# Graph Attention Networks

Juntao Wang

The Chinese University of Hong Kong, Shenzhen
School of Data Science

November 2, 2021

## Contents

## Table of Contents

## Graph Convolutional Networks[1]

### Multi-layer Graph Convolutional Network

We consider a **multi-layer Graph Convolutional Network** (GCN) with the following layer-wise propagation rule:

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right). \tag{1}$$

- $\tilde{A} = A + I_N$ is the adjacency matrix of the undirected graph $\mathcal{G}$ with added self-connections.
- $\tilde{D}$ is the degree matrix, $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$.
- $W^{(l)}$ is a layer-specific trainable weight matrix.
- $\sigma(\cdot)$ denotes an activation function
- $H^{(l)}$ is the matrix of activations in the $l^{\text{th}}$ layer; $H^{(0)} = X$.

[1] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

## Graph Convolutional Networks

Or we can rewrite (1) as:

$$H^{(l+1)} = \sigma\left(\hat{A} H^{(l)} W^{(l)}\right) . \qquad (2)$$

where $\hat{A} = \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}}$.

**Example (Two-layer GCN)**

Considering a two-layer GCN, we first calculate $\hat{A} = \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}}$ in a pre-processing step. The forward model then takes the simple form:

$$Z = f(X, A) = \mathrm{softmax}\left(\hat{A}\ \mathrm{ReLU}\left(\hat{A} X W^{(0)}\right) W^{(1)}\right) . \qquad (3)$$

Here, $W^{(0)} \in \mathbb{R}^{C \times H}$ is an input-to-hidden weight matrix for a hidden layer with $H$ feature maps. $W^{(1)} \in \mathbb{R}^{H \times F}$ is a hidden-to-output weight matrix.

## Graph Convolutional Networks

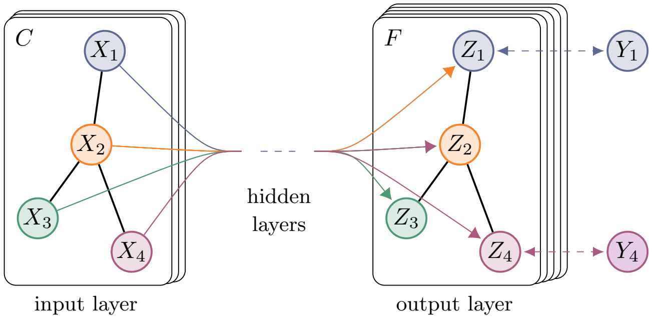


Figure: Schematic depiction of multi-layer Graph Convolutional Network (GCN) for semi-supervised learning with $C$ input channels and $F$ feature maps in the output layer. The graph structure (edges shown as black lines) is shared over layers, labels are denoted by $Y_i$.

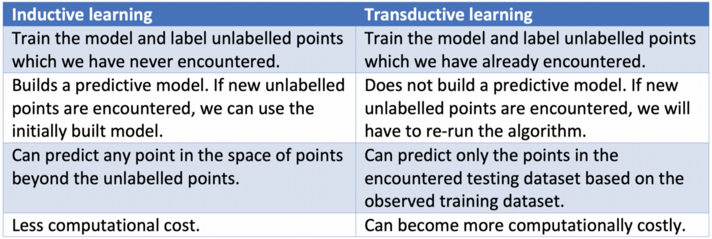
## Table of Contents

## Motivation

Most embedding frameworks are inherently **transductive**

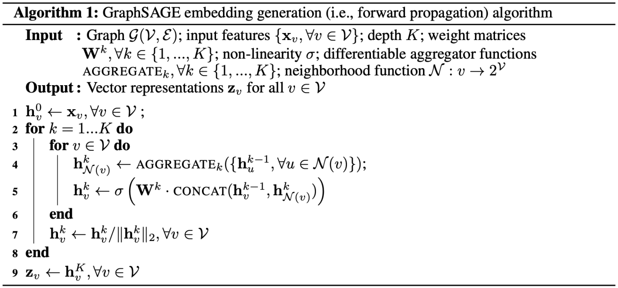
- They can only generate embeddings for a single fixed graph.
- They do not efficiently generalize to unseen nodes (e.g., in evolving graphs), and these approaches cannot learn to generalize across different graphs.

