transductive*, meaning they learn a fixed embedding for each node in the training graph. Adding new nodes requires retraining on the entire graph. Note that these are not GNNs since they do not use message passing.
- **Graph Convolutional Networks (GCNs):** GCNs [3] are proper GNNs that use message passing, but they typically operate on the full graph adjacency matrix during training and inference. This makes them difficult to scale to large graphs and also transductive in practice.

The fundamental problem: **no existing method provided a compact parametric function that could generate embeddings for previously unseen nodes.**

2 The Key Insight

The central innovation of GraphSAGE is a shift in perspective:

> *Rather than learning embeddings for each node, learn a **function** that generates embeddings.*

Instead of optimizing a unique embedding vector for each node (transductive), GraphSAGE learns *aggregator functions* that can generate embeddings for any node by sampling and aggregating features from its neighborhood. This makes the approach **inductive**: the learned model can generalize to entirely new nodes or even new graphs, as long as node features are available.

The key observation is that a node's local neighborhood structure contains rich information about its role and function in the graph. By learning to aggregate this neighborhood information in a principled way, we can generate meaningful embeddings for any node.

3 GraphSAGE Framework

GraphSAGE (Graph SAmple and aggreGatE) implements the inductive learning paradigm through two main components: neighbor sampling and learned aggregation.

3.1 Algorithm Overview

The GraphSAGE forward propagation algorithm works as follows:

Algorithm 1 GraphSAGE Forward Propagation

Require: Graph $G = (V, E)$; node features $\{x_v, \forall v \in V\}$; depth K; weight matrices $W^k, \forall k \in \{1, \ldots, K\}$; aggregator functions $\mathrm{AGG}_k, \forall k \in \{1, \ldots, K\}$; neighborhood sampling functions $\mathcal{N}_k : v \to 2^V$

Ensure: Node embeddings $z_v, \forall v \in V$

$h_v^0 \leftarrow x_v, \forall v \in V$
for $k = 1$ to K **do**
  for $v \in V$ **do**
    $h_{\mathcal{N}(v)}^k \leftarrow \mathrm{AGG}_k\left(\{h_u^{k-1}, \forall u \in \mathcal{N}_k(v)\}\right)$
    $h_v^k \leftarrow \sigma\left(W^k \cdot \mathrm{CONCAT}(h_v^{k-1}, h_{\mathcal{N}(v)}^k)\right)$
  end for
  $h_v^k \leftarrow h_v^k / \|h_v^k\|_2, \forall v \in V$ ▷ Optional normalization
end for
return $z_v \leftarrow h_v^K, \forall v \in V$

3.2 Neighbor Sampling

A critical component for scalability is **neighbor sampling**. Instead of aggregating over all neighbors (which can be prohibitively expensive for high-degree nodes), GraphSAGE uniformly samples a fixed-size set of neighbors at each layer.

For a K-layer model with sample sizes $S_1, S_2, \ldots, S_K$, the computational complexity per node is $O(\prod_{k=1}^{K} S_k)$, independent of the actual node degrees in the graph.

3.3 Aggregator Functions

GraphSAGE explores several aggregator architectures:

- **Mean aggregator:** Simply takes the element-wise mean of neighbor embeddings:

$$\mathrm{AGG}_{\mathrm{mean}} = \frac{1}{|\mathcal{N}(v)|} \sum_{u \in \mathcal{N}(v)} h_u^{k-1} \tag{2}$$

- **LSTM aggregator:** Applies an LSTM to a random permutation of neighbors, providing more expressive power at the cost of losing strict permutation invariance.

- **Pooling aggregator:** Applies a neural network to each neighbor followed by element-wise max pooling:

$$\mathrm{AGG}_{\mathrm{pool}} = \max\left(\{\sigma(W_{\mathrm{pool}} h_u + b), \forall u \in \mathcal{N}(v)\}\right) \tag{3}$$

3.4 Training Objective

For supervised node classification, GraphSAGE is trained end-to-end using cross-entropy loss:

$$\mathcal{L} = -\sum_{v \in V_{\mathrm{train}}} \sum_{c=1}^{C} y_{v,c} \log(\hat{y}_{v,c}) \tag{4}$$

where $y_{v,c}$ is the ground-truth label and $\hat{y}_{v,c}$ is the predicted probability for class c.

4 Implementation

Implementation details will be provided here, including our PyTorch/DGL code structure, data loading pipeline, and distributed training setup.

5 Hyperparameters

Hyperparameter tuning methodology and final configurations will be documented here, including learning rate schedules, hidden dimensions, number of layers, sample sizes, dropout rates, and batch sizes.

6 Results

Experimental results on the ogbn-products dataset will be presented here, including accuracy metrics, training curves, and comparisons with baseline methods.

7 Conclusion

Summary of findings and conclusions will be provided here.

References

[1] Aditya Grover and Jure Leskovec. "node2vec: Scalable Feature Learning for Networks". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2016, pp. 855–864.

[2] William L. Hamilton, Rex Ying, and Jure Leskovec. "Inductive Representation Learning on Large Graphs". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 30. 2017, pp. 1024–1034. URL: `https://arxiv.org/abs/1706.02216`.

[3] Thomas N. Kipf and Max Welling. "Semi-Supervised Classification with Graph Convolutional Networks". In: *arXiv preprint arXiv:1609.02907* (2016).

[4] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. "DeepWalk: Online Learning of Social Representations". In: *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2014, pp. 701–710.