Journal of Signal Processing Systems (2024) 96:1–14
https://doi.org/10.1007/s11265-023-01897-1

Backpropagation Computation for Training Graph Attention Networks

Joe Gould[1] · **Keshab K. Parhi**[1]

Received: 3 June 2023 / Revised: 27 September 2023 / Accepted: 28 September 2023 / Published online: 16 October 2023

Abstract
Graph Neural Networks (GNNs) are a form of deep learning that have found use for a variety of problems, including the modeling of drug interactions, time-series analysis, and traffic prediction. They represent the problem using non-Euclidian graphs, allowing for a high degree of versatility, and are able to learn complex relationships by iteratively aggregating more contextual information from neighbors that are farther away. Inspired by its power in transformers, Graph Attention Networks (GATs) incorporate an attention mechanism on top of graph aggregation. GATs are considered the state of the art due to their superior performance. To learn the best parameters for a given graph problem, GATs use traditional backpropagation to compute weight updates. To the best of our knowledge, these updates are calculated in software, and closed-form equations describing their calculation for GATs aren't well known. This paper derives closed-form equations for backpropagation in GATs using matrix notation. These equations can form the basis for design of hardware accelerators for training GATs.

Keywords Neural network training · Backpropagation · Gradient computation · Graph attention networks

1 Introduction

Many complex problems can be represented by graphs, and graph neural networks (GNNs) have been used to address these issues. GNNs have been successfully applied to applications such as drug interactions [1, 2], time-series analysis [3], traffic prediction [4], neurological damage detection [5], and others. One of the best performing GNN architectures are Graph Attention Networks (GATs) [6]. These networks build upon previous models by combining the statically weighted graph convolution with an input-dependent attention mechanism to achieve higher expressive power.

Although GAT networks are powerful, they are computationally expensive, requiring processing for an attention coefficient for every edge in the graph. While the forward pass, or inference, of GAT network processing is well understood, to the best of our knowledge, closed-form expressions for backpropagation for training GAT networks have not been presented before. The main contribution of this paper is the derivation of these equations using matrix notation for both the original GAT and a similar derivative work, GATv2 [7]. These equations can enable design of accelerators for training GATs by exploiting techniques such as gradient interleaving [8].

This paper is organized as follows. Section 2 presents a brief review of GNNs and GATs. Section 3 presents the forward pass equations associated with the inference from GATs in matrix form. In Section 4, we derive the backpropagation equations for training the GATs in matrix form. Section 5 concludes the paper. Large figures that may be helpful alongside the text are found in Section of Appendix.

2 Background

2.1 Graph Neural Networks

GNNs are a broad class of architectures that developed from early work with Recurrent Neural Networks [9, 10] that use deep neural networks to perform different graph-based tasks [11, 12]. The most relevant type of GNN for understanding GATs is Graph Convolutional Network (GCN) [13], which many GNN derivatives are based on. GCN and GAT operate by performing spatial graph convolution and traditional machine learning transformations on a graph to extract meaningful information over a wider

✉ Joe Gould
gould146@umn.edu

✉ Keshab K. Parhi
parhi@umn.edu

[1] Department of Electrical and Computer Engineering, University of Minnesota, Minneapolis 55455, MN, USA

region. GCNs and GAT networks consist of layers, with each performing the convolution steps and applying a non-linear activation before outputting into the next layer.

Spatial graph convolution can be described as a form of message passing [14] on graphs. Nodes have features which are used to produce a message, the messages are transformed by and passed along edges, and the messages are gathered at edge destinations and are used to update the destination node states. This description is general, and GCN simplifies computation by limiting the convolution operation. In this modified convolution operation, edges have scalar weights which are multiplied with source node features. These weighted features are summed at the destination node, and the resultant sum is linearly transformed with learnable parameters. This convolution can be written using vector notation,

$$\mathbf{z_i} = \sum_{j \in \mathcal{N}(i)} \mathbf{A_{i,j}} \cdot \mathbf{h_j} \times \mathbf{W} \tag{1}$$

where $\mathcal{N}(i)$ is the neighborhood of node i, $\mathbf{A_{i,j}}$ is the scalar edge weight from node j to node i, $\mathbf{W}$ is the learnable weight matrix, and $\mathbf{h_i}$ and $\mathbf{z_i}$ are row vectors and the pre- and post-convolution features of node i. A similar operation is used in GAT.

As can be seen from the description above, the convolution is able to propagate information across the structure of a graph. This has the effect of allowing subsequent node representations to incorporate a wider neighborhood into their extracted features, and is the main contributor to the expressive power of GCNs and GATs. It is analogous to the widening receptive field of pixels in traditional Convolutional Neural Networks [11]. This is illustrated in Fig. 1 for a 2-layer network on a graph of 8 nodes.

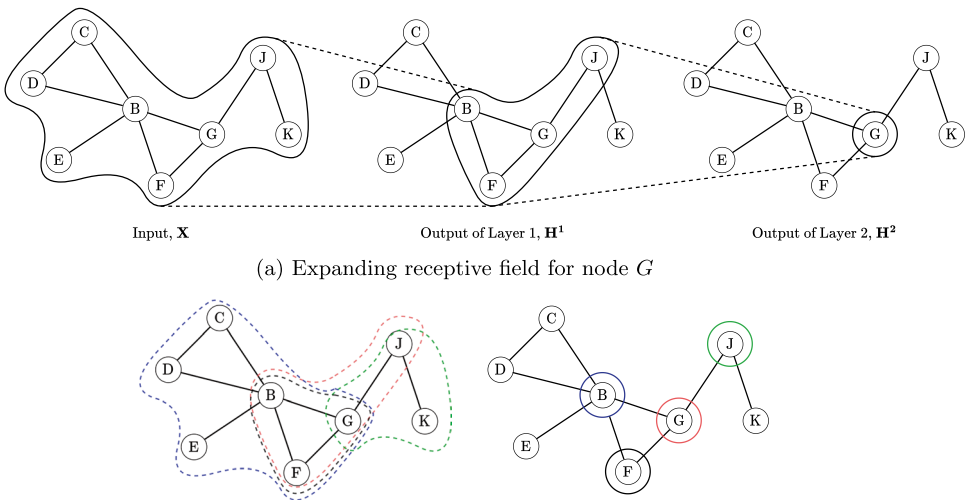
A number of hardware accelerators exist for GNN processing. The forward pass is well understood, and architectures which can support GAT inference exist [15–17]. However, architectures which support training are usually geared towards training GCN [18–20] or accelerating software solutions [21, 22] for training GAT [23]. While these methods work, it is possible a more powerful accelerator could be created by specifically targeting the computations for GAT training operations.

2.2 Neural Network Attention

Attention is a concept in machine learning that has been extensively used in applications such as machine translation [24, 25] and machine vision [26], and is the primary mechanism that transformers use for their generalized effectiveness [27]. It is based on the intuition that prioritizing the important features of an input will allow a network to better extract information from it. A network layer typically implements attention by performing a nonlinear transformation on the inputs with some learned attention parameters, and then using a softmax operation for each of the outputs. This has the effect of allowing the network to learn input features that are indicative of importance and then raising their corresponding weights to focus on them.

3 GAT Forward Pass

GAT networks operate similar to GCNs, but implement an attention mechanism to adjust edge coefficients based on node features. Like other neural network types, GATs use stacked layers to extract information from inputs and generate useful output representations. Each layer has its own set of learnable parameters, which are updated during training. Taking from existing literature on attention [27], GAT splits processing steps within each layer across multiple attention



(a) Expanding receptive field for node G

(b) 1-hop neighbors for each of the neighbors of G, i.e. B, F, J, and G itself

Figure 1 Example GNN with 2 layers on a graph of 8 nodes.

heads, which have their own attention weights during the aggregation step. Baseline GAT consists of the following processing within a layer, for each attention head:

1. Node features are linearly transformed by learnable weights to form combined features.
2. Each node's combined features are transformed into source and destination coefficients.
3. For every edge, a scalar value is computed using its source and destination coefficients. The scalar values of the edges sharing a common destination node are input to a softmax function to generate the attention coefficients of the edges.
4. GCN-like aggregation is performed using the original combined features and attention coefficients. A nonlinear activation is applied, and the result is concatenated with the results from the other attention heads.

There are currently two forms of GATs in use, the original GAT and GATv2. Hereafter, the original GAT will be referred to as GATv1, and GAT will be used to refer to the general architecture of these networks. GATv2 was proposed because of issues in GATv1's attention mechanism, which made it unable to effectively differentiate between certain types of inputs. However, because GATv1 has been in use for longer than GATv2, we present derivations for both versions. We begin by constructing equations for GAT inference in matrix form. We first define the relevant parameters in Table 1, and then follow each step of layer processing as described above.

We use the superscripts l to mean pertaining to layer l, and k, l to mean pertaining to attention head k of layer l. Non-scalar variables are denoted in bold. Vectors are assumed to be columns, unless they are taken from rows of matrices, and use lower case. When taking sub-elements of a multidimensional variable, we use subscripts to denote the row or column being taken from, and keep the same bolding and case as the variable the sub-elements are taken from. We consider graphs with directed edges, and do not require the adjacency matrix to be symmetric.

3.1 Linear Transformation of Node Features

The layer begins by linearly transforming the output of the previous layer to a new feature, not necessarily of the same size, for the current layer. GATv1 uses a single learnable weight matrix common to every node, while GATv2 has two matrices, with the result of the combination used differently depending on if a node is a source or destination of an edge.