In contrast, GraphSAGE is an **inductive** framework that leverages node attribute information to efficiently generate representations on previously unseen data.



| Inductive learning | Transductive learning |
|---|---|
| Train the model and label unlabelled points which we have never encountered. | Train the model and label unlabelled points which we have already encountered. |
| Builds a predictive model. If new unlabelled points are encountered, we can use the initially built model. | Does not build a predictive model. If new unlabelled points are encountered, we will have to re-run the algorithm. |
| Can predict any point in the space of points beyond the unlabelled points. | Can predict only the points in the encountered testing dataset based on the observed training dataset. |
| Less computational cost. | Can become more computationally costly. |

# GraphSAGE[2]

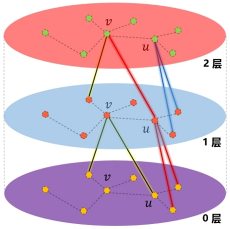


**Algorithm 1:** GraphSAGE embedding generation (i.e., forward propagation) algorithm

**Input :** Graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$; input features $\{\mathbf{x}_v, \forall v \in \mathcal{V}\}$; depth $K$; weight matrices $\mathbf{W}^k, \forall k \in \{1, ..., K\}$; non-linearity $\sigma$; differentiable aggregator functions $\text{AGGREGATE}_k, \forall k \in \{1, ..., K\}$; neighborhood function $\mathcal{N} : v \rightarrow 2^{\mathcal{V}}$

**Output :** Vector representations $\mathbf{z}_v$ for all $v \in \mathcal{V}$

```
$\mathbf{h}_v^0 \leftarrow \mathbf{x}_v, \forall v \in \mathcal{V}$ ;
for $k = 1...K$ do
    for $v \in \mathcal{V}$ do
        $\mathbf{h}_{\mathcal{N}(v)}^k \leftarrow \text{AGGREGATE}_k(\{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\})$;
        $\mathbf{h}_v^k \leftarrow \sigma\left(\mathbf{W}^k \cdot \text{CONCAT}(\mathbf{h}_v^{k-1}, \mathbf{h}_{\mathcal{N}(v)}^k)\right)$
    end
    $\mathbf{h}_v^k \leftarrow \mathbf{h}_v^k / \|\mathbf{h}_v^k\|_2, \forall v \in \mathcal{V}$
end
$\mathbf{z}_v \leftarrow \mathbf{h}_v^K, \forall v \in \mathcal{V}$
```



[2] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 2017, pp. 1025–1035.

## GraphSAGE

### Graph SAmple and aggreGatE

GraphSAGE adopts sampling to obtain a fixed number of neighbors for each node. It performs graph convolutions by

$$\mathbf{h}_v^{(k)} = \sigma(\mathbf{W}^{(k)} \cdot f_k(\mathbf{h}_v^{(k-1)}, \{\mathbf{h}_u^{(k-1)}, \forall u \in \mathcal{N}(v)\})), \tag{4}$$

where $\mathbf{h}_v^{(0)} = \mathbf{x}_v$, $f_k(\cdot)$ is an aggregation function, $\mathcal{N}(v)$ is a random sample of the node $v$'s neighbors. The aggregation function should be **invariant to the permutations of node orderings** such as a mean, sum or max function.

## Table of Contents

## Motivation and Properties

The idea[3] is to compute the hidden representations of each node in the graph, by attending over its neighbors, following a self-attention strategy. The attention architecture has several interesting properties:

- The operation is **efficient**, since it is **parallelizable** across node-neighbor pairs;
- It can be applied to graph nodes having **different degrees** by specifying **arbitrary weights** to the neighbors;
- The model is directly applicable to **inductive** learning problems, including tasks where the model has to generalize to completely unseen graphs.

[3] P. Veličković, G. Cucurull, A. Casanova, *et al.*, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2017.

# Table of Contents

## Graph Attention Layer

### Attention Coefficient

A shared linear transformation, parametrized by a *learnable weight matrix*, $\mathbf{W} \in \mathbb{R}^{F' \times F}$, is applied to every node. We then perform *self-attention* on the nodes—a *shared attentional mechanism* $a : \mathbb{R}^{F'} \times \mathbb{R}^{F'} \to \mathbb{R}$ computes **attention coefficients** which indicate the *importance* of node $j$'s features to node $i$.

$$e_{ij} = a(\mathbf{W}\vec{h}_i, \mathbf{W}\vec{h}_j) \tag{5}$$

- In its most general formulation, the model allows every node to attend on every other node, *dropping all structural information*.
- We inject the graph structure into the mechanism by performing **masked attention**—we only compute $e_{ij}$ for nodes $j \in \mathcal{N}_i$, where $\mathcal{N}_i$ is *some neighborhood* of node $i$ in the graph.

