Learning to Reduce Graphs with Differentiable Merging

CPSC 532S Final Report

**Tim Straubinger**

April 2021

**Abstract**

Many neural network models have been proposed that operate directly on graph data structures. Existing methods are able to produce per-node feature embeddings for classification and regression and differ mainly in the methods by which they pass messages and gather features from node neighbourhoods. While learning from graph topology in this way has been explored well, the process of learning to modify graph topology in a semantically meaningful manner has received less attention. This work explores learnable reductions of graph topology by augmenting conventional graph representations and graph network models with the ability to merges together, while remaining fully-differentiable. The resulting framework is able to learn image segmentation using only merge operations, and is theoretically applicable to a large variety of problem domains where semantic reduction of graph data is desired.

# 1 Introduction

Graphs are a profoundly useful data structure that is widely applicable in many domains of computer science. Many common forms of data lend themselves directly to graph representations, such as scene graphs [Li+20], dependencies between publications [Sca+09], and social networks [PAS14; BRR18; GL16]. Other common data types, such as language [Xu+18], molecular structures [Sca+09; HYL17], music [MFP20], and images [Yan+21; Sca+09], may also be represented using graphs. Consequently, graphs have drawn much attention among deep learning researchers as a choice of representation, and many dedicated neural network architectures and training techniques have been proposed and explored.

Node-focused models [Sca+09] which produce per-node predictions differ in the techniques by which information is shared between neighbouring nodes during learning and evaluation, and the choice of technique is closely tied to the resulting model's capacity, generalizability, and awareness of the graph's topological structure. Graph neural networks [Sca+09; BRR18] learn to refine per-node features by combining features from neighbouring nodes using learnable per-node functions. Graph convolutional networks use shared functions accross nodes while still gathering features from local node neighbourhoods. Other works use random sampling procedures in order to overcome the highly-variable topology that many graphs have [PAS14; HYL17].

While these methods are able to produce useful predictions for each node in a graph, the graph topology is largely held fixed during training and evaluation, and the models being trained have no capability to change the graph topology. Learning to change a graph's topology is a formidable problem, most notably due to the discrete and non-differentiable nature of topological operations such as adding or removing nodes and edges. Yet there are many applications in which models that are able to modify graph topologies would be useful, such as segmentation, data reduction and visualization, or geometry processing.

In order to allow topological reductions on graphs in an end-to-end learnable manner, this work presents an extended graph representation and neural network architecture that allow merging operations on graphs to be performed within a fully-differentiable framework. The graph representation consists of conventional per-node feature vectors and edges represented using a dense adjacency matrix, and is extended to include a further dense matrix for

recording pairs or groups of nodes that have been merged, which is referred to here as the equality matrix.

The proposed network architecture consists of basic building blocks that aggregate features from node neighbourhoods and decide to merge adjacent nodes. By representing the adjacency and equality matrices densely, allowing continuous values, and expressing topological changes to the graph using fully-differentiable operations, the system is able to be trained end-to-end. While this incurs a high memory and computation overhead during training time, due largely to many redundant computations, the resulting network model is agnostic to the size and topology of the graph, and may later be discretized for greatly improved computational efficiency at test time.

# 2 Related Works

Conventional graph neural networks [Sca+09; BRR18] are a framework for learning on graph data by associating individual graph nodes with differentiable functions that gather per-node features according to local graph connectivity. Graph convolutional networks [KW17] are a slight modification that allows weight-sharing between per-node functions for improved generality. Stochastic learning models based on random walks [PAS14] and random sampling [HYL17] are an alternative method for gathering graph features and learning topological information. GraphSAGE [HYL17] is a random-sampling based graph neural network framework that scales well to large graphs and performs well on previously-unseen graph sizes and topologies. GraphSAGE has been applied successfully to dense image classification [Yan+21] by first converting image pixels into nodes and connecting nearby pixels with edges.