For GATv1, this step can be written as

$$\mathbf{X^{k,l}} = \mathbf{H^{l-1}} \times \mathbf{W^{k,l}}. \tag{2}$$

GATv2 has two combined matrices, which are

$$\begin{aligned} \mathbf{X^{k,l}_{src}} &= \mathbf{H^{l-1}} \times \mathbf{W^{k,l}_{src}}, \\ \mathbf{X^{k,l}_{dst}} &= \mathbf{H^{l-1}} \times \mathbf{W^{k,l}_{dst}}. \end{aligned} \tag{3}$$

3.2 Edge Coefficient Computation

Using the combined features, GAT generates a set of source and destination coefficients that are used for computing attention coefficients. In GATv1, these coefficients are scalars obtained from performing a dot product between the learnable attention weights and the combined feature vector. In GATv2, the coefficients are vectors, and are just the combined features generated in step 1. GATv2 introduces the attention weights in the next step.

For GATv1, the scalar coefficients for node i are

$$\begin{aligned} c^{k,l}_{i,\mathrm{V1,src}} &= \mathbf{a^{k,l}_{src}}^{\mathbf{T}} \times \mathbf{X^{k,l}_{i}} = \mathbf{a^{k,l}_{src}}^{\mathbf{T}} \times \mathbf{W^{k,l}}^{\mathbf{T}} \times \mathbf{h^{k,l}_{i}}, \\ c^{k,l}_{i,\mathrm{V1,dst}} &= \mathbf{a^{k,l}_{dst}}^{\mathbf{T}} \times \mathbf{X^{k,l}_{i}} = \mathbf{a^{k,l}_{dst}}^{\mathbf{T}} \times \mathbf{W^{k,l}}^{\mathbf{T}} \times \mathbf{h^{k,l}_{i}}, \\ \mathbf{c^{k,l}_{V1,src}}^{\mathbf{T}} &= \Big\Vert_{i \in \mathcal{N}} c^{k,l}_{i,\mathrm{V1,src}}, \\ \mathbf{c^{k,l}_{V1,dst}}^{\mathbf{T}} &= \Big\Vert_{i \in \mathcal{N}} c^{k,l}_{i,\mathrm{V1,dst}}. \end{aligned} \tag{4}$$

The vector coefficients for GATv2 are

$$\begin{aligned} \mathbf{c^{k,l}_{i,V2,src}} &= \mathbf{X^{k,l}_{i,src}} = \mathbf{W^{k,l}_{src}}^{\mathbf{T}} \times \mathbf{h^{k,l}_{i}}, \\ \mathbf{c^{k,l}_{i,V2,dst}} &= \mathbf{X^{k,l}_{i,dst}} = \mathbf{W^{k,l}_{dst}}^{\mathbf{T}} \times \mathbf{h^{k,l}_{i}}, \\ \mathbf{C^{k,l}_{V2,src}}^{\mathbf{T}} &= \Big\Vert_{i \in \mathcal{N}} \mathbf{c^{k,l}_{i,V2,src}} = \mathbf{X^{k,l}_{src}}^{\mathbf{T}}, \\ \mathbf{C^{k,l}_{V2,dst}}^{\mathbf{T}} &= \Big\Vert_{i \in \mathcal{N}} \mathbf{c^{k,l}_{i,V2,dst}} = \mathbf{X^{k,l}_{dst}}^{\mathbf{T}}. \end{aligned} \tag{5}$$

3.3 Attention Coefficient Computation

Next, every edge in the graph has a scalar value computed. This is done by first adding the source and destination coefficients from the source and destination nodes of each edge, respectively. After this sum is computed, it is put through a nonlinear leaky ReLU (LReLU) operation. In GATv1, the result of this is the pre-softmax normalization attention coefficient, while in GATv2 it is a vector that is used in an inner product with the learnable attention coefficients. The result of the GATv2 inner product is its pre-softmax normalization attention coefficient. For GATv1, this can be written as

$$e^{k,l}_{i,j} = \begin{cases} \sigma_{\mathrm{L}}\left(c^{k,l}_{i,\mathrm{V1,dst}} + c^{k,l}_{j,\mathrm{V1,src}}\right) & \mathbf{A_{i,j}} \neq 0 \\ -\infty & \mathbf{A_{i,j}} = 0 \end{cases}, \tag{6}$$

Table 1 Notations for GAT Inference.

Notation		Description
N		Number of nodes in the graph
L		Layers within network
l		Index of a layer within network, $\in [1, L]$
K^l		Number of attention heads in layer l
k		Index of an attention head within layer, $\in \left[1, K^l\right]$
d^l		Output feature dimension for an attention head in layer l
k^0		Defined as 1
d^0		Defined as the feature dimension of input data
$\sigma_{\mathrm{L}}, \sigma_{\mathrm{S}}, \sigma_{\mathrm{E}}$		LReLU, softmax, and ELU nonlinear functions, respectively
$\Vert \cdots$		Concatenation of elements horizontally
$\oplus$		Defined in Eqs. (6) and (7), shown visually in Fig. 4 subfigure c
Notation	Size	Description
$\mathbf{A}$	$\mathbb{R}^{N\times N}$	Adjacency matrix of graph. The ith row corresponds to the incoming edges to node i, and the jth column corresponds to the outgoing from node j. All entries are either 0 or 1.
$\mathbf{h_i^l}$	$\mathbb{R}^{(K^l\cdot d^l)}$	Output features of node i from layer l.
$\mathbf{H^l}$	$\mathbb{R}^{N\times(K^l\cdot d^l)}$	Output features from layer l. The ith row corresponds to the feature vector of node i.
$\mathbf{W^{k,l}}, \mathbf{W^{k,l}_{src}}, \mathbf{W^{k,l}_{dst}}$	$\mathbb{R}^{(K^{l-1}\cdot d^{l-1})\times d^l}$	Learnable weight matrices for attention head k in layer l. Separate among source/destination in GATv2, common in GATv1.
$\mathbf{X^{k,l}}, \mathbf{X^{k,l}_{src}}, \mathbf{X^{k,l}_{dst}}$	$\mathbb{R}^{N\times d^l}$	Combined feature matrices for attention head k in layer l, from their respective weight matrices.
$\mathbf{a^{k,l}}, \mathbf{a^{k,l}_{src}}, \mathbf{a^{k,l}_{dst}}$	$\mathbb{R}^{d^l}$	Learnable source/destination attention weights for attention head k in layer l. Separate among source/destination in GATv1, common in GATv2.
$c^{k,l}_{i,\mathrm{V1,src}}, c^{k,l}_{i,\mathrm{V1,dst}}$	$\mathbb{R}$	GATv1: Source/destination scalar coefficients for node i in attention head k of layer l.
$\mathbf{c^{k,l}_{V1,src}}, \mathbf{c^{k,l}_{V1,dst}}$	$\mathbb{R}^N$	and collected into a vector.
$\mathbf{c^{k,l}_{i,V2,src}}, \mathbf{c^{k,l}_{i,V2,dst}}$	$\mathbb{R}^{d^l}$	GATv2: Source/destination vector coefficients for node i in attention head k of layer l.
$\mathbf{C^{k,l}_{V2,src}}, \mathbf{C^{k,l}_{V2,dst}}$	$\mathbb{R}^{N\times d^l}$	and collected into a matrix.
$e^{k,l}_{i,j}, \alpha^{k,l}_{i,j}$	$\mathbb{R}$	Pre- and post-softmax normalization attention coefficients for the edge outgoing from node j to node i.
$\mathbf{L^{k,l}}, \mathbf{S^{k,l}}$	$\mathbb{R}^{N\times N}$	Pre- and post-softmax attention matrices. Entry (i,j) is the corresponding weight from node j to node i.
$\mathbf{Z^{k,l}}$	$\mathbb{R}^{N\times(K^l\times d^l)}$	Pre-activation output features for layer l.

and in GATv2 as

$$e^{k,l}_{i,j} = \begin{cases} \mathbf{a^{k,l}} \times \sigma_{\mathrm{L}}\left(\mathbf{c^{k,l}_{i,V2,dst}} + \mathbf{c^{k,l}_{j,V2,src}}\right) & \mathbf{A_{i,j}} \neq 0 \\ -\infty & \mathbf{A_{i,j}} = 0 \end{cases}. \tag{7}$$

We denote the addition operation with $\oplus$. In the case of GATv1, $\mathbf{c^{k,l}_{V1,src}}$ and $\mathbf{c^{k,l}_{V1,dst}}$ are vectors, and the operation generates a matrix by performing an outer sum. In GATv2, $\mathbf{C^{k,l}_{V2,dst}}$ and $\mathbf{C^{k,l}_{V2,src}}$ are matrices, and the operation generates a 3-dimensional tensor by performing the outer sum on the columns of the matrices. Using this notation, we can write the resulting attention matrices as

$$\mathbf{L^{k,l}} = \sigma_{\mathrm{L}}\left(\mathbf{c^{k,l}_{V1,dst}} \oplus \mathbf{c^{k,l}_{V1,src}}\right) \odot \mathbf{A} \tag{8}$$

for GATv1, and for GATv2 as

$$\mathbf{L^{k,l}} = \left(\mathbf{a^{k,l}} \times \sigma_{\mathrm{L}}\left(\mathbf{C^{k,l}_{V2,dst}} \oplus \mathbf{C^{k,l}_{V2,src}}\right)\right) \odot \mathbf{A}. \tag{9}$$

