GNN Training Acceleration

Presenters:

Aditya Gupta

Armaan Khetarpaul

Shankaradithyaa Venkateswaran

Umang Majumder

Methodology

- AIM: Analyze tradeoffs of different methods for GNN training
- Compute: Ryzen 7 8845 HS, RTX 4060 laptop GPU
- 2 Sampling Methods & 2 Sparsification Methods
- Compared results obtained from each method
- Measured EDP for each method

Dataset

The Open Graph Benchmark Proteins dataset

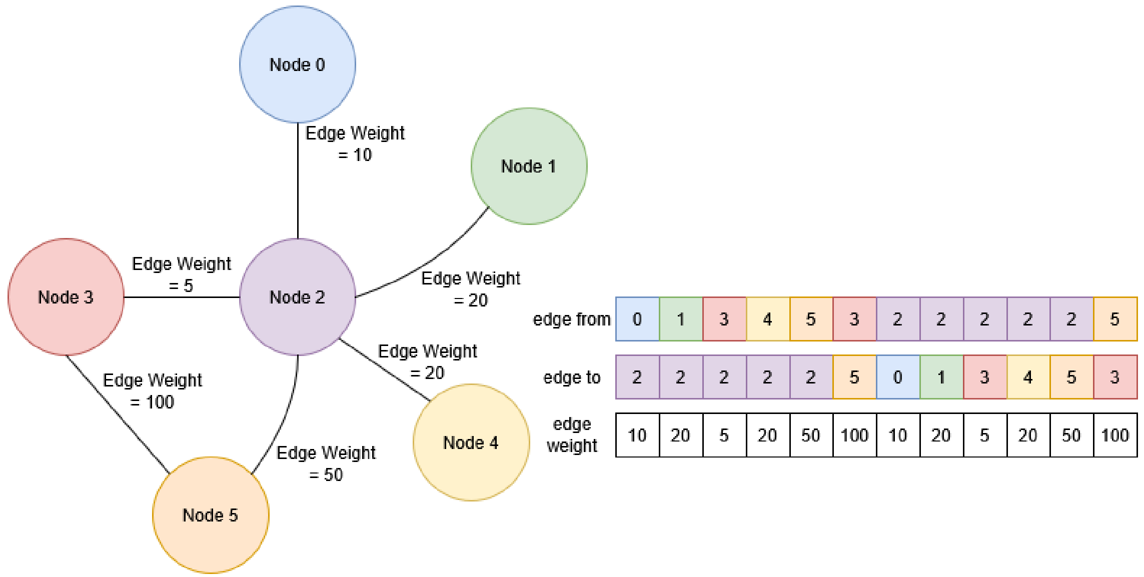
Nodes: 132,534
Edges: 39,561,252

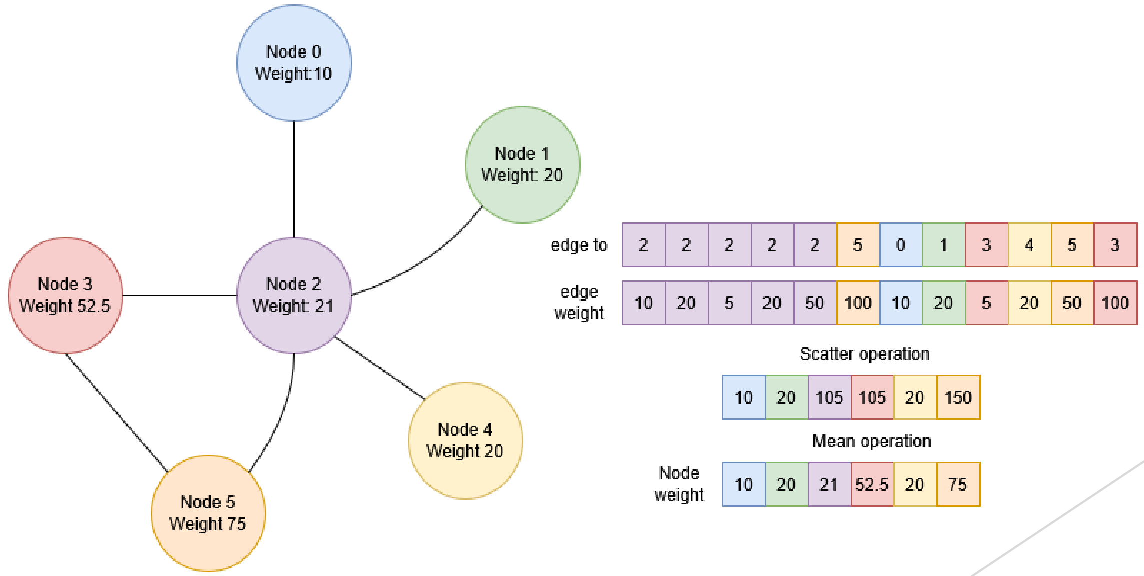
Each edge has 8 attributes

To predict: 112 binary properties on each Node

Scatter

Edge information to
Node information





DropEdge

- At each training epoch, DropEdge technique drops out a certain rate of edges of the input graph by random.
- It enforced $V \times P$ non-zero elements of the adjacency matrix to be zero. Where V is the total number of edges at that epoch and P is the dropping rate.

GraphSAGE

- SAGE: SAmple + aggreGatE method.
- Random sampling to reduce memory constraints while loading large datasets into memory.
- Max-Pooling Aggregator to aggregate node level information.
- Produces node level embedding for graph classification tasks.
- Can be used inductively on unseen structures to generate embeddings.

GraphSAINT

- Assign probabilities to nodes based on their features
- Build an adjacency matrix A containing node features, then normalize it across a row
- Calculate the probability of choosing a node $P(u) \propto \left|\left|\tilde{A}_{:,u}\right|\right|^2$
- Nodes with more connections and stronger features are naturally chosen more often

Neural Sparse

- Identify set of candidate neighbors (immediate, 1-hop, etc.)
- Concatenate candidate edge weights and node weights
- MLP gives "importance"
- Pick top k
- Gumbel Softmax to make samples differentiable

$\forall v$ in batch:

$$N_v = \text{Candidate neighbors of } v$$

$$E_v = \text{Candidate edges}$$

$$z = Softmax(MLP_\phi(\mathbb{V}(v), \mathbb{V}(N_v), \mathbb{E}(E_v)))$$

$$\pi_u = \frac{e^{(\log z_u + \epsilon_u)/\tau}}{\sum_{u' \in N_v} e^{(\log z_{u'} + \epsilon_{u'})/\tau}}$$

$$u_1, \ldots, u_k \sim \pi$$

$(v, u_1), \ldots, (v, u_k)$ are used to make the sparsified subgraph.

Results

Method	Test ROC AUC	Training Time per Epoch	Epochs	Power (after subtracting idle power draw)	EDP (time/epoch * #epochs * Power) (W min)
Baseline	66.56	18s	49	8.31 watts	122.16
DropEdge	64.15	35s	17	18.50 watts	183.46
GraphSAGE	68.27	14.7s	29	11.92 watts	84.69
GraphSAINT	72.23	1.4s	45	10.5 watts	11.03
NeuralSparse	76.46	96s	25	14.77 watts	590.80

GPU Usage while idle: 4 watts

Thank You