# 3 Method

An extended graph representation is presented which allows node merging operations to be represented in a fully-differentiable manner, and a basic neural network architecture that operates on this representation is proposed.

A graph containing $N$ nodes is associated with per-node feature matrix $F \in \mathbf{R}^{N \times D}$ where $D$ is the number of feature dimensions. Edges between nodes are represented using a dense adjacency matrix $A \in \mathbf{R}^{N \times N}$, such that $A_{i,j}$ is 1 if nodes $i$ and $j$ are connected to one another, and 0 if they are not. Additionally, in order to express the merging of nodes, an additional matrix $E \in \mathbf{R}^{N \times N}$, termed the equality matrix, is defined as part of the graph, such that $E_{i,j}$ is 1 if nodes $i$ and $j$ are considered to be the same node, and 0 if they are distinct. Graphs are thus represented as a triple $(E, A, F)$ containing a feature matrix, an adjacancy matrix and an equality matrix. Only undirected graphs are considered in this work, which implies that the adjacency matrix $A$ is symmetric. The equality matrix $E$ is symmetric due to the commutativity of equality and has a diagonal of all ones due to reflexivity (every node is equal to itself).

During training, the adjacency matrix $A$ and the equality matrix $E$ are permitted to contain continuous values between 0 and 1, in order to allow gradient-based optimization. When two nodes are considered to be equal, it is assumed that they have identical feature vectors and identical edges to other nodes. When pairs of nodes are to be merged, these invariants as well as the logical integrity of the equality matrix are upheld by additional procedures, referred to as coalescing, which are described in a later section.

The neural network model used in this work is node-focused, meaning that it refines per-node feature representations, and consists of a modular architecture that is able to use conventional fully-connected layers and activation functions on each node individually. Two additional network components are proposed for gathering features from node vectors and for deciding to merge pairs of nodes, which are referred to here as AGGREGATE and

MERGE. The AGGREGATE function is nearly identical to that of GraphSAGE [HYL17], except that no random sampling is used. The MERGE function is designed specifically for the proposed network architecture and is described in a later section.

It should be noted that by choosing dense matrix representations, a large memory and computational overhead is incurred for many types of graphs, which limits the complexity of graphs that may be seen during training time. Additionally, merging operations result in redundant representations and computations, which are needed for gradient-based training but may be wasteful at test time. During evaluation time, these "fuzzy" adjacency and equality values may be discretized, using thresholding for example, and sparse matrix representations may be used without redundancies in order to save on computational resources. While the training procedure is relatively resource-constrained, a highly-efficient evaluation procedure is thus theoretically possible. Additionally, because the network model is agnostic to the graph's size and topology, these optimizations mean that computations on much larger graphs than those seen at training time are possible. However, these optimizations are not explored in this work, and implementations thereof are deferred to future explorations.

## 3.1 Graph Coalescing

The introduction of the equality matrix and its interpretation allows the graph to contain contradictory pieces of information without additional precautions. For example, given three nodes $i, j$ and $k$, if we have $E_{i,j} = 1$ and $E_{j,k} = 1$, then nodes $i$ and $j$ must be equal and nodes $j$ and $k$ must also be equal. By transitivity, we expect nodes $i$ and $k$ to also be equal. Unfortunately, because the equality of nodes $i$ and $k$ is separately represented as $E_{i,k}$, it is possible without other constraints to have $E_{i,k} = 0$ which leads to a contradiction. Similar contradictions may be derived using the presence or absence of edges to other nodes. In order to make these contradictions impossible, whenever pairs of nodes are desired to be merged, the graph representation is carefully updated to preserve its integrity while still performing the desired merge operations.

Given a possibly-contradictory graph $(\hat{E}, A, F)$ where $\hat{E}$ denotes pairs of nodes that are to be merged, a valid graph $(E', A', F')$ is created in three major steps. First, the equality matrix is coalesced to gather any pair-wise equalities that are implied by transitivity. Next, the adjacency matrix is coalesced to ensure that all identical nodes have identical edges to other nodes. Finally, the feature matrix is coalesced to ensure that identical nodes have identical features.