Finally, in both GATv1 and GATv2, these coefficients are put through a softmax, grouped by common destinations, to normalize them to the same dynamic range. These matrices have an identical nonzero pattern to the adjacency matrix $\mathbf{A}$. This results in the $\mathbf{L}$ and $\mathbf{S}$ matrices that have their (i, j) th entries be the pre- and post-softmax attention coefficient for the edge outgoing from node j to node i as

$$\mathbf{S^{k,l}_i} = \sigma_{\mathrm{S}}\left(\mathbf{L^{k,l}_i}\right). \tag{10}$$

3.4 GCN-like Aggregation and Activation

The final step in the forward pass is to propagate the combined node features using the final attention coefficient on each edge. This propagation is identical to GCN, with the adjacency matrix replaced with the post-softmax attention matrix. By performing matrix multiplication in this way, the

combined features of all of the in-neighbors of each node are summed together, weighed by their attention. In GATv1, this is given by

$$\mathbf{Z^{k,l}} = \mathbf{S^{k,l}} \times \mathbf{X^{k,l}}. \tag{11}$$

This is identical to GATv2, with the features combined with the source weights matrix in the aggregation, written as

$$\mathbf{Z^{k,l}} = \mathbf{S^{k,l}} \times \mathbf{X^{k,l}_{src}}. \tag{12}$$

Finally, the combined and aggregated features are put through a nonlinear activation. We use σ_{E} to represent the exponential linear unit (ELU) activation function. The resulting features for each attention head are concatenated together to form the output of the layer:

$$\mathbf{H^{l+1}} = \Big\Vert_{k=1}^{K^l} \sigma_{\mathrm{E}}\big(\mathbf{Z^{k,l}}\big). \tag{13}$$

In the case of the output layer, multiple attention heads have their results averaged instead of being concatenated, and a nonlinear operation like softmax is applied on the resulting node features. This can be written as

$$\mathbf{H^L_i} = \sigma_{\mathrm{S}}\left(\frac{1}{K^L} \cdot \sum_{k=1}^{K^L} \mathbf{Z^{k,L}_i}\right). \tag{14}$$

We summarize the forward pass in a data flow graph (DFG) in Fig. 2. An illustrative example of each step for GATv1 and GATv2 in the forward pass is shown for a single layer's attention head in Fig. 4. The example assumes an input feature size of 5 and an output feature size of the layer's attention heads of 4.

4 GAT Backward Pass

We now derive the equations for the backpropagation for GAT's learnable parameters, i.e., equations for gradients of the loss with respect to weight matrices $\left(\mathbf{W^{k,l}}, \mathbf{W^{k,l}_{src}}, \mathbf{W^{k,l}_{dst}}\right)$, attention weights $\left(\mathbf{a^{k,l}_{src}}, \mathbf{a^{k,l}_{dst}}, \mathbf{a^{k,l}}\right)$, and feature inputs $\left(\mathbf{H^l}\right)$ for each layer. This will follow the same structure as the forward pass, starting with the input to the layers during backpropagation and computing the gradients associated with that layer. A summary of the notations for different variables relevant to the backward pass is provided in Table 2.

4.1 Gradients with Respect to Activation

We first compute the gradients of the loss with respect to the pre-activation outputs using the gradients input to the layer. The gradient can be computed as

$$\boldsymbol{\delta}'^{\mathbf{k,l}} = \sigma'_{\mathrm{E}}\big(\mathbf{Z^{k,l}}\big) \odot \boldsymbol{\delta^{k,l}}. \tag{15}$$

For the last layer in the network, a softmax activation function is used. Its derivative is well known, and the derivative of any vector through a softmax can be described by a symmetric gradient matrix,

$$\mathbf{u} = \sigma_{\mathrm{S}}(\mathbf{v}),$$
$$\frac{\partial u_i}{\partial v_j} = \begin{cases} u_i - u_i^2 & i = j \\ -\big(u_i \cdot u_j\big) & i \neq j \end{cases},$$
$$\frac{\partial \mathbf{u}}{\partial \mathbf{v}} = \mathrm{diag}(\mathbf{u}) - \mathbf{u} \times \mathbf{u^T},$$

where the (i, j)th entry in the $\frac{\partial \mathbf{u}}{\partial \mathbf{v}}$ matrix is the gradient contributed by the jth element of the input to the ith element of the output. In this case, $\mathbf{v}$ and $\mathbf{u}$ are the rows of the $\mathbf{Z^L}$ and $\mathbf{H^L}$ matrices. The gradient matrix described above can be computed for each node, and the gradient of the loss with respect to the pre-softmax features for that node can be computed by multiplying this matrix with the gradient of the loss with respect to the post-softmax features. This is shown as

$$\frac{\partial \mathbf{H^L_i}}{\partial \mathbf{Z^L_i}} = \mathrm{diag}\left(\mathbf{H^L_i}\right) - \mathbf{H^L_i}^{\mathbf{T}} \times \mathbf{H^L_i}, \tag{16}$$

$$\boldsymbol{\delta}'^{\mathbf{L}}_{\mathbf{i}} = \frac{\partial \mathbf{H^L_i}}{\partial \mathbf{Z^L_i}} \times \boldsymbol{\delta^L_i}. \tag{17}$$

For the rest of the backward pass, there is no difference between layers if we scale this pre-softmax gradient by K^L to account for the averaging step, written as

$$\boldsymbol{\delta}'^{\mathbf{k,L}} = \frac{1}{K^L} \cdot \boldsymbol{\delta}'^{\mathbf{L}}. \tag{18}$$

It is important to note that the matrices involved in this step are dense, unlike the matrix of attention coefficients. Because of the complexity of the softmax derivative and because it only needs to be computed for the output layer, $\boldsymbol{\delta}'^{\mathbf{L}}$ should be computed outside of a dedicated accelerator.

4.2 Gradients with Respect to Attention

We now look at the attention mechanism and how the gradient of the loss with respect to its inputs can be computed. The first step is to compute the gradient of the loss with respect post-softmax attention coefficients, through the aggregation, and then the pre-softmax edge coefficients, which has a form identical to the softmax derivative described before. This can be written as

$$\frac{\partial \mathcal{L}}{\partial \mathbf{S^{k,l}}} = \boldsymbol{\delta}'^{\mathbf{k,l}} \times \mathbf{X^{k,l}}^{\mathbf{T}}, \tag{19}$$

$$\frac{\partial \mathbf{S}_{\mathbf{i}}^{\mathbf{k,l}}}{\partial \mathbf{L}_{\mathbf{i}}^{\mathbf{k,l}}} = \text{diag}\left(\mathbf{S}_{\mathbf{i}}^{\mathbf{k,l}}\right) - \mathbf{S}_{\mathbf{i}}^{\mathbf{k,l}^{\mathbf{T}}} \times \mathbf{S}_{\mathbf{i}}^{\mathbf{k,l}}, \tag{20}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{L}_{\mathbf{i}}^{\mathbf{k,l}}} = \frac{\partial \mathbf{S}_{\mathbf{i}}^{\mathbf{k,l}}}{\partial \mathbf{L}_{\mathbf{i}}^{\mathbf{k,l}}} \times \left(\frac{\partial \mathcal{L}}{\partial \mathbf{S}_{\mathbf{i}}^{\mathbf{k,l}}}\right)^{\mathbf{T}} = \sigma'_{\mathrm{S}}\left(\frac{\partial \mathcal{L}}{\partial \mathbf{S}_{\mathbf{i}}^{\mathbf{k,l}}}\right). \tag{21}$$

Because of the sparsity in $\mathbf{S}^{\mathbf{k,l}}$, $\frac{\partial \mathbf{S}_{\mathbf{i}}^{\mathbf{k,l}}}{\partial \mathbf{L}_{\mathbf{i}}^{\mathbf{k,l}}}$ will only have nonzeros in rows/columns which correspond to the edges for node i. The complete loss with respect to the pre-softmax attentions, $\frac{\partial \mathcal{L}}{\partial \mathbf{L}^{\mathbf{k,l}}}$, also has the same sparsity pattern as the adjacency matrix.

Now, because of the differences in GATv1 and GATv2, we separate the derivation for the rest of the attention backward pass for each of the two versions.

4.2.1 GATv1 Attention Gradient

The rest of the backward pass depends on the result of the LReLU that was used to compute the pre-softmax weight matrix. Since the input to the LReLU was a scalar, we define a $\mathbf{C'}^{\mathbf{k,l}}$ matrix to hold values from the derivative of the LReLU, and then define the gradient with respect to its input as $\mathbf{\Delta}^{\mathbf{k,l}}$, which can be written as

$$\mathbf{C'}_{\mathrm{V1}\,\mathbf{i,j}}^{\quad\mathbf{k,l}} = \sigma'_{\mathrm{L}}\left(c_{i,\mathrm{V1,dst}}^{k,l} + c_{j,\mathrm{V1,src}}^{k,l}\right), \tag{22}$$

$$\mathbf{C'}_{\mathrm{V1}}^{\mathbf{k,l}} = \sigma'_{\mathrm{L}}\left(\mathbf{c}_{\mathrm{V1,dst}}^{\mathbf{k,l}} \oplus \mathbf{c}_{\mathrm{V1,src}}^{\mathbf{k,l}}\right), \tag{23}$$

$$\mathbf{\Delta}_{\mathrm{V1}}^{\mathbf{k,l}} = \mathbf{C'}_{\mathrm{V1}}^{\mathbf{k,l}} \odot \frac{\partial \mathcal{L}}{\partial \mathbf{L}^{\mathbf{k,l}}}. \tag{24}$$

