Lecture Notes on Graph Neural Networks and Dataset Condensation

Swaroop Sridhar, Albert Bao, Misha Scokam

1 Background

1.1 Understanding Graphs

A **graph** $G = (V, E)$ is a mathematical structure that represents relationships between entities. This can be broken down with an example:

- In a social network, each user is a **node** (or vertex) $v \in V$
- Friendships between users are **edges** $e \in E$, where $E \subseteq V \times V$
- A user's profile information (age, interests, location) forms their **node features** $\mathbf{x}_v \in \mathbb{R}^d$

More formally, for a graph with n nodes:

- Node set: $V = \{v_1, v_2, \ldots, v_n\}$
- Feature matrix: $\mathbf{X} \in \mathbb{R}^{n \times d}$, where each row represents a node's d-dimensional feature vector
- Adjacency matrix: $\mathbf{A} \in \mathbb{R}^{n \times n}$, where a_{ij} represents the connection between nodes i and j

1.2 Graph Neural Networks (GNNs)

Traditional neural networks are designed for grid-like data (images) or sequential data (text), but they cannot directly process graph-structured data. This limitation led to the development of Graph Neural Networks.

1.2.1 Key Concepts

A GNN processes graph data through iterative **message passing**, where each node:

1. Aggregates information from its neighbors
2. Updates its representation based on the aggregated information

3. Passes its updated information to neighbors in the next iteration

Mathematically, for a node v at layer l, this process can be expressed as:

$$\mathbf{h}_v^{(l+1)} = \text{UPDATE}\left(\mathbf{h}_v^{(l)}, \text{AGGREGATE}\left(\{\mathbf{h}_u^{(l)} : u \in \mathcal{N}(v)\}\right)\right)$$

where:

- $\mathbf{h}_v^{(l)}$ is the node features at layer l
- $\mathcal{N}(v)$ represents the neighbors of node v
- AGGREGATE and UPDATE are learnable functions

1.3 Dataset Condensation

Dataset condensation addresses these challenges by creating a smaller, synthetic graph that captures the essential characteristics of the original graph. The key idea is to:

1. Reduce the number of nodes while preserving important graph properties
2. Maintain sufficient information for downstream tasks
3. Enable efficient training of GNN models

1.3.1 Benefits

- **Storage Efficiency**: Dramatic reduction in memory requirements (e.g., Reddit dataset: 435.5MB $\rightarrow$ 0.4MB)
- **Training Speed**: Faster model iteration and experimentation
- **Prototype Testing**: Easier testing of new architectures and approaches
- **Hyperparameter Tuning**: More efficient architecture search

1.3.2 Challenges

- **Structure Preservation**: Maintaining important topological properties
- **Feature Relationships**: Preserving node feature correlations
- **Balance**: Trading off between size reduction and model performance
- **Optimization**: Designing effective algorithms for graph condensation

2 Motivations

2.1 Real-World Large-Scale Graph Examples

Graphs are becoming essential for large-scale data storage, with social networks like Facebook and Instagram being prime examples where users (nodes) and their interactions (edges) form massive, computationally intensive networks. A specific example shows a citation network containing 169,343 nodes (academic papers) and 1,166,243 edges (citations between papers) [1]. Molecular graphs represent another important application, where nodes represent atoms or compounds and edges represent chemical bonds, crucial for drug discovery and chemical analysis. In e-commerce networks, users and products form bipartite graphs where edges represent interactions like purchases or ratings. The paper notes that üsers purchase products due to certain product attributes,ödemonstrating how graphs can effectively represent and analyze complex, interconnected data structures across various domains [1].

2.2 Computational Bottlenecks

Statistics show significant storage requirements for different datasets, with Reddit needing 435.5 MB and Ogbn-arxiv requiring 100.4 MB due to adjacency matrices, node feature matrices, and model parameters that scale with graph size [1]. Training efficiency becomes more challenging when models need to be retrained multiple times for incremental learning settings such as hyper-parameter tuning and neural architecture search. GNNs also face certain challenges due to their requirement for message passing across all nodes in each layer and multiple training epochs for convergence. These computational demands constrain the scale of models that can be trained, especially with limited resources.