## Graph Attention Layer

In this paper, the attention mechanism $a$ is a single-layer feedforward neural network, parametrized by a learnable weight vector $\vec{\mathbf{a}} \in \mathbb{R}^{2F'}$, and applying the LeakyReLU nonlinearity (with negative input slope $\alpha = 0.2$).

$$e_{ij} = \text{LeakyReLU}\left(\vec{\mathbf{a}}^T [\mathbf{W}\vec{h}_i \| \mathbf{W}\vec{h}_j]\right) \tag{6}$$

where $\|$ is the concatenation operation.

To make coefficients comparable across different nodes, we normalize them across all choices of $j$ using the *softmax* function:

$$\alpha_{ij} = \mathrm{softmax}_j(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{k\in\mathcal{N}_i} \exp(e_{ik})}. \tag{7}$$

## Graph Attention Layer

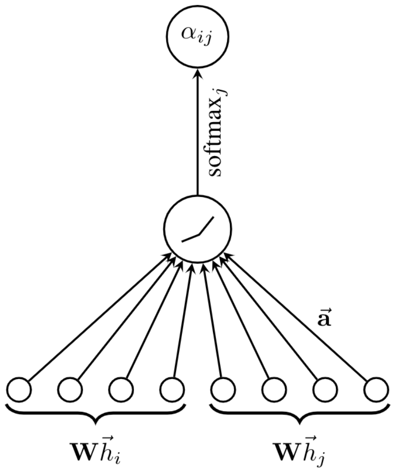
Fully expanded out, the **normalized attention coefficients** can be expressed as:

$$\alpha_{ij} = \frac{\exp\left(\text{LeakyReLU}\left(\vec{\mathbf{a}}^T[\mathbf{W}\vec{h}_i \| \mathbf{W}\vec{h}_j]\right)\right)}{\sum_{k\in\mathcal{N}_i}\exp\left(\text{LeakyReLU}\left(\vec{\mathbf{a}}^T[\mathbf{W}\vec{h}_i \| \mathbf{W}\vec{h}_k]\right)\right)} \tag{8}$$

Once obtained, the normalized attention coefficients are used to compute a linear combination of the features corresponding to them, to serve as the final output features for every node (after potentially applying a nonlinearity, $\sigma$):

$$\vec{h}'_i = \sigma\left(\sum_{j\in\mathcal{N}_i} \alpha_{ij}\mathbf{W}\vec{h}_j\right). \tag{9}$$

## Graph Attention Layer



## Table of Contents

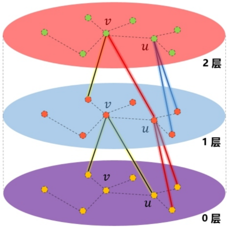
# Graph Attention Networks

## Multi-layer GAT

For multi-layer version, GAT can be expressed as

$$\mathbf{h}_v^{(k)} = \sigma\Big( \sum_{u \in \mathcal{N}(v) \cup v} \alpha_{vu}^{(k)} \mathbf{W}^{(k)} \mathbf{h}_u^{(k-1)} \Big) \tag{10}$$

where $\mathbf{h}_v^{(0)} = \mathbf{x}_v$. The attention weight $\alpha_{vu}^{(k)}$ measures the connective strength between the node $v$ and its neighbor $u$



## Table of Contents

## Multi-head Attention

To stabilize the learning process of self-attention, this paper found extending the mechanism to employ **multi-head attention** to be beneficial, similarly to Vaswani et al.[4].

