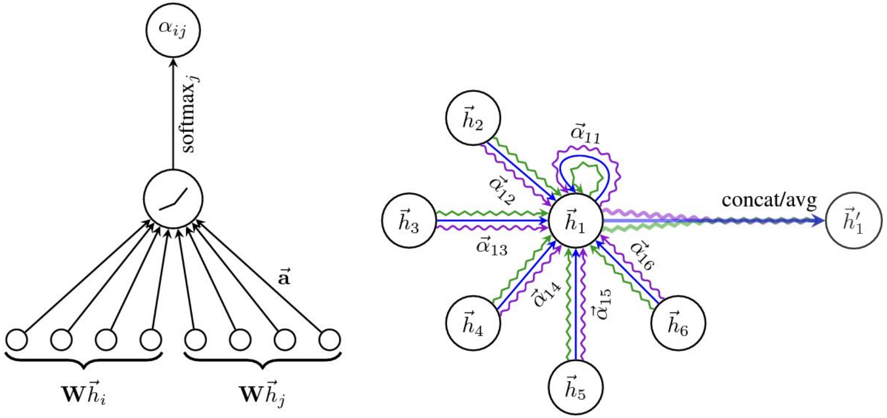


Graph Attention Networks

"*Everything connects to everything else*" - Leonardo DaVinci

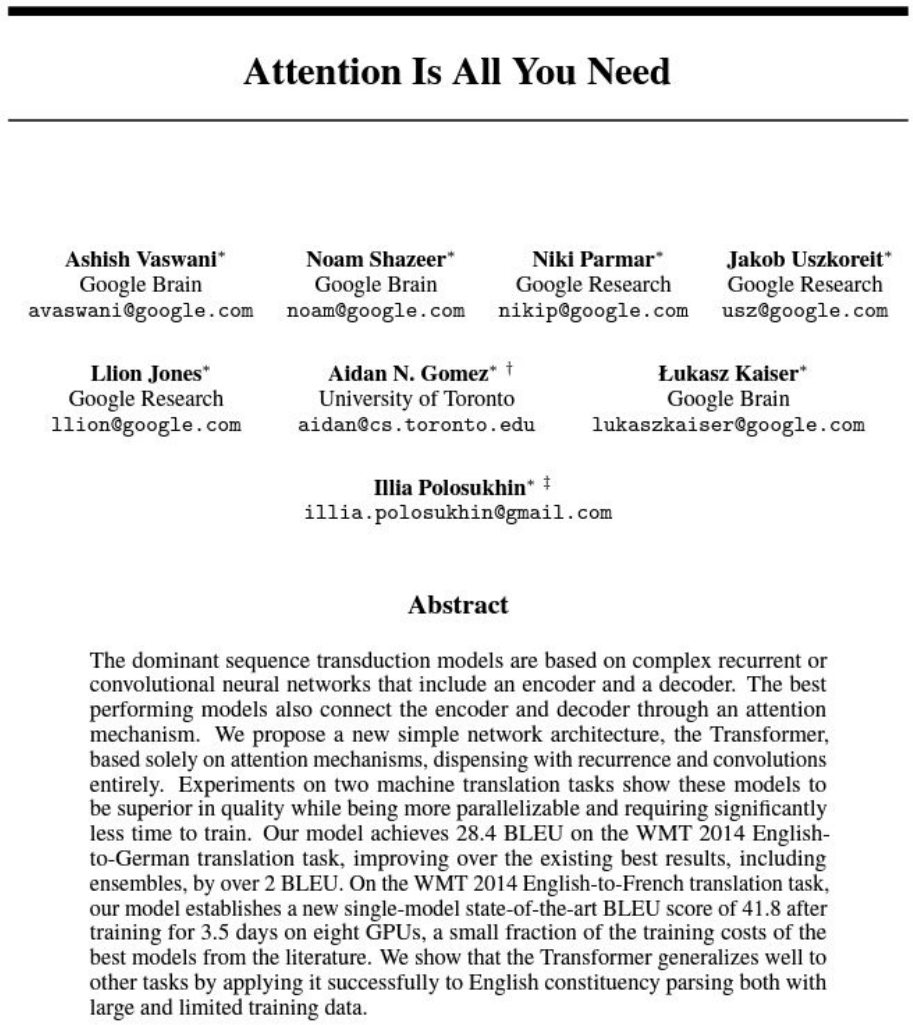
Jay Urbain, PhD - 9/27/2022

References

Attention is All You Need

https://arxiv.org/abs/1706.03762

Attention is All You Need - Yanic Kilcher
https://www.youtube.com/watch?v=iDulhoQ2pro



Attention Is All You Need

Ashish Vaswani*
Google Brain
avaswani@google.com

Noam Shazeer*
Google Brain
noam@google.com

Niki Parmar*
Google Research
nikip@google.com

Jakob Uszkoreit*
Google Research
usz@google.com

Llion Jones*
Google Research
llion@google.com

Aidan N. Gomez* †
University of Toronto
aidan@cs.toronto.edu

Łukasz Kaiser*
Google Brain
lukaszkaiser@google.com

Illia Polosukhin* ‡
illia.polosukhin@gmail.com

Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.8 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature. We show that the Transformer generalizes well to other tasks by applying it successfully to English constituency parsing both with large and limited training data.