This $\mathbf{\Delta}$ matrix is never used as a matrix, but instead is summed along its rows and columns. Depending on which dimension it is summed over, it will correspond to gradient passed along source or destination edges. We can define the following vectors

$$\begin{aligned} \mathbf{\Sigma}_{\mathrm{V1,dst}}^{\mathbf{k,l}} &= \mathrm{Sum}\left(\mathbf{\Delta}^{\mathbf{k,l}}, \mathrm{rows}\right), \\ \mathbf{\Sigma}_{\mathrm{V1,src}}^{\mathbf{k,l}} &= \mathrm{Sum}\left(\mathbf{\Delta}^{\mathbf{k,l}}, \mathrm{cols}\right). \end{aligned} \tag{25}$$

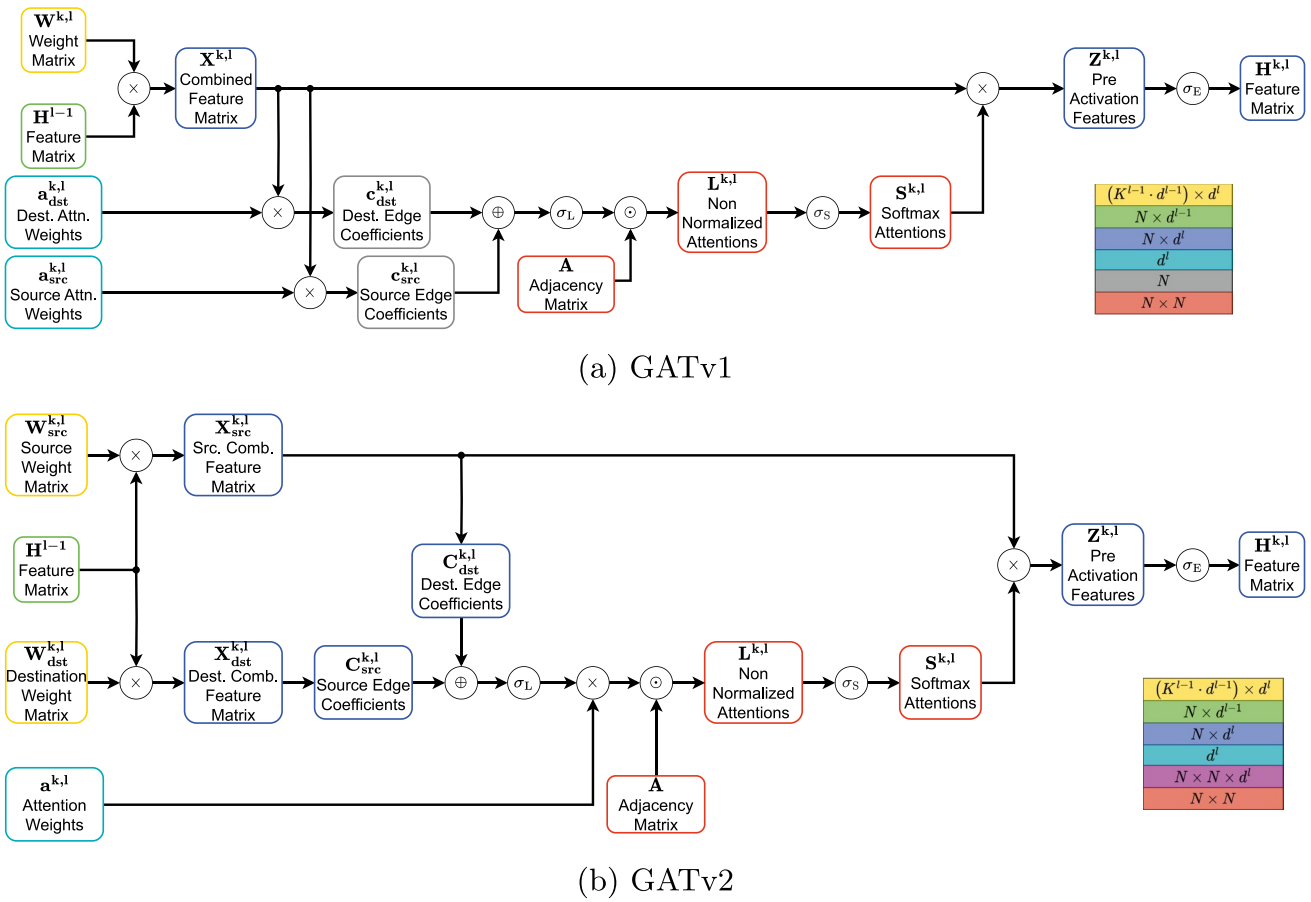


Figure 2 DFG for the forward pass of a single attention head within a hidden layer. Each box is colored according to the shape of the matrix at that point in the computation.

Table 2 Notations for GAT Training.

Notation		Description
$\sigma'_{\mathrm{L}}, \sigma'_{\mathrm{S}}, \sigma'_{\mathrm{E}}$		Derivatives of LReLU, softmax, and ELU, respectively
$\odot$		Hadamard (element-wise) vector or matrix product
Notation	Size	Description
$\mathcal{L}$	$\mathbb{R}^{N \times d^L}$	Loss from output of GAT network
$\boldsymbol{\delta}^{\mathbf{l}}, \boldsymbol{\delta}'^{\mathbf{l}}$	$\mathbb{R}^{N \times (K^l \cdot d^l)}$	Gradient into layer l before and after being transformed by the activation derivative. Equivalent to $\frac{\partial \mathcal{L}}{\partial \mathbf{H}^{\mathbf{l}}}$ and $\frac{\partial \mathcal{L}}{\partial \mathbf{Z}^{\mathbf{l}}}$, respectively.
$\frac{\partial \mathcal{L}}{\partial \mathbf{L}^{\mathbf{k,l}}}, \frac{\partial \mathcal{L}}{\partial \mathbf{S}^{\mathbf{k,l}}}$	$\mathbb{R}^{N \times N}$	Loss with respect to pre- and post-softmax attention coefficients
$\mathbf{C'}^{\mathbf{k,l}}_{\mathrm{V1}}$	$\mathbb{R}^{N \times N}$	GATv1: Matrix holding LReLU derivative of the source/destination coefficient sums from the forward pass
$\mathbf{C'}^{\mathbf{k,l}}_{\mathrm{V2}}$	$\mathbb{R}^{N \times N \times d^l}$	GATv2: Matrix of vectors holding LReLU derivative of the source/destination coefficient sums from the forward pass
$\boldsymbol{\Delta}^{\mathbf{k,l}}_{\mathrm{V1}}$	$\mathbb{R}^{N \times N}$	GATv1: Scalar loss with respect to edges
$\boldsymbol{\Delta}^{\mathbf{k,l}}_{\mathrm{V2}}$	$\mathbb{R}^{N \times N \times d^l}$	GATv2: Vector loss with respect to edges
$\boldsymbol{\Sigma}^{\mathbf{k,l}}_{\mathrm{V1,dst}}, \boldsymbol{\Sigma}^{\mathbf{k,l}}_{\mathrm{V1,src}}$	$\mathbb{R}^{N}$	GATv1: Scalar losses with respect to nodes being destinations and sources of edges.
$\boldsymbol{\Sigma}^{\mathbf{k,l}}_{\mathrm{V2,dst}}, \boldsymbol{\Sigma}^{\mathbf{k,l}}_{\mathrm{V2,src}}$	$\mathbb{R}^{N \times d^l}$	GATv2: Vector losses with respect to nodes being destinations and sources of edges
$\frac{\partial \mathcal{L}}{\partial \mathbf{a}^{\mathbf{k,l}}_{\mathrm{dst}}}, \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{\mathbf{k,l}}_{\mathrm{src}}}, \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{\mathbf{k,l}}}$	$\mathbb{R}^{d^l}$	Loss with respect to the attention weights of head k of layer l
$\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{\mathbf{k,l}}}, \frac{\partial \mathcal{L}}{\partial \mathbf{W}^{\mathbf{k,l}}_{\mathrm{dst}}}, \frac{\partial \mathcal{L}}{\partial \mathbf{W}^{\mathbf{k,l}}_{\mathrm{src}}}$	$\mathbb{R}^{d^{l-1} \times d^l}$	Loss with respect to weights of head k of layer l

We can then derive the gradient of the loss with respect to the attention weights using these vectors

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{\mathbf{k,l}}_{\mathrm{dst}}} &= \mathbf{X}^{\mathbf{k,l}^{\mathbf{T}}} \times \boldsymbol{\Sigma}^{\mathbf{k,l}}_{\mathrm{V1,dst}}, \\ \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{\mathbf{k,l}}_{\mathrm{src}}} &= \mathbf{X}^{\mathbf{k,l}^{\mathbf{T}}} \times \boldsymbol{\Sigma}^{\mathbf{k,l}}_{\mathrm{V1,src}}. \end{aligned} \tag{26}$$

These source and edge gradient vectors will also be used in computing the gradient of the loss with respect to input features and weights.