### Multi-head Attention

Multi-head attention allows the model to jointly attend to information from different representation subspaces at different positions. With a single attention head, averaging inhibits this.

$$\mathrm{MultiHead}(Q, K, V) = \mathrm{Concat}(\mathrm{head_1}, ..., \mathrm{head_h})W^O$$
$$\text{where } \mathrm{head_i} = \mathrm{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

[4] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

## Multi-head Attention

Specifically, $K$ independent attention mechanisms execute the transformation of Equation (9), and then their features are concatenated, resulting in the following output feature representation:

$$\vec{h}'_i = \|_{k=1}^{K} \sigma\left(\sum_{j\in\mathcal{N}_i} \alpha_{ij}^{k}\mathbf{W}^{k}\vec{h}_j\right) \tag{11}$$

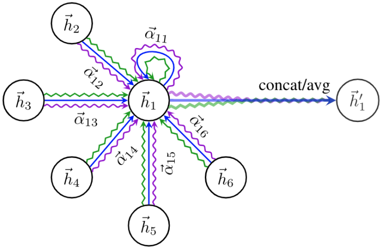
where $\|$ represents concatenation, $\alpha_{ij}^{k}$ are normalized attention coefficients computed by the $k$-th attention mechanism ($a^k$), and $\mathbf{W}^k$ is the corresponding input linear transformation's weight matrix.

**Note:** In this setting, the final returned output, $\mathbf{h}'$, will consist of $KF'$ features (rather than $F'$) for each node.

## Multi-head Attention

If we perform multi-head attention on the final (prediction) layer of the network, concatenation is no longer sensible—instead, we employ *averaging*, and delay applying the final nonlinearity (usually a softmax or logistic sigmoid for classification problems) until then:

$$\vec{h}'_i = \sigma\left(\frac{1}{K}\sum_{k=1}^{K}\sum_{j\in\mathcal{N}_i}\alpha_{ij}^k\mathbf{W}^k\vec{h}_j\right) \tag{12}$$



# Table of Contents

## Comparison with Related Work

**Computationally, it is highly efficient:**

- *No eigendecompositions or similar costly matrix operations*.
- The operation of the self-attentional layer can be *parallelized* across all edges, and the computation of output features can be *parallelized* across all nodes.
- The time complexity is *on par with the baseline methods (e.g.GCN)*.
  - The time complexity of a single GAT attention head computing $F'$ features may be expressed as $O(|V|FF' + |E|F')$, where $F$ is the number of input features, and $|V|$ and $|E|$ are the numbers of nodes and edges in the graph, respectively.
- Applying multi-head attention multiplies the storage and parameter requirements by a factor of $K$, while the *heads' computations are fully independent and can be parallelized*.

## Comparison with Related Work

**Parametric weights:**

- As opposed to GCNs, GAT allows for (implicitly) assigning *different importances* to nodes of a same neighborhood, enabling a leap in model capacity. Furthermore, analyzing the learned attentional weights may lead to benefits in interpretability.

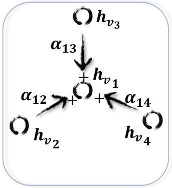


Figure: GCN explicitly assigns a non-parametric weight $a_{ij} = 1/\sqrt{deg(v_i)deg(v_j)}$ to the neighbor $v_j$ of $v_i$.

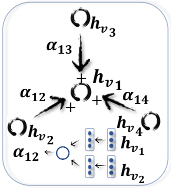


Figure: GAT implicitly captures the weight $a_{ij}$ via an end-to-end neural network architecture, so that more important nodes receive larger weights.

## Comparison with Related Work

**Doesn't dependent on global graph structure:**
The attention mechanism is applied in a shared manner to all edges in the graph, and therefore it does not depend on upfront access to the global graph structure (a limitation of many prior techniques). This has several desirable implications:

- The graph can be directed (we may simply leave out computing $\alpha_{ij}$ if edge $j \to i$ is not present).
- It makes GAT directly applicable to *inductive learning*, including tasks where the model is evaluated on graphs that are *completely unseen* during training.

## Comparison with Related Work

**Entirety of the neighborhood & Doesn't assume any node ordering:**

- GraphSAGE samples a *fixed-size neighborhood* of each node, in order to keep its computational footprint consistent; this does not allow it access to the entirety of the neighborhood while performing inference. Moreover, this technique acquired aggregation mechanism which is *insensible to the node orderings*.
- GAT does not suffer from either of these issues—it works with the entirety of the neighborhood , and does not assume any ordering within it.