References

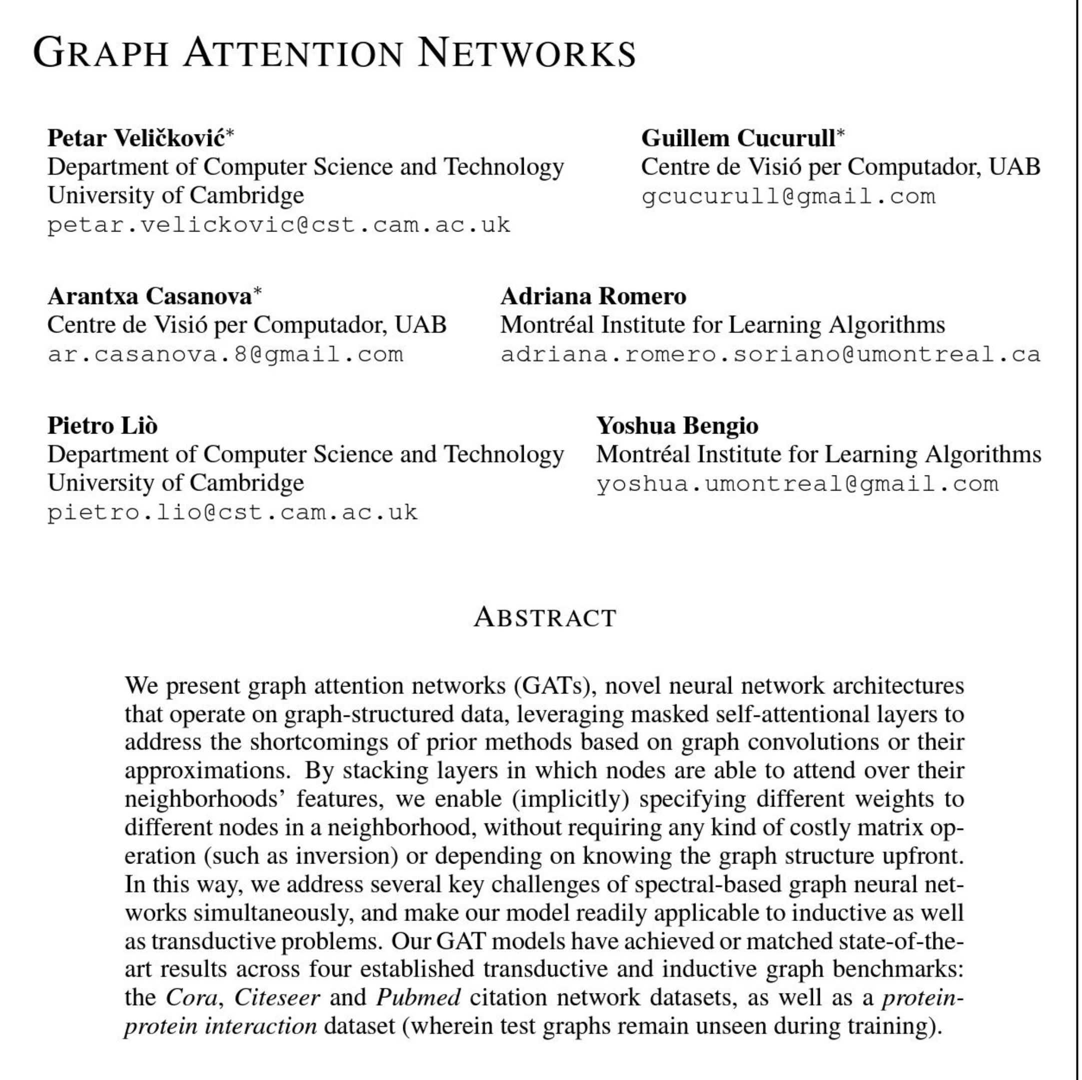
Graph Attention Networks

https://arxiv.org/pdf/1710.10903.pdf

Rasa Algorithm Whiteboard - Transformers & Attention 1: Self Attention
https://www.youtube.com/watch?v=SnRfBfXwLuY

Understanding Graph Attention Networkshttps://www.youtube.com/watch?v=A-yKQamf2Fc

Graph Attention Networks (GATs)
https://www.youtube.com/watch?v=SnRfBfXwLuY



GRAPH ATTENTION NETWORKS

Petar Veličković*
Department of Computer Science and Technology
University of Cambridge
petar.velickovic@cst.cam.ac.uk

Guillem Cucurull*
Centre de Visió per Computador, UAB
gcucurull@gmail.com

Arantxa Casanova*
Centre de Visió per Computador, UAB
ar.casanova.8@gmail.com

Adriana Romero
Montréal Institute for Learning Algorithms
adriana.romero.soriano@umontreal.ca

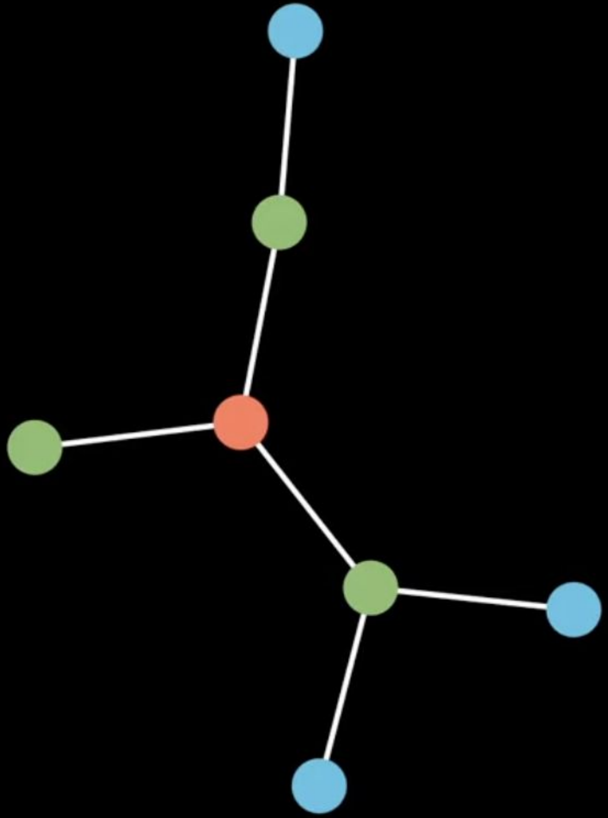
Pietro Liò
Department of Computer Science and Technology
University of Cambridge
pietro.lio@cst.cam.ac.uk