4.2.2 GATv2 Attention Gradient

The derivation for GATv2 is similar to GATv1, though here the $\mathbf{C'}^{\mathbf{k,l}}$ matrix is instead a 3-dimensional tensor, because the attention coefficients are vectors instead of scalars. Effectively, it is a matrix identical in behavior to the GATv1 case, with the scalar elements replaced by vectors. This is written as

$$\mathbf{C'}^{\mathbf{k,l}}_{\mathrm{V2}\mathbf{i,j}} = \mathbf{a}^{\mathbf{k,l}} \odot \sigma'_{\mathrm{L}}\left(\mathbf{c}^{\mathbf{k,l}}_{\mathbf{i},\mathrm{V2,dst}} + \mathbf{c}^{\mathbf{k,l}}_{\mathbf{j},\mathrm{V2,src}}\right), \tag{27}$$

$$\begin{aligned} \boldsymbol{\Delta}^{\mathbf{k,l}}_{\mathrm{V2}\mathbf{i,j}} &= \frac{\partial \mathcal{L}}{\partial \mathbf{L}^{\mathbf{k,l}}_{\mathbf{i,j}}} \cdot \mathbf{C'}^{\mathbf{k,l}}_{\mathrm{V2}\mathbf{i,j}}, \\ \boldsymbol{\Delta}^{\mathbf{k,l}}_{\mathrm{V2}} &= \mathbf{C'}^{\mathbf{k,l}}_{\mathrm{V2}} \odot \frac{\partial \mathcal{L}}{\partial \mathbf{L}^{\mathbf{k,l}}}, \end{aligned} \tag{28}$$

$$\begin{aligned} \boldsymbol{\Sigma}^{\mathbf{k,l}}_{\mathrm{V2,dst}} &= \mathrm{Sum}\left(\boldsymbol{\Delta}^{\mathbf{k,l}}, \mathrm{rows}\right), \\ \boldsymbol{\Sigma}^{\mathbf{k,l}}_{\mathrm{V2,src}} &= \mathrm{Sum}\left(\boldsymbol{\Delta}^{\mathbf{k,l}}, \mathrm{cols}\right). \end{aligned} \tag{29}$$

Instead of the $\boldsymbol{\Sigma}^{\mathbf{k,l}}$ tensors being vectors, they are matrices, owing to the vector elements in the $\Delta^{k,l}$ tensor. Finally, the loss with respect to attention weights can be derived as

$$\frac{\partial \mathcal{L}}{\partial \mathbf{a}^{\mathbf{k,l}}} = \sum_{i,j} \frac{\partial \mathcal{L}}{\partial \mathbf{L}^{\mathbf{k,l}}_{\mathbf{i,j}}} \cdot \sigma_{\mathrm{L}}\left(\mathbf{c}^{\mathbf{k,l}}_{\mathbf{i},\mathrm{V2,dst}} + \mathbf{c}^{\mathbf{k,l}}_{\mathbf{j},\mathrm{V2,src}}\right). \tag{30}$$

Unlike GATv1, this does not depend on the $\Sigma^{k,l}$ matrices, instead only depends on $\frac{\partial \mathcal{L}}{\partial \mathbf{L}^{\mathbf{k,l}}}$. This is because the attention weights in GATv2 are not used until after the LReLU operation.

4.3 Gradients Associated with Combination

The final part of the backpropagation is similar between GATv1 and GATv2, consisting of combining the gradients for the feature and weight matrices with the contributions from the attention mechanism and the aggregation step. Because there are differences, we will again look at the two GAT versions separately.

4.3.1 GATv1 Weight and Feature Gradient

Both the feature and weight matrices can be written as a combination of gradient contributions from three sources: the gradients along the source and destination edges and the gradient due to weighted aggregation. The loss with respect to input features can be written as

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial \mathbf{H^l}} = \sum_{k=1}^{K^l} \Big(&\Big(\mathbf{\Sigma^{k,l}_{V1,dst}} \times \mathbf{a^{k,l}_{dst}}^{\mathbf{T}} \times \mathbf{W^{k,l}}^{\mathbf{T}}\Big) \\ &+ \Big(\mathbf{\Sigma^{k,l}_{V1,src}} \times \mathbf{a^{k,l}_{src}}^{\mathbf{T}} \times \mathbf{W^{k,l}}^{\mathbf{T}}\Big) \\ &+ \Big(\mathbf{S^{k,l}}^{T} \times \mathbf{\delta'^{k,l}} \times \mathbf{W^{k,l}}^{\mathbf{T}}\Big)\Big),\end{aligned} \tag{31}$$

$$\begin{aligned}= \sum_{k=1}^{K^l} \Big[&\Big(\mathbf{\Sigma^{k,l}_{V1,dst}} \times \mathbf{a^{k,l}_{dst}}^{\mathbf{T}}\Big) \\ &+ \Big(\mathbf{\Sigma^{k,l}_{V1,src}} \times \mathbf{a^{k,l}_{src}}^{\mathbf{T}}\Big) \\ &+ \Big(\mathbf{S^{k,l}}^{\mathbf{T}} \times \mathbf{\delta'^{k,l}}\Big)\Big] \times \mathbf{W^{k,l}}^{\mathbf{T}},\end{aligned} \tag{32}$$

$$= \mathbf{\delta^{l-1}}. \tag{33}$$

The loss with respect to weights can be written similarly

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial \mathbf{W^{k,l}}} = \Big(&\Big(\mathbf{H^{l-1}}^{\mathbf{T}} \times \mathbf{\Sigma^{k,l}_{V1,dst}} \times \mathbf{a^{k,l}_{dst}}^{\mathbf{T}}\Big) \\ &+ \Big(\mathbf{H^{l-1}}^{\mathbf{T}} \times \mathbf{\Sigma^{k,l}_{V1,src}} \times \mathbf{a^{k,l}_{src}}^{\mathbf{T}}\Big) \\ &+ \Big(\mathbf{H^{l-1}}^{\mathbf{T}} \times \mathbf{S^{k,l}}^{\mathbf{T}} \times \mathbf{\delta'^{k,l}}\Big)\Big),\end{aligned} \tag{34}$$

$$\begin{aligned}= \mathbf{H^{l-1}}^{\mathbf{T}} \times \Big[&\Big(\mathbf{\Sigma^{k,l}_{V1,dst}} \times \mathbf{a^{k,l}_{dst}}^{\mathbf{T}}\Big) \\ &+ \Big(\mathbf{\Sigma^{k,l}_{V1,src}} \times \mathbf{a^{k,l}_{src}}^{\mathbf{T}}\Big) \\ &+ \Big(\mathbf{S^{k,l}}^{\mathbf{T}} \times \mathbf{\delta'^{k,l}}\Big)\Big].\end{aligned} \tag{35}$$

As can be seen from the equations, the terms in the square brackets are shared among both the gradients.

4.3.2 GATv2 Weight and Feature Gradient

The GATv2 equations are similar to GATv1, but because there are source and destination weight matrices, the three terms used to compute the feature gradient are split among the two weight matrix gradients. In GATv2, the loss with respect to input features is given by

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial \mathbf{H^{l-1}}} = \sum_{k=1}^{K^l} \Big(&\Big(\mathbf{\Sigma^{k,l}_{V2,dst}} \times \mathbf{W^{k,l}_{dst}}^{\mathbf{T}}\Big) \\ &+ \Big(\mathbf{\Sigma^{k,l}_{V2,src}} \times \mathbf{W^{k,l}_{src}}^{\mathbf{T}}\Big) \\ &+ \Big(\mathbf{S^{k,l}}^{T} \times \mathbf{\delta'^{k,l}} \times \mathbf{W^{k,l}_{src}}^{\mathbf{T}}\Big)\Big),\end{aligned} \tag{36}$$

$$\begin{aligned}= \sum_{k=1}^{K^l} \Big(&\Big(\mathbf{\Sigma^{k,l}_{V2,dst}} \times \mathbf{W^{k,l}_{dst}}^{\mathbf{T}}\Big) \\ &+ \Big[\mathbf{\Sigma^{k,l}_{V2,src}} + \Big(\mathbf{S^{k,l}}^{\mathbf{T}} \times \mathbf{\delta'^{k,l}}\Big)\Big] \times \mathbf{W^{k,l}_{src}}^{\mathbf{T}}\Big)\end{aligned}, \tag{37}$$

$$= \mathbf{\delta^{l-1}}. \tag{38}$$

Then, the two weight matrix gradients are

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial \mathbf{W^{k,l}_{dst}}} &= \mathbf{H^{l-1}}^{\mathbf{T}} \times \mathbf{\Sigma^{k,l}_{V2,dst}}, \\ \frac{\partial \mathcal{L}}{\partial \mathbf{W^{k,l}_{src}}} &= \Big(\Big(\mathbf{H^{l-1}}^{\mathbf{T}} \times \mathbf{\Sigma^{k,l}_{V2,src}}\Big) + \Big(\mathbf{H^{l-1}}^{\mathbf{T}} \times \mathbf{S^{k,l}}^{\mathbf{T}} \times \mathbf{\delta'^{k,l}}\Big)\Big), \\ &= \mathbf{H^{l-1}}^{\mathbf{T}} \times \Big[\mathbf{\Sigma^{k,l}_{V2,src}} + \Big(\mathbf{S^{k,l}}^{\mathbf{T}} \times \mathbf{\delta'^{k,l}}\Big)\Big].\end{aligned} \tag{39}$$

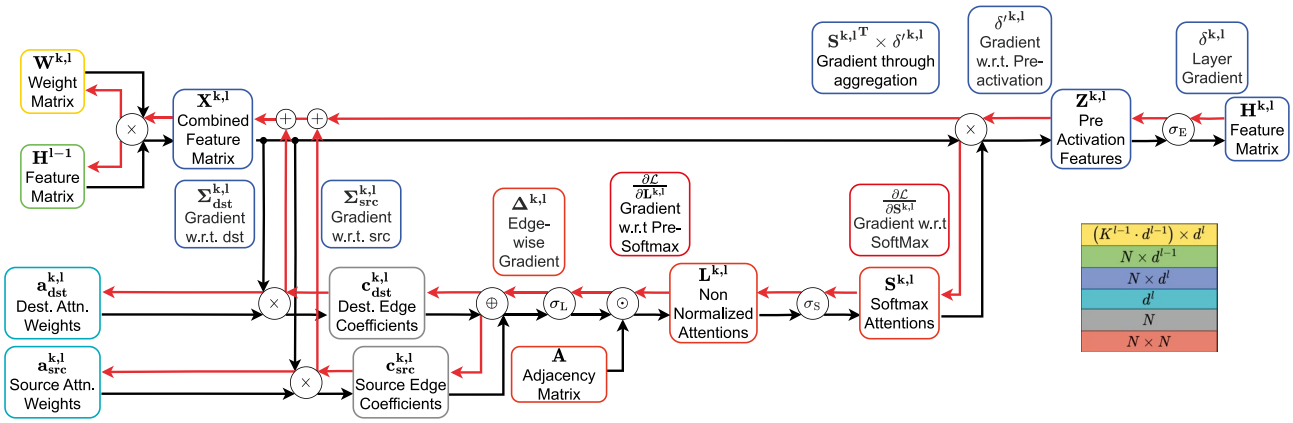
The shared terms are once again shown in square brackets.