### 3.1.1 Coalescing the equality matrix

For any pair of nodes $i$ and $j$, if there exists a third node $k$ such that nodes $i$ and $k$ are equal and nodes $j$ and $k$ are equal, then nodes $i$ and $j$ must also be equal by transitivity. This means that given a tentative, non-coalesced equality matrix $\hat{E}$, the coalesced equality matrix $E'$ is given by

$$E'_{i,j} = \max_k (\hat{E}_{i,k} * \hat{E}_{j,k}). \tag{1}$$

where multiplying elements of $\hat{E}$ behaves like a continuous logical AND, and computing the maximum behaves like a continuous logical OR. Note that this procedure implicitly prefers merging over not merging in the case of ties.

### 3.1.2 Coalescing the adjacency matrix

For any pair of nodes $i$ and $j$, if there exists a third node $k$ such that nodes $i$ and $k$ are equal and nodes $j$ and $k$ are connected by an edge, then nodes $i$ and $j$ must also be connected by an edge. Given the previous adjacency matrix $A$ and the coalesced equality matrix $E'$, the coalesced adjacency matrix $A'$ may be similarly computed as

$$A'_{i,j} = \max_k (E'_{i,k} * \hat{A}_{j,k}). \tag{2}$$

### 3.1.3 Coalescing the feature matrix

If two nodes $i$ and $j$ are equal, then their corresponding feature vectors $F_i$ and $F_j$ must also be equal. When two nodes with non-equal feature vectors are being merged, there exist many viable means by which to make those feature vectors identical. In this work, the arithmetic mean is used for simplicity, and may be computed given the coalesced equality matrix $E'$ using

$$F'_i = \frac{\sum_j (F_j * E'_{i,j})}{\sum_j (E'_{i,j})}. \tag{3}$$

In other words, each new feature vector $F'_i$ is the mean of all previous feature vectors from nodes that are equal to $i$.

## 3.2 MERGE Network Module

Given a graph state $(E, A, F)$, the MERGE neural network component allows the model to decide to merge adjacent nodes. The learnable parameters of the MERGE module are the weights of a two-layer fully-connected neural network the predicts either a value between 0 and 1 for all pairs of nodes, which may be defined as

$$Z_{i,j} = \text{sigmoid}(W_1^{\text{MERGE}} \text{relu}(W_0^{\text{MERGE}} [F_i | F_j])) \tag{4}$$

where $[a|b]$ denotes the concatenation of vectors $a$ and $b$.

This yields a matrix $Z$ of values denoting whether any pair of nodes should be merged and made equal. However, this does not yet take topology into account, nor does it necessarily produce a symmetric matrix. The matrix $Z$ is made symmetric by averaging itself with its transpose

$$\bar{Z} = \frac{Z + Z^T}{2}. \tag{5}$$

Next, we ensure that nodes which were previously equal remain equal using the previous equality matrix $E$, and that only adjacent nodes may be merged using the previous adjacency matrix $A$, yielding a tentative equality matrix $\hat{E}$ given by

$$\hat{E}_{i,j} = 1 - (1 - E_{i,j}) * (1 - (\bar{Z}_{i,j} * A_{i,j})) \tag{6}$$

where multiplication is used as a continuous extension of the logical AND, and the formula $1 - (1 - a) * (1 - b)$ is used as a continuous extension of logical OR.

This results in a tentative equality matrix $\hat{E}$ which may still possibly result in an invalid graph. Thus, the final step in the MERGE component is to coalesce the graph $(\hat{E}, A, F)$ into $(E', A', F')$ as described above. This results in a valid graph with some number of adjacent nodes having been merged.