2.3 Benefits and Challenges of Graph Condensation

Some of the benefits that come with graph condensation are storage efficiency, training speed, easier prototyping, and efficient hyperparameter tuning. For storage efficiency, dramatic reduction of the dataset can happen. For example, in the Reddit dataset, it was able to be reduced from 435.5 MB to just 0.4MB while retaining 95.3% of the original accuracy [1]. By reducing the storage, it also allows for better training speeds since smaller graphs enable faster model iteration and experimentation. It also enables easier prototyping because, by reducing the computational requirements, we can make it more feasible to test new architectures and approaches. This allows efficient hyperparameter tuning as well because of condensed graphs can be used effectively for architecture search.

2.4 Implementation Challenges

Some of the challenges in graph condensation include keeping the graph topology, maintaining node feature relationships, and balancing size reduction against

performance. Keeping the graph topology is challenging due to the complex structural information inherent in the node connections which would require complex optimization methods in order to reduce. Since nodes in real-world graphs contain rich feature information that correlates with their position and connections which is why careful maintenance of these relationships is important during the condensation. The balance between size reduction and performance presents another significant challenge as greater reduction has better computational benefits but risks losing important information needed for downstream tasks. For example, when condensing large-scale graphs like Reddit and Flickr by more than 99.9%, methods must carefully optimize both structural and feature information to maintain performance close to the original graph while achieving the benefits of condensation [1].

3 Methodology

In this section, we present the *graph condensation* framework proposed by Given a graph dataset $T = \{A, X, Y\}$, where $A \in \mathbb{R}^{N \times N}$, is the adjacency matrix, N denotes the number of nodes, $X \in \mathbb{R}^{N \times d}$ denotes the d-dimensional feature vectors corresponding to each node, $Y \in \{0, ..., C-1\}^N$ denotes the label corresponding to each feature. The objective is to learn a small, synthetic graph dataset $S = \{A', X', Y'\}$ with $A' \in R^{N' \times N'}, X' \in \mathbb{R}^{N' \times d}, Y' \in \{0, ..., C-1\}^{N'}$ where $N' << N$.

3.1 Problem Formulation

The objective could be formulated into a bi-level optimization problem as follows:

$$\min_S L(\underset{\theta_S}{\mathbf{GNN}}(\mathbf{A}, \mathbf{X}), \mathbf{Y}) \text{ s.t. } \theta_S = \arg\min_\theta L(\underset{\theta}{\mathbf{GNN}}(\mathbf{A}', \mathbf{X}'), \mathbf{Y}')$$

where $\mathbf{GNN}_\theta$ denotes the GNN model with parameter θ, and θ_S denotes the parameter of the model trained on S. L denotes the loss function, i.e. cross-entropy loss.

3.2 Gradient Matching

In order to solve the optimization problem above, an intuitive solution is solving $\hat{\mathbf{A}}, \hat{\mathbf{X}}, \hat{\mathbf{Y}} = \arg\min_{\hat{\mathbf{A}}, \hat{\mathbf{X}}, \hat{\mathbf{Y}}} L(\mathbf{GNN}_{\theta_S}(\mathbf{A}, \mathbf{X}), \mathbf{Y})$ s.t. $\theta_S = \arg\min_\theta L(\mathbf{GNN}_\theta(\hat{\mathbf{A}}, \hat{\mathbf{X}}), \hat{\mathbf{Y}})$. This is unattainable to solve since we encountered a nested loop optimization. To bypass such optimization, the author aimed to match the parameters of GNN w.r.t the original and synthetic training data. The parameter matching process for GNN can be modeled as:

$$\min_S \Sigma_{t=0}^{T-1} D(\theta_t^S, \theta_t^T)$$

, with

$$\theta_{t+1}^S = \theta_t^S - \alpha \nabla_\theta L(\underset{\theta_t^S}{\mathbf{GNN}}(\mathbf{A}', \mathbf{X}'), \mathbf{Y}') \text{ and } \theta_{t+1}^T = \theta_t^T - \alpha \nabla_\theta L(\underset{\theta_t^T}{\mathbf{GNN}}(\mathbf{A}, \mathbf{X}), \mathbf{Y})$$

Where $D(\cdot, \cdot)$ is a distance function, T is the number of steps of the whole training trajectory. If θ^S, θ^T are initialized in the same way, then we can simplify the objective as a gradient matching process as follows:

$$\min_S \Sigma_{t=0}^{T-1} D(\nabla_\theta L(\underset{\theta_t}{\mathbf{GNN}}(\mathbf{A}', \mathbf{X}'), \mathbf{Y}'), \nabla_\theta L(\underset{\theta_t}{\mathbf{GNN}}(\mathbf{A}, \mathbf{X}), \mathbf{Y}))$$

Where θ_t^S and θ_t^T can be represented as θ_t, which is trained on the small-synthetic graph.
Since jointly learning the three variables $\mathbf{A}', \mathbf{X}', \mathbf{Y}'$ would be challenging, to simplify the problem, we initialize the node labels $\mathbf{Y}'$ from the original graph by sampling and fix it.