## Comparison with Related Work

**Problems:**

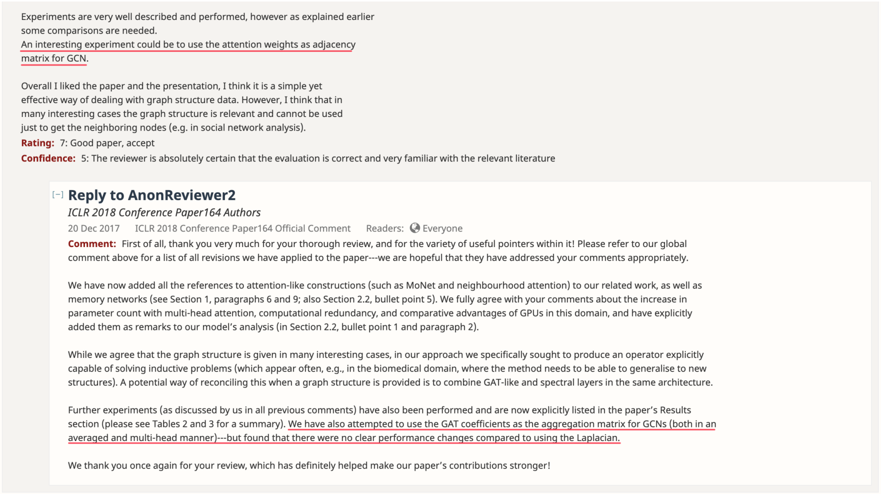
- Depending on the regularity of the graph structure in place, GPUs may not be able to offer major performance benefits compared to CPUs in sparse scenarios.
- It should also be noted that the size of the “receptive field” of GAT is upper-bounded by the depth of the network (similarly as for GCN and similar models). Techniques such as skip connections could be readily applied for appropriately extending the depth, however.
- Lastly, parallelization across all the graph edges, especially in a distributed manner, may involve a lot of redundant computation, as the neighborhoods will often highly overlap in graphs of interest.

## Table of Contents

## Some Ideas

- About learnable weight matrix in attention and propagation.
- About attention coefficients of GAT and adjacent matrix of GCN.

# Some Ideas



Experiments are very well described and performed, however as explained earlier
some comparisons are needed.
An interesting experiment could be to use the attention weights as adjacency
matrix for GCN.

Overall I liked the paper and the presentation, I think it is a simple yet
effective way of dealing with graph structure data. However, I think that in
many interesting cases the graph structure is relevant and cannot be used
just to get the neighboring nodes (e.g. in social network analysis).

**Rating:** 7: Good paper, accept

**Confidence:** 5: The reviewer is absolutely certain that the evaluation is correct and very familiar with the relevant literature

[–] **Reply to AnonReviewer2**

*ICLR 2018 Conference Paper164 Authors*

20 Dec 2017 ICLR 2018 Conference Paper164 Official Comment Readers: Everyone

**Comment:** First of all, thank you very much for your thorough review, and for the variety of useful pointers within it! Please refer to our global comment above for a list of all revisions we have applied to the paper---we are hopeful that they have addressed your comments appropriately.

We have now added all the references to attention-like constructions (such as MoNet and neighbourhood attention) to our related work, as well as memory networks (see Section 1, paragraphs 6 and 9; also Section 2.2, bullet point 5). We fully agree with your comments about the increase in parameter count with multi-head attention, computational redundancy, and comparative advantages of GPUs in this domain, and have explicitly added them as remarks to our model's analysis (in Section 2.2, bullet point 1 and paragraph 2).

While we agree that the graph structure is given in many interesting cases, in our approach we specifically sought to produce an operator explicitly capable of solving inductive problems (which appear often, e.g., in the biomedical domain, where the method needs to be able to generalise to new structures). A potential way of reconciling this when a graph structure is provided is to combine GAT-like and spectral layers in the same architecture.

Further experiments (as discussed by us in all previous comments) have also been performed and are now explicitly listed in the paper's Results section (please see Tables 2 and 3 for a summary). We have also attempted to use the GAT coefficients as the aggregation matrix for GCNs (both in an averaged and multi-head manner)---but found that there were no clear performance changes compared to using the Laplacian.

We thank you once again for your review, which has definitely helped make our paper's contributions stronger!

## Table of Contents

## References

[1] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[2] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 2017, pp. 1025–1035.

[3] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2017.

[4] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

The End