## 3.3 AGGREGATE Network Module

The AGGREGATE procedure gathers neighbouring features and allows them to be compared with a given node's features in order to update the per-node representation. For any given node $i$, this is achieved by computing the arithmetic mean of all adjacent nodes, weighted according to the adjacency matrix, and concatenating the resulting vector $\bar{F}_i$ with the given node's feature vector $F_i$ and applying a standard fully-connected layer and non-linearity.

$$\bar{F}_i = \frac{\sum_j (F_j * A'_{i,j})}{\sum_j (A'_{i,j})},$$
$$F'_i = \text{relu}(W^{\text{AGGREGATE}} [F_i | \bar{F}_i]).$$

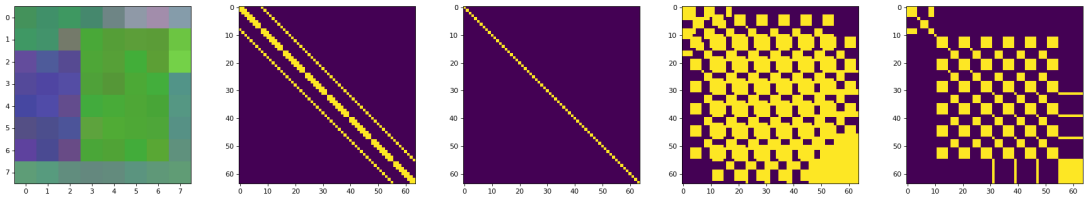

Figure 1: An example from the segmentation dataset. From left to right are the input image, the initial adjacency matrix, the initial equality matrix, the target adjacency matrix, and the target equality matrix. Note that the image is shown in its original two dimensions, but that for all matrices, both the vertical and horizontal axes correspond to the entire, unrolled image.

### 3.4 Training Considerations

As with comparable node-focused network models, the AGGREGATE module modifies the per-node feature vectors and may be trained using conventional per-node labels. But because the MERGE network module modifies the adjacency and equality matrices in a differentiable manner, these may also be used to guide gradient-descent-based training using an appropriate training objective. For example, a network can be trained to match a group of nodes using a target equality matrix where all those nodes are defined to be equal, and using a simple loss function that attempts to match the equality matrix produced by the network to the target matrix.

Furthermore, the possibility of the adjacency and equality matrices to take on continuous values is useful for training but does not necessarily have a meaningful interpretation. The network may be encouraged to create more meaningful binary values using auxiliary losses that are annealed during training, but this is not explored in this work, and the equality and adjacency values are discretized at test time using simple thresholding.

## 4 Experiments

In order to evaluate the performance of the proposed graph representation and neural network modules, a synthetic dataset of image segmentation problems was created. Image regions are randomly filled using a set of 32 randomly-generated class vectors with 32 feature channels, plus noise. These images are turned into graphs simply by creating a node for each pixel and adding edges between each pixel node and its 4 immediate neighbours. A variety of networks are trained to segment these image graphs by providing them with ground-truth equality and adjacency matrices in which nodes from all contiguous regions have been merged.

All networks begin with a per-node linear layer followed by the ReLU activation function. Networks differ in the number of MERGE and AGGREGATE modules that they contain, up to a maximum of 3 such modules in total and with at least one MERGE. All MERGE and AGGREGATE modules within the same network have distinct parameters with respect to each layer. The initial linear layer and all AGGREGATE layers are followed by a batch normalization layer. The training objective was the mean absolute error between the ground truth and predicted equality and adjacency matrices, weighted inversely by the number of node replications due to merges.