Yoshua Bengio
Montréal Institute for Learning Algorithms
yoshua.umontreal@gmail.com

ABSTRACT

We present graph attention networks (GATs), novel neural network architectures that operate on graph-structured data, leveraging masked self-attentional layers to address the shortcomings of prior methods based on graph convolutions or their approximations. By stacking layers in which nodes are able to attend over their neighborhoods' features, we enable (implicitly) specifying different weights to different nodes in a neighborhood, without requiring any kind of costly matrix operation (such as inversion) or depending on knowing the graph structure upfront. In this way, we address several key challenges of spectral-based graph neural networks simultaneously, and make our model readily applicable to inductive as well as transductive problems. Our GAT models have achieved or matched state-of-the-art results across four established transductive and inductive graph benchmarks: the *Cora*, *Citeseer* and *Pubmed* citation network datasets, as well as a *protein-protein interaction* dataset (wherein test graphs remain unseen during training).

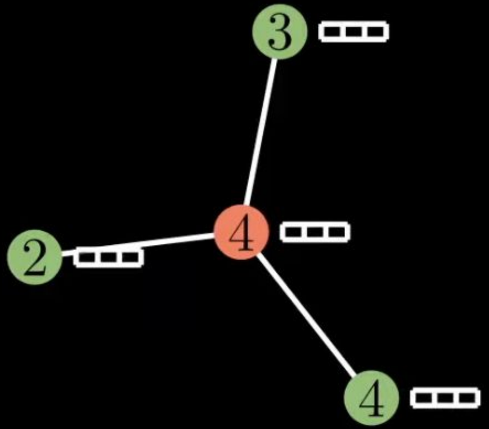
GCN



Review:
Graph Convolution Network

GCN

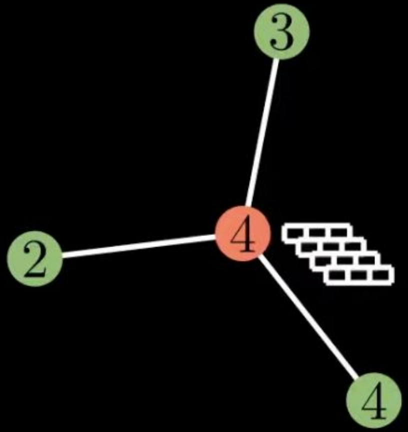
$$\mathbf{h}_{\mathcal{N}(v)} = \frac{1}{|\mathcal{N}(v)|} \sum_{u \in \mathcal{N}(v)} \mathbf{h}_u$$



GCN:
Message passing from neighbor nodes.
Normalize by node degree.

GCN

$$\mathbf{h}_{\mathcal{N}(v)} = \frac{1}{|\mathcal{N}(v)|} \sum_{u \in \mathcal{N}(v)} \mathbf{h}_u$$



Messages are aggregated by each node.

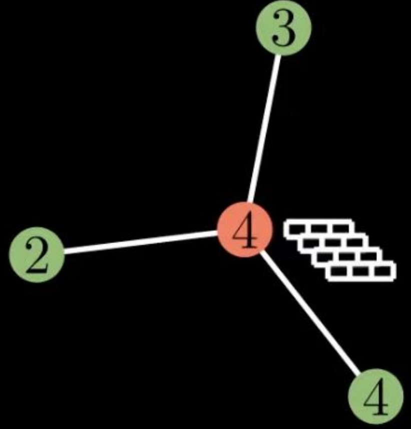
Normalize by incoming degree of target node.

Simple average.

GCN

$$\mathbf{h}_{\mathcal{N}(v)} = \sum_{u \in \mathcal{N}(v)} w_{u,v} \mathbf{h}_u$$

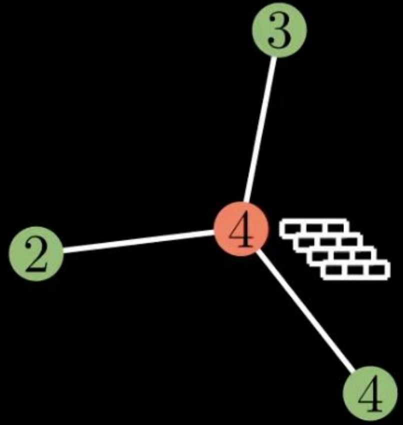
$$= \sum_{u \in \mathcal{N}(v)} \sqrt{\frac{1}{d_v}} \sqrt{\frac{1}{d_u}} \mathbf{h}_u$$



Improve normalization by using a weighted average over outgoing and incoming messages.

Normalize by outgoing and incoming node degree.

GCN



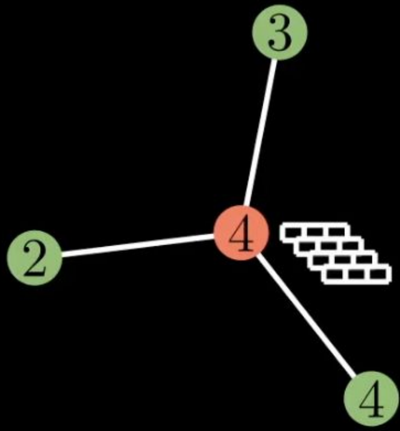
$$\mathbf{h}_{\mathcal{N}(v)} = \sum_{u \in \mathcal{N}(v)} w_{u,v} \mathbf{h}_u$$

$$= \sum_{u \in \mathcal{N}(v)} \sqrt{\frac{1}{d_v}} \sqrt{\frac{1}{d_u}} \mathbf{h}_u$$

$$= \sqrt{\frac{1}{d_v}} \sum_{u \in \mathcal{N}(v)} \sqrt{\frac{1}{d_u}} \mathbf{h}_u$$

Can pull term for target node normalization out of the sum.