To summarize the backward pass pictorially, we modify the forward pass DFGs from Fig. 2 to show the dependencies during backpropagation, shown in Fig. 3. The example from the forward pass is used again to show the steps of backpropagation in Fig. 5.

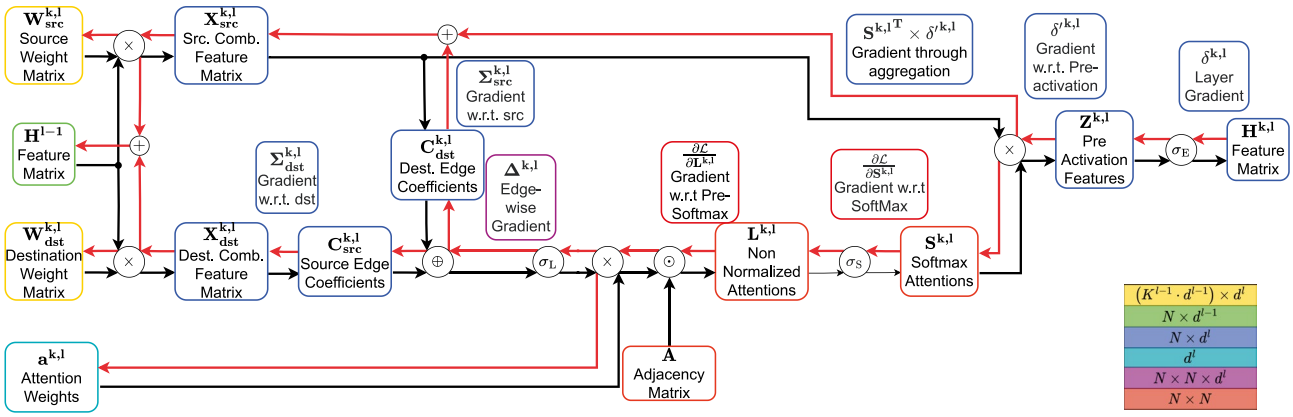
5 Conclusion

This paper has presented closed-form matrix equations for the training of Graph Attention Networks. These equations explicitly illustrate the dependencies among various parts of the computations. These dependencies can be exploited to design accelerators for training GATs using techniques such as variable reuse, gradient interleaving and inter-layer pipelining [28].

Appendix. Supplementary Figures



(a) GATv1



(b) GATv2

Figure 3 DFG for the forward and backward pass of a single attention head within a hidden layer. Backward pass matrices are shown adjacent to data flow. Colors of the boxes correspond to the shape of their respective matrices.

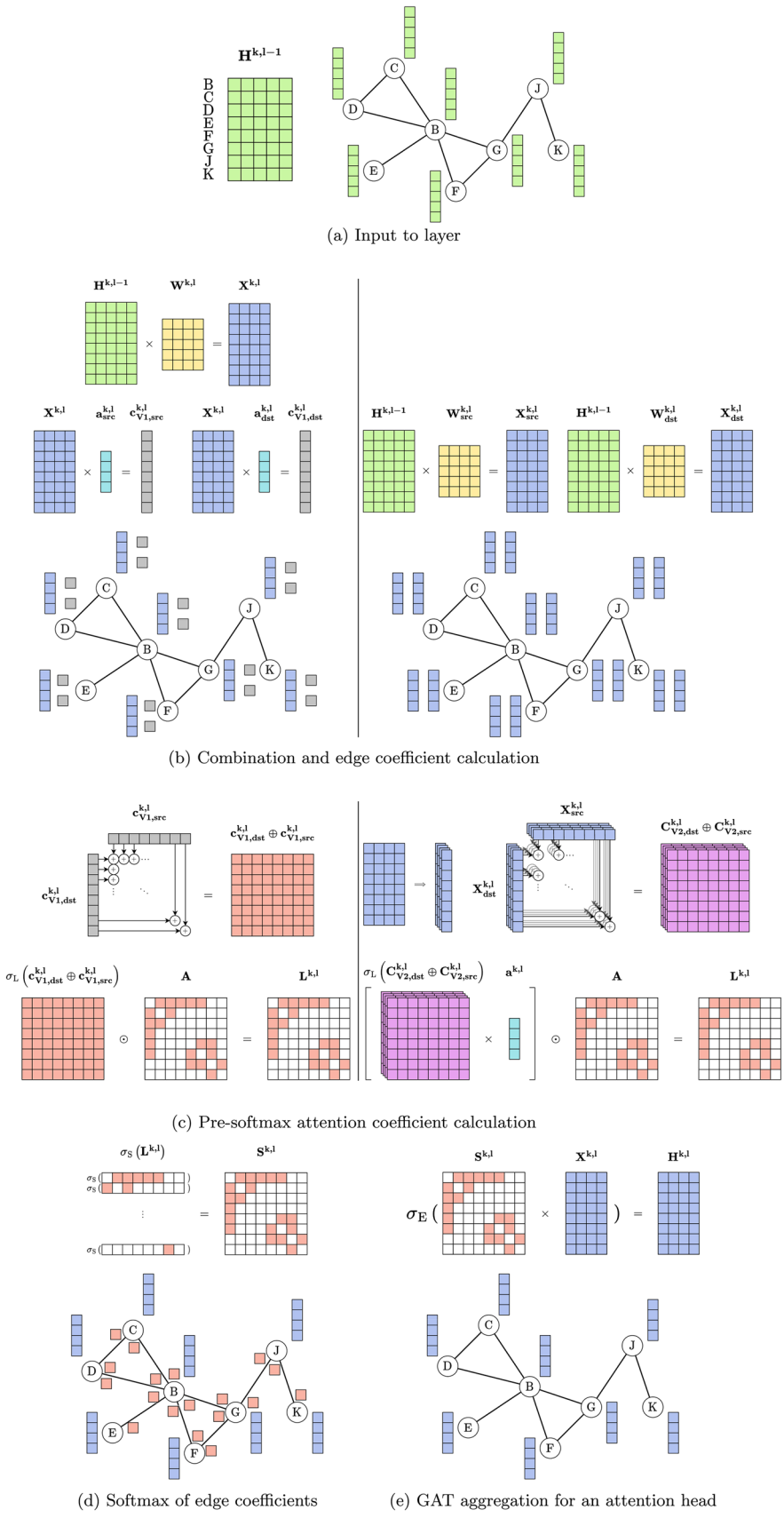
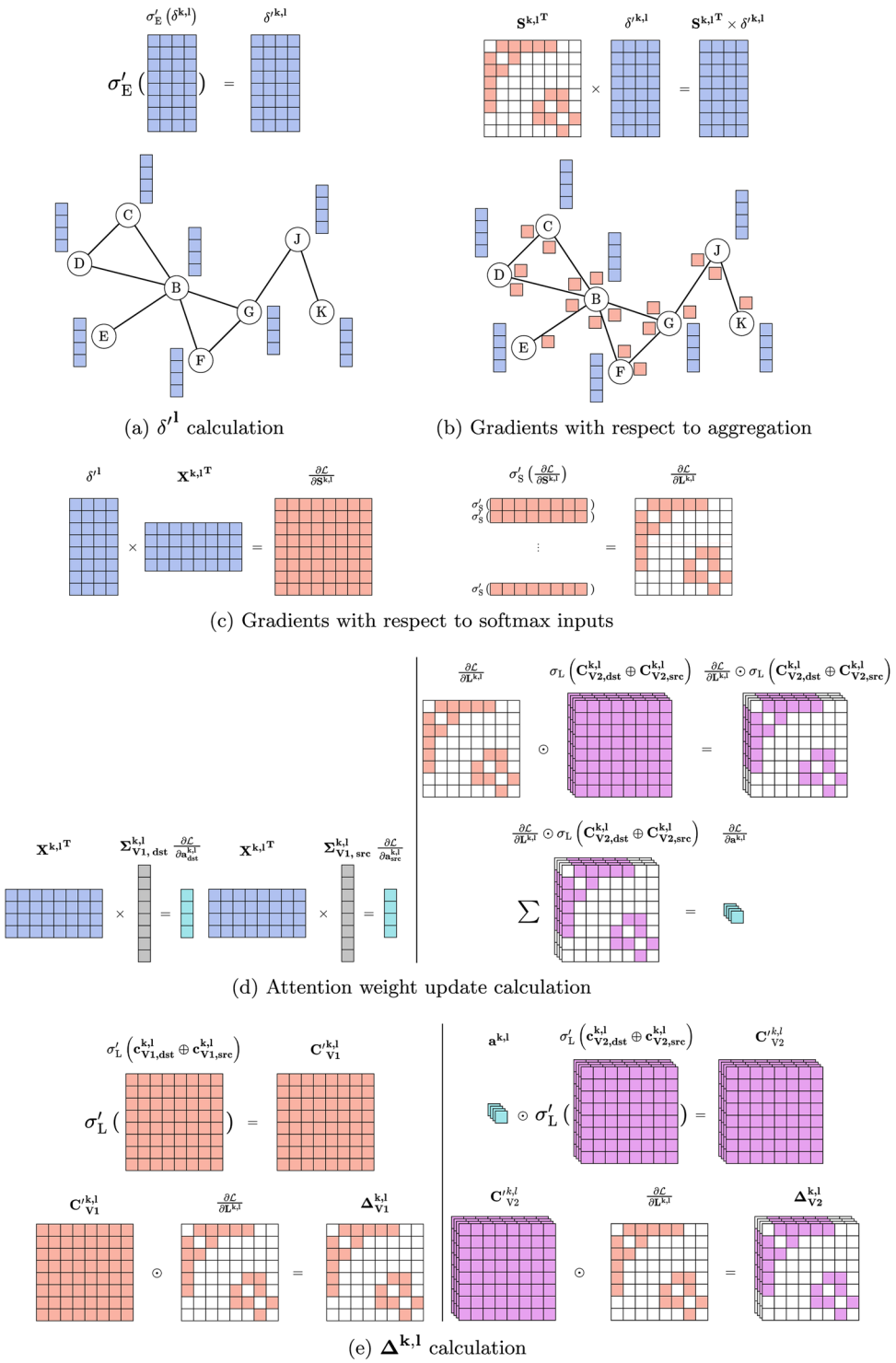