After training for 6 hours each, the networks' segmentation performance was evaluated on a withheld test set using the adjusted Rand index. Segmentation was determined by discretizing the equality matrix using a threshold value for each network that was tuned using a withheld validation set. The training set consists of 2048 examples, while the

| Model Architecture | Adjusted Rand Index ↑ |
|---|---|
| MERGE, MERGE, MERGE | 0.559 |
| MERGE, AGGREGATE, MERGE | 0.524 |
| AGGREGATE, AGGREGATE, MERGE | 0.234 |
| AGGREGATE, MERGE | 0.230 |
| MERGE | 0.230 |
| AGGREGATE, MERGE, AGGREGATE | 0.229 |
| MERGE, AGGREGATE | 0.229 |
| AGGREGATE, MERGE, MERGE | 0.189 |
| MERGE, MERGE, AGGREGATE | 0.161 |
| MERGE, MERGE | 0.157 |
| MERGE, AGGREGATE, AGGREGATE | 0.144 |

Table 1: Test results on the synthetic image segmentation task.

validation and test set both consist of 512 examples. Segmentation results are shown in the next section.

The proposed methods were implemented using the PyTorch [Pas+17] library for GPU computation and automatic differentiation. Models were trained using the AdaM optimizer [KB17] with a learning rate of 0.001 and a batch size of 32. The UBC ARC Sockeye computing platform [UBC19] was used to train and evaluate models.

# 5 Results and Discussion

Segmentation performance on the synthetic image dataset is summarized in Table 1. The best-performing model was that consisting of three consecutive MERGE stages, which simply had the most MERGE stages of any model. This makes sense, given that segmenting image by merging adjacent pixels may take several merges for large regions. It is not clear from these results what the benefit of the AGGREGATE module is, but it should be noted that AGGREGATE is proposed and implemented here for the sake of completeness of the graph neural network framework being presented, and is highly analogous to existing techniques. The purpose of this experiment is to demonstrate the behaviour and abilities of the MERGE module, and a more appropriate test of the AGGREGATE module would be one in which predicted per-node features are used to guide training. However, this was not done in this work for simplicity. An example output produced by the "MERGE, MERGE, MERGE" model is illustrated in Figure 2.

Readers may note that the images used in the synthetic dataset are tiny, at 8 by 8 pixels, and that well-studied image segmentation datasets are available. The image size constrained by the significant memory and computational requirements of the graph coalescing operations. For example, the equality matrix and adjacency matrix coalescing procedures each have a cost of order $(n^3)$, where $n$ is the number of nodes in the graph. But number of nodes in an image graph is quadratic in the image's side length, and thus the computational cost of this method is quite dramatic at order $O(l^6)$ with respect to the image side length, $l$. It may be possible to avoid this prohibitive overhead cost by using sparse matrix representations and reducing redundancies, but this is made complicated by the need for gradients to flow and the by tendency of some sparse matrix implementations to represent redundant nodes, thus incurring a possibly even higher overhead in the worst case.

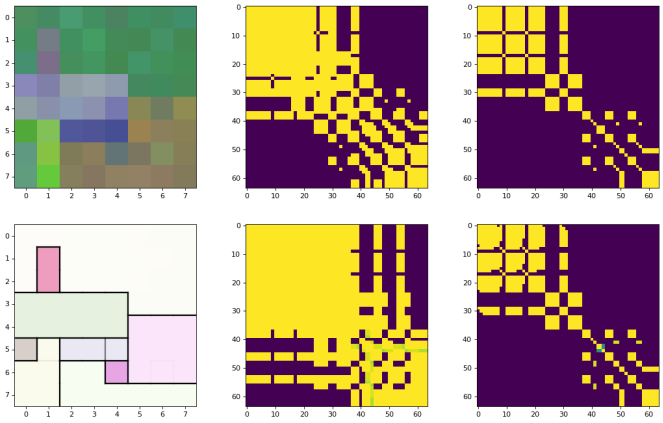

Figure 2: An example output from the "MERGE, MERGE, MERGE" segmentation model. On the top row from left to right are the input image, its target adjacency matrix, and its target equality matrix. On the bottom row are the learned node features with boundaries drawn between adjacent non-equal nodes, as well as the predicted adjacency and equality matrices.