GCN



$$\begin{aligned}\mathbf{h}_{\mathcal{N}(v)} &= \sum_{u \in \mathcal{N}(v)} w_{u,v} \mathbf{h}_u \\ &= \sum_{u \in \mathcal{N}(v)} \sqrt{\frac{1}{d_v}} \sqrt{\frac{1}{d_u}} \mathbf{h}_u \\ &= \sqrt{\frac{1}{d_v}} \sum_{u \in \mathcal{N}(v)} \sqrt{\frac{1}{d_u}} \mathbf{h}_u\end{aligned}$$

$$\mathbf{h}_{\mathcal{N}(0)} = \frac{1}{\sqrt{4}} \left(\frac{\mathbf{h}_0}{\sqrt{4}} + \frac{\mathbf{h}_1}{\sqrt{3}} + \frac{\mathbf{h}_2}{\sqrt{2}} + \frac{\mathbf{h}_3}{\sqrt{4}} \right)$$

Graph Attention Networks

GCN - Graph Convolutional Networks use a weighted average where the weight depends on the number of neighbors.

$$\mathbf{h}_{\mathcal{N}(v)} = \sum_{u \in \mathcal{N}(v)} w_{u,v} \mathbf{h}_u$$

How are Graph Attention Networks Different?

Graph Attention Networks

GCN - Graph Convolutional Networks use a weighted average where the weight depends on the number of neighbors.

$$\mathbf{h}_{\mathcal{N}(v)} = \sum_{u \in \mathcal{N}(v)} a(\mathbf{h}_u, \mathbf{h}_v)\mathbf{h}_u$$

GAT - Graph Attention Networks use a (learnable) function that calculates the weights.

Allows function to consider the embeddings of the source and target nodes.

Embeddings capture features and local structure.

Graph Attention Networks

$$\mathbf{h}_{\mathcal{N}(v)} = \sum_{u\in\mathcal{N}(v)} \mathrm{softmax}_u\left(a(\mathbf{h}_u, \mathbf{h}_v)\right)\mathbf{h}_u$$

Apply softmax so the attention across all incoming nodes for each target node sums to 1.0.

Graph Attention Networks

$$\mathbf{h}_{\mathcal{N}(v)} = \sum_{u \in \mathcal{N}(v)} \overbrace{\text{softmax}_u\left(a(\mathbf{h}_u, \mathbf{h}_v)\right)}^{\alpha_{u,v}} \mathbf{h}_u$$

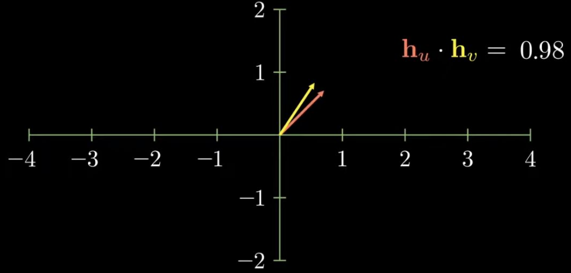
Normalized attention for u, v.

$$\alpha_{u,v} = \frac{\exp(a(\mathbf{h}_u, \mathbf{h}_v))}{\sum_{k \in \mathcal{N}(v)} \exp(a(\mathbf{h}_u, \mathbf{h}_v))}$$

How should we calculate the *attention scores*?

Calculating attention

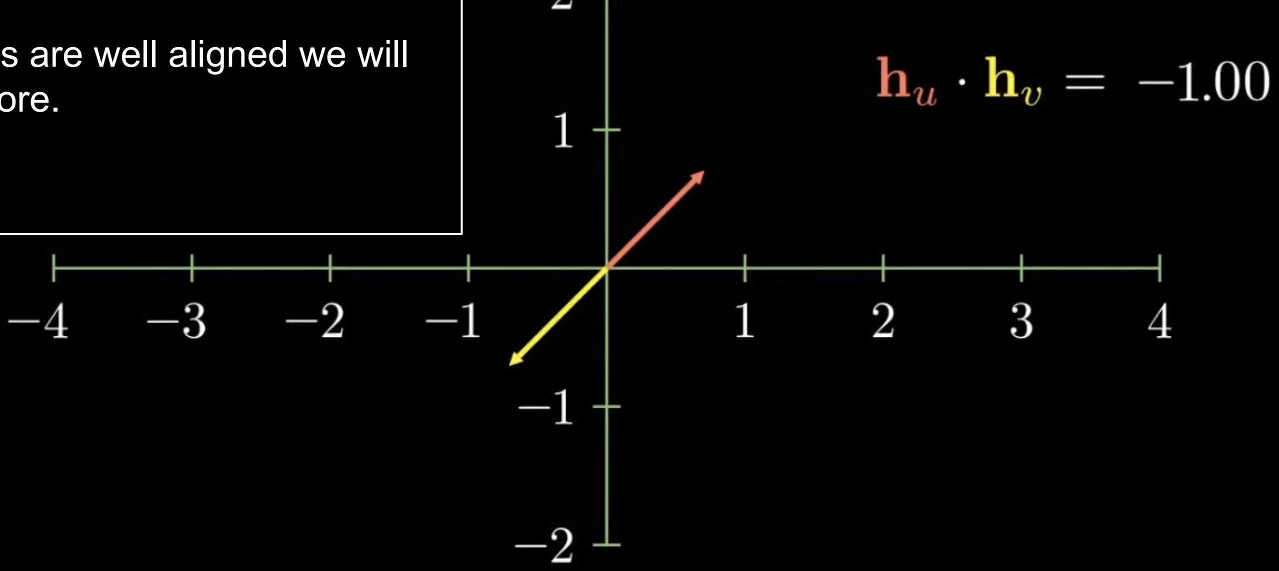
$$a(\mathbf{h}_u, \mathbf{h}_v) = \mathbf{h}_u \cdot \mathbf{h}_v$$



Can use dot product (Cosine) between embeddings to measure attention.