Figure 4 Forward pass of GAT Layer on an 8-Node graph, with 5-feature width input and 4-feature width attention head output. Differences between GATv1 and GATv2 are separated with a vertical line, such that GATv1's operations appear on the left and GATv2's on the right. The matrices are colored by shape according to the legends in Fig. 2. Subfigure **a** shows the input to the layer, with each node having a feature vector associated with it. Subfigure **b** shows the combination and edge coefficient calculation steps, corresponding to Eqs. (2)–(5), with each node having a source and destination coefficient associated with it. Subfigure **c** shows the pre-softmax attention coefficient calculation from Eqs. (8) and (9). Subfigure **d** shows the row-wise softmax operation on the matrix of these coefficients, from Eq. (10). Finally, subfigure **e** shows the aggregation step, weighted with the attention coefficients from Eqs. (11) and (12) (for brevity, we omit the near identical GATv2 equation using $\mathbf{X}^{k,l}_{src}$ instead of $\mathbf{X}^{k,l}$. The figure only shows a single attention head, and the output from subfigure **e** would be concatenated with the output of the other attention heads.



(a) $\delta'^{\mathbf{l}}$ calculation

(b) Gradients with respect to aggregation

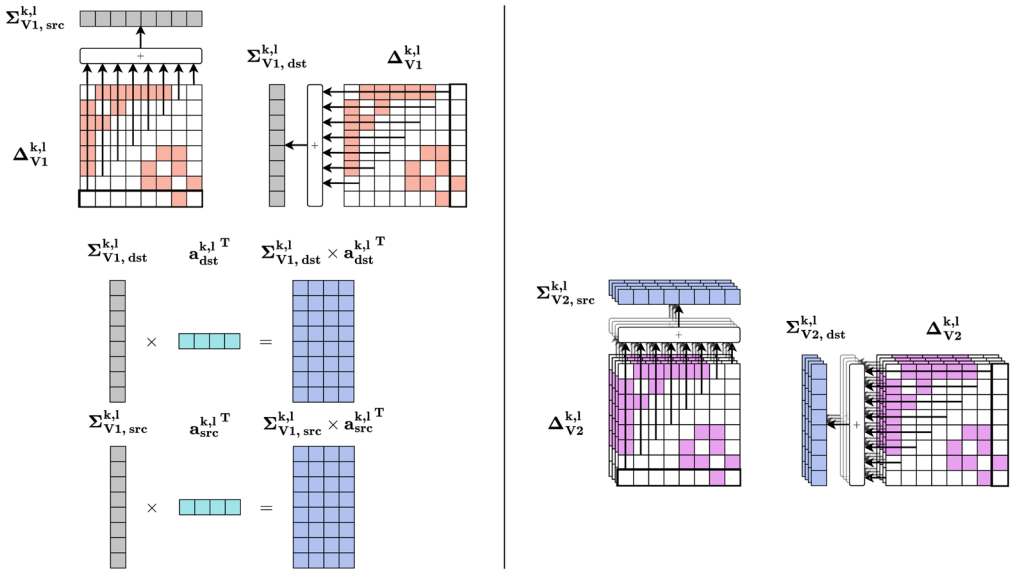
(c) Gradients with respect to softmax inputs

(d) Attention weight update calculation

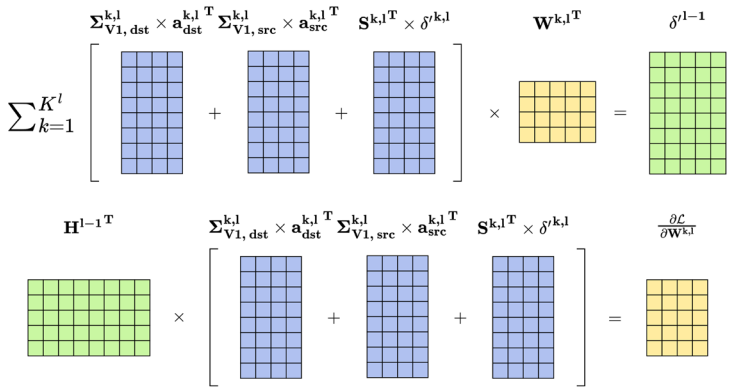
(e) $\mathbf{\Delta}^{\mathbf{k,l}}$ calculation

Figure 5 Backward pass for the forward pass example in Fig. 4. Differences between GATv1 and GATv2 are separated with a vertical line, such that GATv1's operations appear on the left and GATv2's on the right, except for the final step in weight and feature gradient calculation. The matrices are colored by shape according to the legends in Fig. 3. Subfigure **a** shows the calculation of the gradients with respect to the input of the layer activation function from Eq. (15). Subfigure **b** shows the gradient with respect to the aggregation step due to the attention coefficients, corresponding to Eq. (19). Subfigure **c** follows the gradient to the input of the softmax, covering Eq. (21). Subfigure **d** shows the computation of the gradient with respect to attention weights from Eqs. (26) and (30). In subfigure **e**, the $\mathbf{C}'^{\mathbf{k,l}}$ matrix is shown following Eqs. (23) and (27) and used to find $\mathbf{\Delta}^{\mathbf{k,l}}$ as in Eqs. (24) and (28). Subfigure **f** shows the calculation of the $\mathbf{\Sigma}^{\mathbf{k,l}}$ matrices following Eqs. (25) and (29), and the multiplication with the attention weights in GATv1's case. Finally, the gradient with respect to the weights and input features is shown for GATv1 in subfigure **g** from Eqs. (32) and (35) and GATv2 in subfigure **h** from Eqs. (32) and (39).

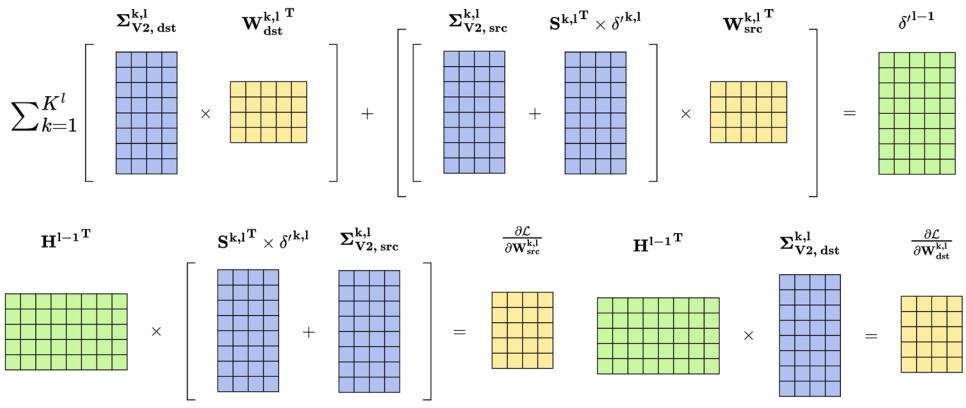
Figure 5 (continued)



(f) $\mathbf{\Sigma^{k,l}_{dst}}$ and $\mathbf{\Sigma^{k,l}_{src}}$ calculation



(g) V1 weight and feature gradient calculation



(h) V2 weight and feature gradient calculation

Acknowledgements The authors thank Nanda Unnikrishnan for numerous useful discussions.

Funding This paper was supported in part by the National Science Foundation under grant number CCF-1954749.

Data Availability Data sharing is not applicable to this article, as no datasets were generated or analyzed during the current study.