### 5.1 Personal Reflections

Early in my explorations towards this project, I was searching for a way to train a neural network to perform arbitrary modifications on graphs, in order to be able to use them as a general latent representation over fixed-length feature vectors. I spent much time exploring ways to train networks to make discrete and non-differentiable decisions using a combination of synthetic gradients and Monte-Carlo estimation of average gradients. This lends itself nicely to dynamic auto-differentiation frameworks such as PyTorch, and early tests on discrete classification problems were promising. However, it became increasingly unclear how this would work with multiple subsequent discrete operations that choose different computational routes in a single network. In particular, it was very unclear how training would disentangle choosing separate computational routes from the performance and parameterization of any single route. Furthermore, in the context of discrete graphs operations, it did not seem possible to define loss functions characterizing the graph's topological changes that were differentiable. This led me to briefly look towards gradient-free optimization using evolution, but I was not able to train networks to solve basic discrete problems such as sorting a list. This led me to the more limited but fully-differentiable framework that is described in this report. I have not been able to prove to myself that my earlier ideas on optimizing discrete operations using synthetic gradients and Monte-Carlo sampling are impossible, and so I would like to return to these ideas in the future. However, if I were to take this course again, I would be more careful to clearly define the scope of my project early on and relate it to similar working techniques, before starting possibly futile explorations. I would also be more careful to determine whether I am asking networks to solve the label-switching problem during training, which turned out to be the case on several occasions.

## 6 Conclusion

A graph neural network framework is presented that allows models to learn to reduce the topology of graphs in a semantically meaningful way by implementing fully-differentiable node merging operations. The framework allows individual nodes to be merged as part of decision make while supporting conventional graph-based message-passing operations.

# References


[Sca+09] F. Scarselli et al. "The Graph Neural Network Model". English. In: *IEEE transactions on neural networks* 20.1 (2009), pp. 61–80.

[PAS14] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. "DeepWalk: online learning of social representations". English. In: ACM, 2014, pp. 701–710. ISBN: 9781450329569;145032956X;

[GL16] Aditya Grover and Jure Leskovec. "node2vec: Scalable Feature Learning for Networks". English. In: ACM, 2016, pp. 855–864. ISBN: 1450342329;9781450342322;

[HYL17] William L. Hamilton, Rex Ying, and Jure Leskovec. "Inductive Representation Learning on Large Graphs". English. In: (2017).

[KB17] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: `1412.6980 [cs.LG]`.

[KW17] Thomas N. Kipf and Max Welling. *Semi-Supervised Classification with Graph Convolutional Networks*. 2017. arXiv: `1609.02907 [cs.LG]`.

[Pas+17] Adam Paszke et al. "Automatic differentiation in PyTorch". In: (2017).

[BRR18] Thang Bui, Sujith Ravi, and Vivek Ramavajjala. "Neural Graph Learning: Training Neural Networks Using Graphs". English. In: ACM, 2018, pp. 64–71. ISBN: 1450355811;9781450355810;

[Xu+18] Kun Xu et al. "Graph2Seq: Graph to Sequence Learning with Attention-based Neural Networks". English. In: (2018).

[UBC19] UBC Advanced Research Computing. *UBC ARC Sockeye*. en. 2019. DOI: `10.14288/SOCKEYE`. URL: `https://arc.ubc.ca/ubc-arc-sockeye`.

[Li+20] Shuohao Li et al. "Attentive Gated Graph Neural Network for Image Scene Graph Generation". English. In: *Symmetry (Basel)* 12.4 (2020), p. 511.

[MFP20] Dirceu de Freitas Piedade Melo, Inacio d. S. Fadigas, and Hernane Borges de Barros Pereira. "Graph-based feature extraction: A new proposal to study the classification of music signals outside the time-frequency domain". English. In: *PloS one* 15.11 (2020), e0240915–e0240915.

[Yan+21] Pan Yang et al. "Hyperspectral Image Classification With Spectral and Spatial Graph Using Inductive Representation Learning Network". English. In: *IEEE journal of selected topics in applied earth observations and remote sensing* 14 (2021), pp. 791–800.