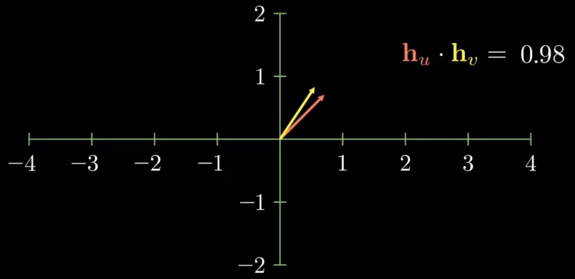
If the embeddings are well aligned we will have a higher score.

Disadvantage?



Calculating attention

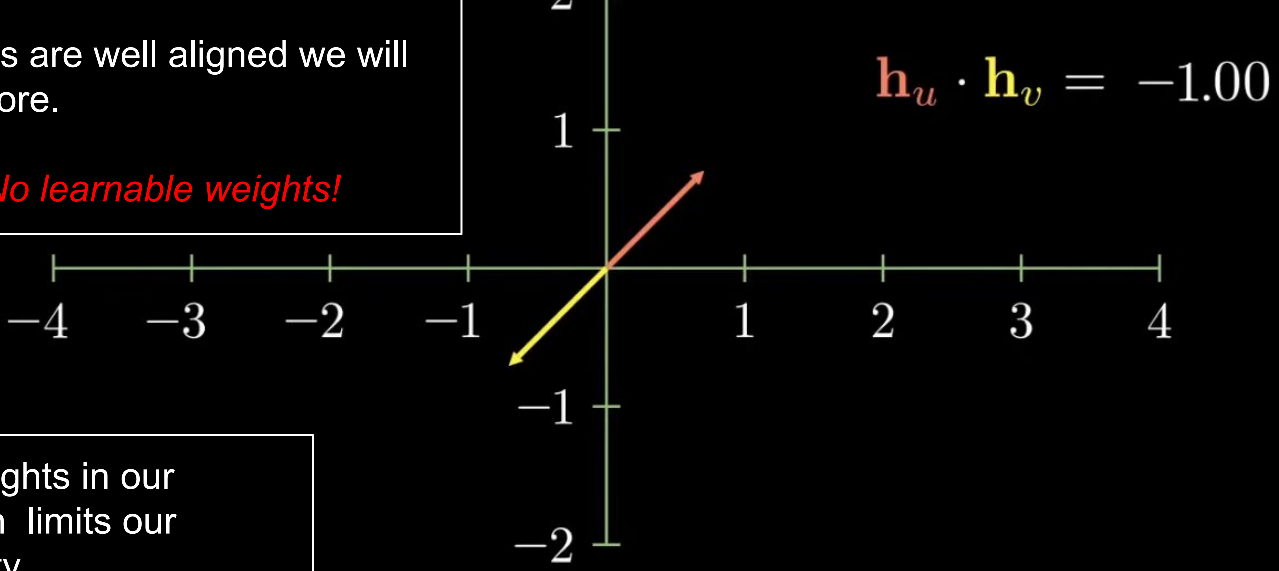
$$a(\mathbf{h}_u, \mathbf{h}_v) = \mathbf{h}_u \cdot \mathbf{h}_v$$



Can use dot product (Cosine) between embeddings to measure attention.

If the embeddings are well aligned we will have a higher score.

Disadvantage? *No learnable weights!*



No learnable weights in our attention function limits our modeling capacity.

Calculate raw attention scores with Neural Network!

$$a(\mathbf{h}_u, \mathbf{h}_v) = \text{LeakyReLU}(\mathbf{a}^T \cdot [\mathbf{W}\mathbf{h}_u || \mathbf{W}\mathbf{h}_v])$$

W is a learnable linear projection matrix.

Applies a linear projection to each of the embeddings separately.

Concatenates the result with target node and then takes the dot product which represents the direction that should be attended to.

Finally apply leaky ReLU for non-linearity.

Normalize raw attention scores with SoftMax

$$a(\mathbf{h}_u, \mathbf{h}_v) = \text{LeakyReLU}(\mathbf{a}^T \cdot [\mathbf{W}\mathbf{h}_u || \mathbf{W}\mathbf{h}_v])$$

$$\alpha_{uv} = \frac{\exp\big(\text{LeakyReLU}(\mathbf{a}^T \cdot [\mathbf{W}\mathbf{h}_u || \mathbf{W}\mathbf{h}_v])\big)}{\sum_{k\in\mathcal{N}(u)} \exp(\text{LeakyReLU}(\mathbf{a}^T \cdot [\mathbf{W}\mathbf{h}_u || \mathbf{W}\mathbf{h}_k]))}$$

$$\mathbf{h}_u = \sigma\left(\sum_{v\in\mathcal{N}_u} \alpha_{uv}\mathbf{W}\mathbf{h}_v\right)$$

Multihead Attention

$$a(\mathbf{h}_u, \mathbf{h}_v) = \mathrm{LeakyReLU}(\mathbf{a}^T \cdot [\mathbf{W}\mathbf{h}_u || \mathbf{W}\mathbf{h}_v])$$

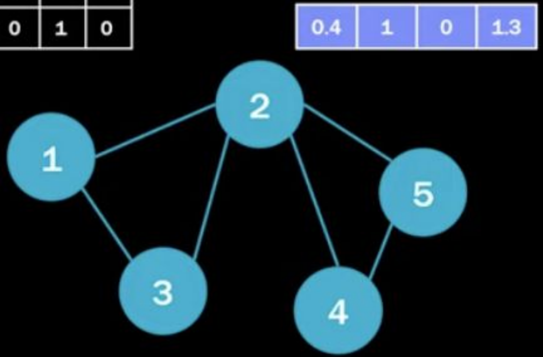
$$\alpha_{uv} = \frac{\exp\left(\mathrm{LeakyReLU}(\mathbf{a}^T \cdot [\mathbf{W}\mathbf{h}_u || \mathbf{W}\mathbf{h}_v])\right)}{\sum_{k \in \mathcal{N}(u)} \exp(\mathrm{LeakyReLU}(\mathbf{a}^T \cdot [\mathbf{W}\mathbf{h}_u || \mathbf{W}\mathbf{h}_k]))}$$