References

1. Cheng, Z., Yan, C., Wu, F. X., & Wang, J. (2022). Drug-target interaction prediction using multi-head self-attention and graph attention network. *IEEE/ACM Transactions on Computational Biology and Bioinformatics, 19*(4), 2208–2218. https://doi.org/10.1109/TCBB.2021.3077905. Conference Name: IEEE/ACM Transactions on Computational Biology and Bioinformatics.
2. Yang, Z., Liu, J., Wang, Z., Wang, Y., & Feng, J. (2020). Multi-class metabolic pathway prediction by graph attention-based deep learning method. In: *2020 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pp. 126–131. https://doi.org/10.1109/BIBM49941.2020.9313298
3. Zhao, H., Wang, Y., Duan, J., Huang, C., Cao, D., Tong, Y., Xu, B., Bai, J., Tong, J., & Zhang, Q. (2020). Multivariate time-series anomaly detection via graph attention network. In: *2020 IEEE International Conference on Data Mining (ICDM)*, pp. 841–850. https://doi.org/10.1109/ICDM50108.2020.00093. ISSN: 2374-8486.
4. Zhang, C., James, J. Q., & Liu, Y. (2019). Spatial-temporal graph attention networks: A deep learning approach for traffic forecasting. *IEEE Access, 7*, 166246–166256. https://doi.org/10.1109/ACCESS.2019.2953888. Conference Name: IEEE Access.
5. Balaji, S. S., & Parhi, K. K. (2023). Classifying Subjects with PFC Lesions from Healthy Controls during Working Memory Encoding via Graph Convolutional Networks. In: *2023 11th International IEEE/EMBS Conference on Neural Engineering (NER)*, pp. 1–4. https://doi.org/10.1109/NER52421.2023.10123793. ISSN: 1948-3554.
6. Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., & Bengio, Y. (2018). Graph Attention Networks. arXiv. arXiv:1710.10903 [cs, stat]. http://arxiv.org/abs/1710.10903. Accessed 24 Feb 2023.
7. Brody, S., Alon, U., & Yahav, E. (January 2022). How attentive are graph attention networks? Technical Report arXiv:2105.14491, arXiv. arXiv:2105.14491 [cs] type: article. http://arxiv.org/abs/2105.14491. Accessed 2024 Feb 2023.
8. Unnikrishnan, N. K., & Parhi, K. K. (2023). InterGrad: Energy-efficient training of convolutional neural networks via interleaved gradient scheduling. *IEEE Transactions on Circuits and Systems I: Regular Papers, 70*(5), 1949–1962. https://doi.org/10.1109/TCSI.2023.3246468. Conference Name: IEEE Transactions on Circuits and Systems I: Regular Papers
9. Gori, M., Monfardini, G., & Scarselli, F. (2005). A new model for learning in graph domains. In: *Proceedings. 2005 IEEE International Joint Conference on Neural Networks*, vol. 2, pp. 729–7342. https://doi.org/10.1109/IJCNN.2005.1555942. ISSN: 2161-4407
10. Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., & Monfardini, G. (2009). The graph neural network model. *IEEE Transactions on Neural Networks, 20*(1), 61–80. https://doi.org/10.1109/TNN.2008.2005605. Conference Name: IEEE Transactions on Neural Networks
11. Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., & Yu, P. S. (2021). A comprehensive survey on graph neural networks. *IEEE Transactions Neural Networks Learning System, 32*(1), 4–24. https://doi.org/10.1109/TNNLS.2020.2978386. arXiv:1901.00596 [cs, stat]. Accessed 24 Feb 2023.
12. Parhi, K. K., & Unnikrishnan, N. K. (2020). Brain-inspired computing: models and architectures. *IEEE Open Journal of Circuits and Systems, 1*, 185–204. https://doi.org/10.1109/OJCAS.2020.3032092. Conference Name: IEEE Open Journal of Circuits and Systems
13. Kipf, T. N., & Welling, M. (February 2017). Semi-supervised classification with graph convolutional networks. *Technical Report*. arXiv:1609.02907, arXiv. arXiv:1609.02907 [cs, stat] type: article. http://arxiv.org/abs/1609.02907. Accessed 24 Feb 2023.
14. Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., & Dahl, G. E. (June 2017). Neural message passing for quantum chemistry. *Technical Report*. arXiv:1704.01212, arXiv. arXiv:1704.01212 [cs] type: article. http://arxiv.org/abs/1704.01212. Accessed 24 Feb 2023.
15. Zhang, B., & Prasanna, V. (2023). Dynasparse: Accelerating GNN inference through dynamic sparsity exploitation. https://arxiv.org/abs/2303.12901v1. Accessed 3 Jun 2023.
16. Mondal, S., Manasi, S. D., Kunal, K., Ramprasath, S., & Sapatnekar, S. S. (2021). GNNIE: GNN inference engine with load-balancing and graph-specific caching. https://arxiv.org/abs/2105.10554v2. Accessed 3 Jun 2023.
17. He, Z., Tian, T., Wu, Q., & Jin, X. (2023). FTW-GAT: An FPGA-based accelerator for graph attention networks with ternary weights. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 1–1. https://doi.org/10.1109/TCSII.2023.3280180. Conference Name: IEEE Transactions on Circuits and Systems II: Express Briefs.
18. Geng, T., Wu, C., Zhang, Y., Tan, C., Xie, C., You, H., Herbordt, M., Lin, Y., & Li, A. (2021). I-GCN: A graph convolutional network accelerator with runtime locality enhancement through islandization. In: *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture. MICRO '21*, pp. 1051–1063. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3466752.3480113
19. Zeng, H., & Prasanna, V. (2019). GraphACT: Accelerating GCN training on CPU-FPGA heterogeneous platforms. https://doi.org/10.1145/3373087.3375312. https://arxiv.org/abs/2001.02498v1. Accessed 3 Jun 2023.
20. Chen, X., Wang, Y., Xie, X., Hu, X., Basak, A., Liang, L., Yan, M., Deng, L., Ding, Y., Du, Z., & Xie, Y. (2022). Rubik: A hierarchical architecture for efficient graph neural network training. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 41*(4), 936–949. https://doi.org/10.1109/TCAD.2021.3079142. Conference Name: IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems.
21. Zheng, D., Ma, C., Wang, M., Zhou, J., Su, Q., Song, X., Gan, Q., Zhang, Z., & Karypis, G. (2020). DistDGL: Distributed graph neural network training for billion-scale graphs. https://arxiv.org/abs/2010.05337v3. Accessed 3 Jun 2023.
22. Lin, Z., Li, C., Miao, Y., Liu, Y., & Xu, Y. (2020). Pagraph: Scaling GNN training on large graphs via computation-aware caching. In: *Proceedings of the 11th ACM Symposium on Cloud Computing. SoCC '20*, pp. 401–415. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3419111.3421281
23. Lin, Y.-C., Zhang, B., & Prasanna, V. (2023). HitGNN: High-throughput GNN training framework on CPU+Multi-FPGA heterogeneous platform. https://arxiv.org/abs/2303.01568v1. Accessed 3 Jun 2023.
24. Luong, M.-T., Pham, H., Manning, C.D. (2015). Effective approaches to attention-based neural machine translation. arXiv. arXiv:1508.04025 [cs]. http://arxiv.org/abs/1508.04025. Accessed 27 Jan 2023.
25. Gehring, J., Auli, M., Grangier, D., & Dauphin, Y. N. (July 2017). A convolutional encoder model for neural machine translation. *Technical Report*. arXiv:1611.02344, arXiv. arXiv:1611.02344

[cs] type: article. http://arxiv.org/abs/1611.02344. Accessed 27 Jan 2023.
26. Mnih, V., Heess, N., Graves, A., & Kavukcuoglu, K. (June 2014). Recurrent Models of Visual Attention. *Technical Report*. arXiv:1406.6247, arXiv. arXiv:1406.6247 [cs, stat] type: article. http://arxiv.org/abs/1406.6247. Accessed 27 Jan 2023.
27. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (December 2017). Attention is all you need. *Technical Report*. arXiv:1706.03762, arXiv. arXiv:1706.03762 [cs] type: article. http://arxiv.org/abs/1706.03762. Accessed 27 Jan 2023.
28. Unnikrishnan, N. K., & Parhi, K. K. (2021). LayerPipe: Accelerating deep neural network training by intra-layer and inter-layer gradient pipelining and multiprocessor scheduling. In: *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pp. 1–8. https://doi.org/10.1109/ICCAD51958.2021.9643567. ISSN: 1558-2434.

Joe Gould (Student member, IEEE) is currently pursuing an M.S. degree in electrical engineering at the University of Minnesota, Minneapolis, USA. He worked at Commscope, USA, from 2020 to 2021 under a co-op internship program as a design verification engineer for FPGA products. His interests are in the design of efficient architectures for accelerator and signal processing systems.

Keshab K. Parhi (Fellow, IEEE) received the B.Tech. degree from the Indian Institute of Technology (IIT), Kharagpur, in 1982, the M.S.E.E. degree from the University of Pennsylvania, Philadelphia, in 1984, and the Ph.D. degree from the University of California, Berkeley, in 1988. He has been with the University of Minnesota, Minneapolis, since 1988, where he is currently Erwin A. Kelen Chair and Distinguished McKnight University Professor in the Department of Electrical and Computer Engineering. He has published over 700 papers, is the inventor of 34 patents, and has authored the textbook VLSI Digital Signal Processing Systems (Wiley, 1999). His current research addresses VLSI architecture design of machine learning and signal processing systems, hardware security, and data-driven neuroengineering and neuroscience. Dr. Parhi is the recipient of numerous awards including the 2017 Mac Van Valkenburg award and the 2012 Charles A. Desoer Technical Achievement award from the IEEE Circuits and Systems Society, the 2003 IEEE Kiyo Tomiyasu Technical Field Award, and a Golden Jubilee medal from the IEEE Circuits and Systems Society in 2000. He served as the Editor-in-Chief of the IEEE Trans. Circuits and Systems, Part-I during 2004 and 2005. He is a Fellow of the American Association for the Advancement of Science (AAAS), the Association for Computing Machinery (ACM), the American Institute of Medical and Biological Engineering (AIMBE), and the National Academy of Inventors (NAI).