3.3 Modeling Synthetic Graph Data

The most intuitive approach of modeling the synthetic graph data $\mathbf{A}', \mathbf{X}'$ is to treat them as free parameters. However, this would lead to huge computational challenge since the number of parameters in $\mathbf{A}'$ would grow quadratically as N' increases. Plus, this would lead to overfitting issue since it overlooks the implicit correlations between graph structure and features, leading to potential overfitting issue.
Therefore, our solution is, parameterizing the adjacency matrix $\mathbf{A}'$ with Multi-Layer Perceptron, while taking the synthetic input feature vectors $\mathbf{X}'$ as an input, as follows:

$$\mathbf{A}' = g_\Phi(\mathbf{X}'), \text{ with } \mathbf{A}'_{ij} = Sigmoid\left(\frac{\mathbf{MLP}_\Phi([x'_i; x'_j]) + \mathbf{MLP}_\Phi([x'_j; x'_i])}{2}\right)$$

Where $\mathbf{MLP}_\Phi$ is a Multi-layer perceptron parameterized with Φ and $[\cdot;\cdot]$ denotes concatenation. Since we're mostly dealing with undirected graph, we're intentionally making $\mathbf{A}'$ symmetric. Then, we rewrite our objective as:

$$\min_{\mathbf{X}', \Phi} \Sigma_{t=0}^{T-1} D(\nabla_\theta L(\underset{\theta_t}{\mathbf{GNN}}(g_\Phi(\mathbf{X}'), \mathbf{X}'), \mathbf{Y}'), \nabla_\theta L(\underset{\theta_t^T}{\mathbf{GNN}}(\mathbf{A}, \mathbf{X}), \mathbf{Y}))$$

3.4 Algorithm for Graph Condensation (GCond)

The following is a high-level algorithm for performing graph condensation using gradient matching.

Algorithm 1 GCOND for Graph Condensation

Require: Training data $T = (\mathbf{A}, \mathbf{X}, \mathbf{Y})$, pre-defined condensed labels $\mathbf{Y}'$

Initialize $\mathbf{X}'$ as node features randomly selected from each class
for $k = 0, \ldots, K-1$ **do**
  Initialize $\theta_0 \leftarrow P_0$, Initialize P_0
  for $t = 0, \ldots, T-1$ **do**
    $D' \leftarrow 0$
    for $c = 0, \ldots, C-1$ **do**
      Compute $\mathbf{A}' = g_\Phi(\mathbf{X}')$; then $S \leftarrow \{\mathbf{A}', \mathbf{X}', \mathbf{Y}'\}$
      Sample $(\mathbf{A}_c, \mathbf{X}_c, \mathbf{Y}_c) \sim T$ and $(\mathbf{A}'_c, \mathbf{X}'_c, \mathbf{Y}'_c) \sim S$
      Compute $\mathcal{L}^T = \mathcal{L}(\mathbf{GNN}_{\theta_t}(\mathbf{A}_c, \mathbf{X}_c), \mathbf{Y}_c)$ and $\mathcal{L}^S = \mathcal{L}(\mathbf{GNN}_{\theta_t}(\mathbf{A}'_c, \mathbf{X}'_c), \mathbf{Y}'_c)$
      $D' \leftarrow D' + D(\nabla_{\theta_t}\mathcal{L}^T, \nabla_{\theta_t}\mathcal{L}^S)$
    end for
    if $t\%(\tau_1 + \tau_2) < \tau_1$ **then**
      Update $\mathbf{X}' \leftarrow \mathbf{X}' - \eta_1 \nabla_{\mathbf{X}'} D'$
    else
      Update $\Phi \leftarrow \Phi - \eta_2 \nabla_\Phi D'$
    end if
    Update $\theta_{t+1} \leftarrow \mathrm{opt}_\theta(\theta_t, S, \tau_\theta)$
  end for
end for
$\mathbf{A}' \leftarrow g_\Phi(\mathbf{X}')$
return $(\mathbf{A}', \mathbf{X}', \mathbf{Y}')$ =0

4 Experimental Results and Applications of Graph Condensation

4.1 Experiment Setup

We conducted experiments using the CORA dataset to evaluate benchmark methods, including random, herding, and k-center coreset selection methods, along with the proposed gradient matching approach.