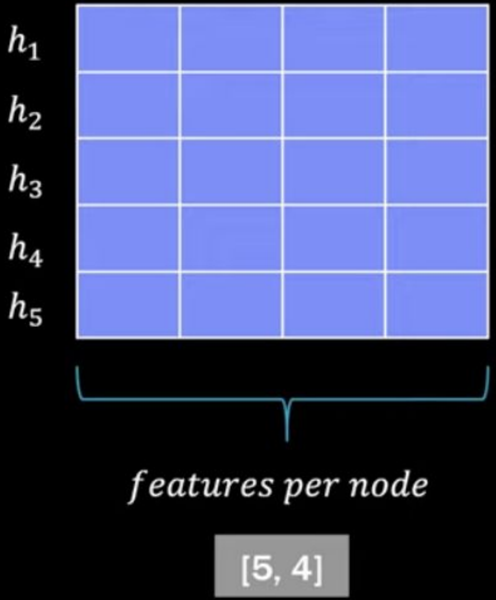
$$\mathbf{h}_u = \Big\|_{k=1}^{K} \sigma\left(\sum_{v \in \mathcal{N}_u} \alpha_{uv}^k \mathbf{W}^k \mathbf{h}_v\right)$$

Matrix Calculation

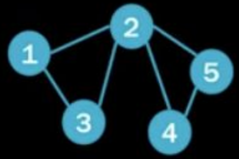
Stack node features into a feature matrix.

0	1	1	0	0
1	0	1	1	1
1	1	0	0	0
0	1	0	0	1
0	1	0	1	0

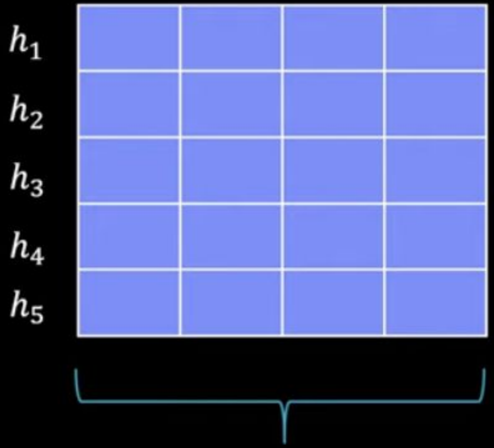




GCN calculation

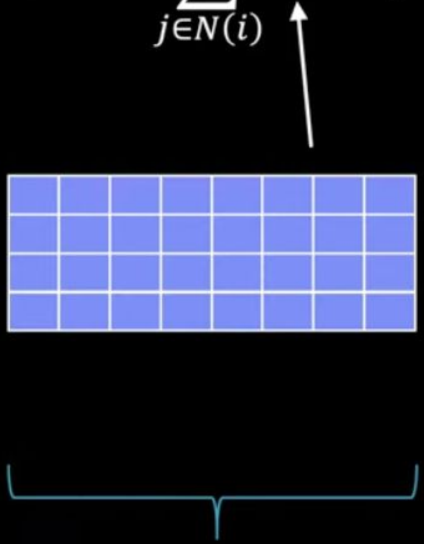


$$h'_i = \sigma(\sum_{j \in N(i)} W * h_j)$$



features per node

[5, 4]



learnable weight matrix

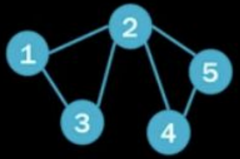
[4, 8]

Multiply features by learnable weight matrix.

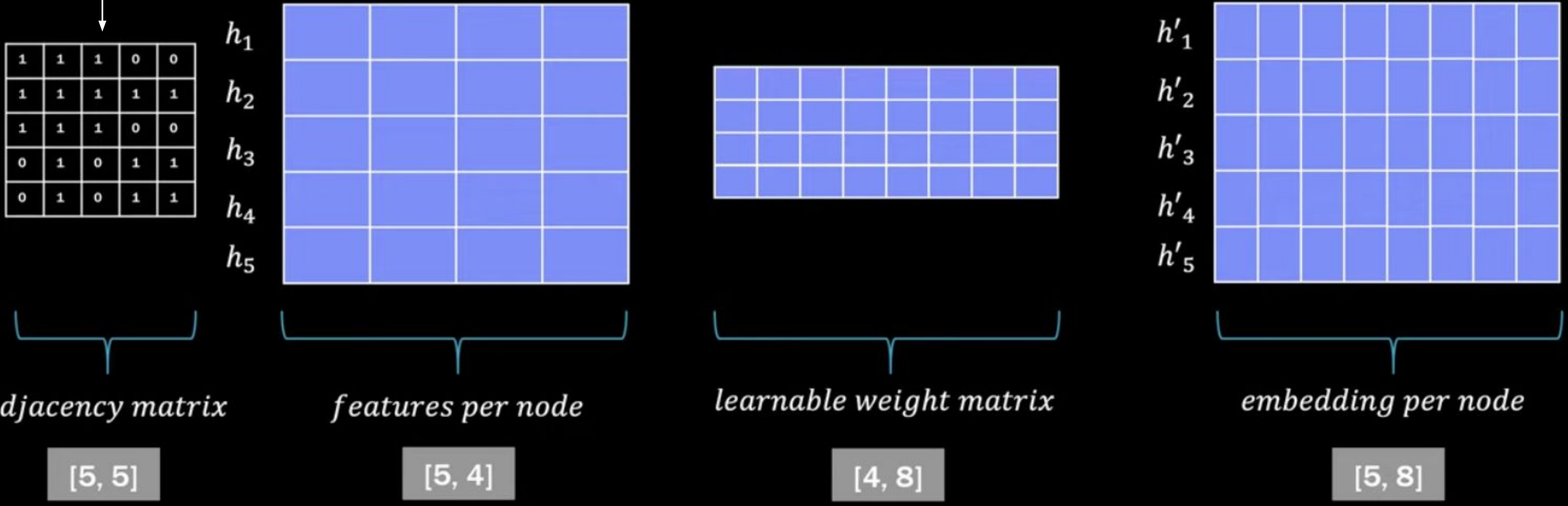
Projects node features into “higher level” embeddings.

Weight matrix comes from a fully connected neural network.

Adjacency matrix defines which neighboring node embeddings to include for each node.



$$h'_i = \sigma\Big(\sum_{j \in N(i)} W * h_j \Big)$$



1	1	1	0	0
1	1	1	1	1
1	1	1	0	0
0	1	0	1	1
0	1	0	1	1

adjacency matrix

[5, 5]

features per node

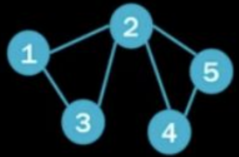
[5, 4]

learnable weight matrix

[4, 8]

embedding per node

[5, 8]

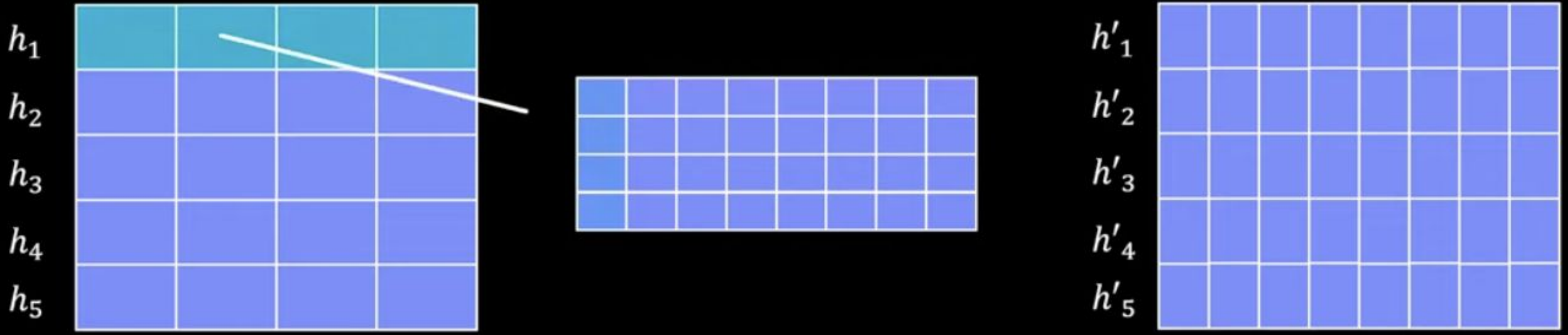


$$h_i' = \sigma\left(\sum_{j \in N(i)} W * h_j \right)$$

1	1	1	0	0
1	1	1	1	1
1	1	1	0	0
0	1	0	1	1
0	1	0	1	1

adjacency matrix

[5, 5]



features per node

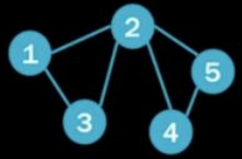
[5, 4]

learnable weight matrix

[4, 8]

embedding per node

[5, 8]

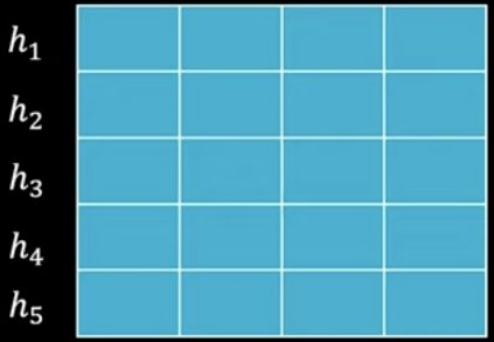


$$h_i' = \sigma\Big(\sum_{j\in N(i)} \underbrace{W * h_j}_{h_j^*}\Big)$$

1	1	1	0	0
1	1	1	1	1
1	1	1	0	0
0	1	0	1	1
0	1	0	1	1

adjacency matrix

[5, 5]

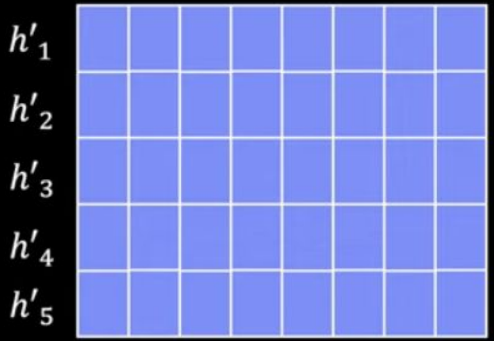


features per node

[5, 4]

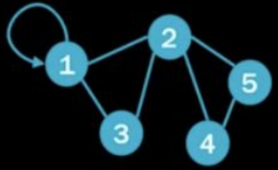
learnable weight matrix

[4, 8]



embedding per node

[5, 8]