The CORA dataset comprises:

- **Original Graph:** 2,708 nodes, 5,429 edges, and 7 classes.

For the synthetic graphs generated under different condensation ratios:

- **Condensation Ratio: 1.3% (r=0.013)**
  - Synthetic Graph: 35 nodes, 10 edges.
- **Condensation Ratio: 2.6% (r=0.026)**
  - Synthetic Graph: 70 nodes, 21 edges.
- **Condensation Ratio: 5.2% (r=0.052)**

– Synthetic Graph: 140 nodes, 42 edges.

The coreset methods were evaluated for 200 epochs under each ratio. For the gradient matching method, adjusted condensation ratios (r=0.25, 0.5, and 1) were tested over 1,500 epochs to explore the correlation between the condensed graph size and the retained performance.

Reduction Ratio	**Coreset Methods (Accuracy)**			**GCond (Accuracy)**
	Random	**Herding**	**K-Center**	
1.3%	0.0897	0.5650	0.5628	0.5405
2.6%	0.0758	0.5650	0.5730	0.4071
5.2%	0.3149	0.4090	0.4090	-

Table 1: Comparison of Accuracy Across Reduction Ratios for Coreset Methods and GCond

4.2 Comparison and Analysis

The experiments on the CORA dataset for coreset methods (Random, Herding, K-Center) and the proposed gradient matching method (GCond) demonstrate the varying efficacy of these techniques across different condensation rations. The results highlight the following:

- **Random:** Achieved an accuracy of 0.3149 at a condensation ratio of 5.2% with the lower accuracies of 0.0897 and 0.0758 at 1.3% and 2.6%, respectively. This indicates the sensitivity of random selection to smaller graph sizes.
- **Herding:** Consistently performed well across condensation ratios, maintaining an accuracy of 0.5650 at both 1.3% and 2.6%, but slightly dropped to 0.4090 at 5.2%
- **K-Center:** Showed stable performance, with accuracies of 0.5628 and 0.5730 at 1.3% and 2.6%, respectively, while dropping to 0.4090 at 5.2%.
- **GCond:** Demonstrated strong results, achieving accuracies of 0.5405 at 1.3% and 0.4071 at 2.6%, with the result at 5.2% yet to be completed.

These findings reveal that structured selection methods (Herding and K-Center) outperform random sampling, particularly at lower condensation ratios, while the gradient matching approach (GCond) provides a promising alternative, especially as its performance scales with larger synthetic graph sizes.

4.3 Applications

The condensed graphs generated in these experiments offer significant advantages in various practical scenarios:

- **Efficient Training:** The reduced graph sizes facilitate faster training of Graph Neural Networks (GNNs), making them suitable for deployment on resource-constrained devices, such as mobile and edge systems.
- **Scalable Experimentation:** Researchers can rapidly iterate on GNN architectures and hyperparameters, leveraging condensed graphs to minimize computational overhead.
- **Extended Utility:** Beyond node classification, these methods can be adapted for advanced tasks like graph classification and multi-label learning, enhancing their applicability across diverse domains such as social network analysis, bioinformatics, and recommendation systems.

The insights from these results set a baseline for evaluating gradient matching methods against coreset-based techniques, underscoring their potential for achieving high accuracy while maintaining computational efficiency.

5 Conclusion

Graph Condensation represents a significant advancement in essentially making large-scale graph neural networks more practical and accessible. Through our examination of the GCond framework, we can see several insights such as:

- Traditional coreset methods (Herding, K-Center) demonstrate strong baseline performance, maintaining accuracies above 0.56 at lower condensation ratios.
- Proposed GCond framework achieves competitive results (0.5405 accuracy at 1.3% ratio), which shows promise for practical applications.
- Different condensation methods show varying sensitivity to reduction ratios. Also, this led to the structured approaches to consistently outperform the random selection.
- The framework successfully balances the trade-off between graph size reduction and model performance

Future directions would comprise of exploring performance at larger reduction ratios, extending those methods to more complex graph learning tasks, and essentially investigating the scalability of the gradient matching approaches. As the graph neural networks continue to grow in how important they are, efficient condensation techniques will become ever more important for practical applications.

References

[1] Jin, W., Zhao, L., Zhang, S., Liu, Y., Tang, J., & Shah, N. (2022). *Graph condensation for graph neural networks*. In International Conference on Learning Representations (ICLR 2022).