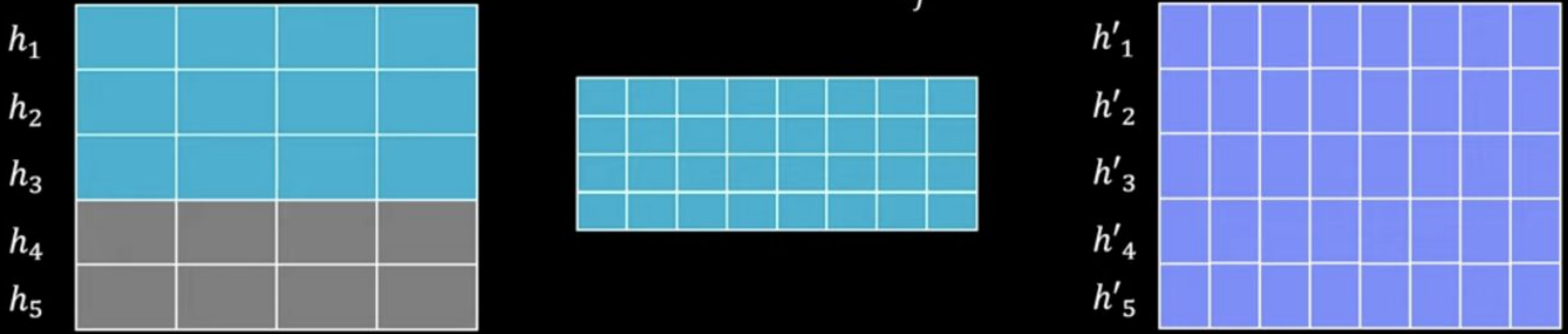


$$h'_i = \sigma\Big(\sum_{j\in N(i)} \underbrace{W * h_j}_{h_j^*}\Big)$$

1	1	1	0	0
1	1	1	1	1
1	1	1	0	0
0	1	0	1	1
0	1	0	1	1

adjacency matrix

[5, 5]



features per node

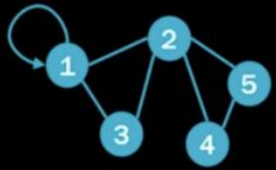
[5, 4]

learnable weight matrix

[4, 8]

embedding per node

[5, 8]

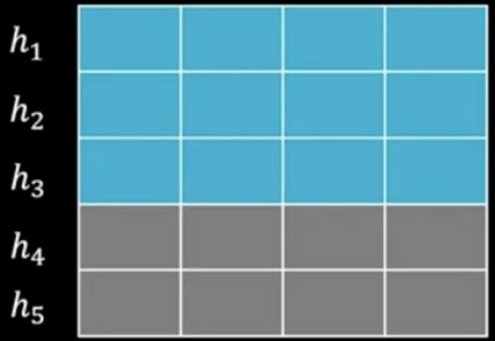


$$h_i' = \sigma\Big(\sum_{j\in N(i)} \underbrace{W * h_j}_{h_j^*}\Big)$$

1	1	1	0	0
1	1	1	1	1
1	1	1	0	0
0	1	0	1	1
0	1	0	1	1

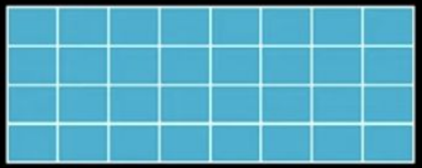
adjacency matrix

[5, 5]



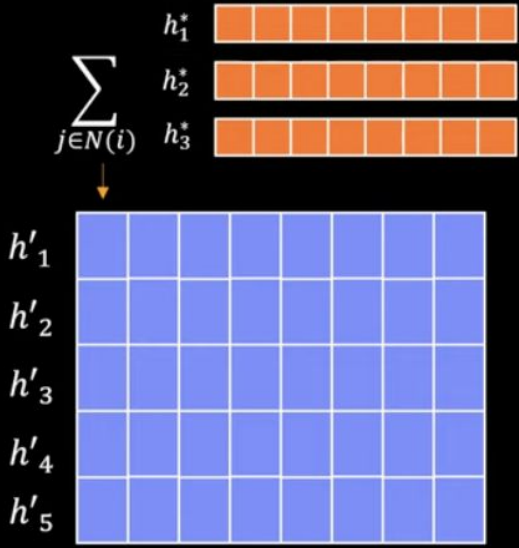
features per node

[5, 4]



learnable weight matrix

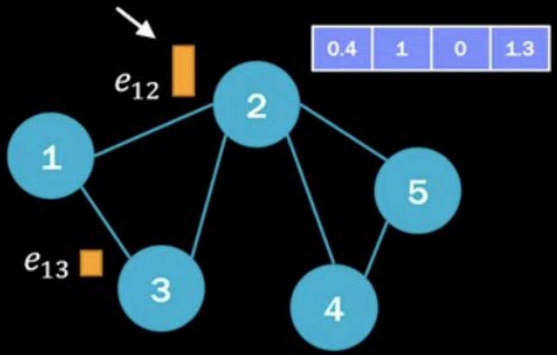
[4, 8]



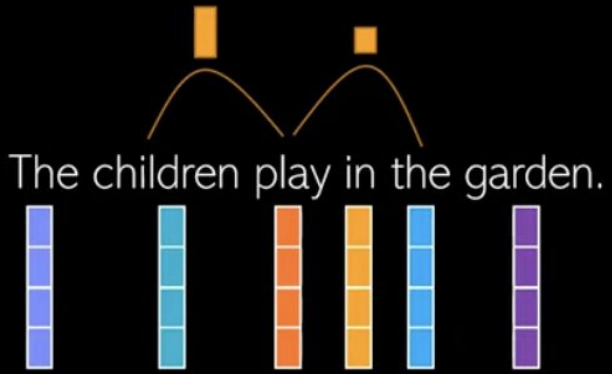
embedding per node

[5, 8]

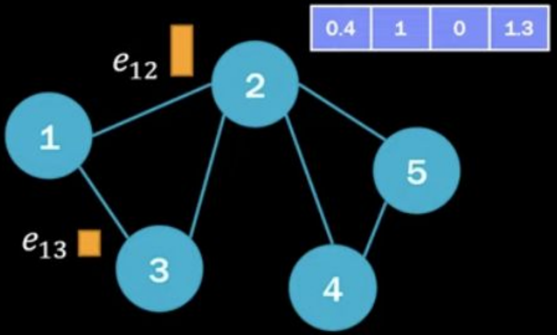
$$\underbrace{e_{ij} = a(Wh_i, Wh_j)}_{attention\ coefficient}$$



$$\underbrace{e_{ij} = a(Wh_i, Wh_j)}_{attention\ coefficient}$$

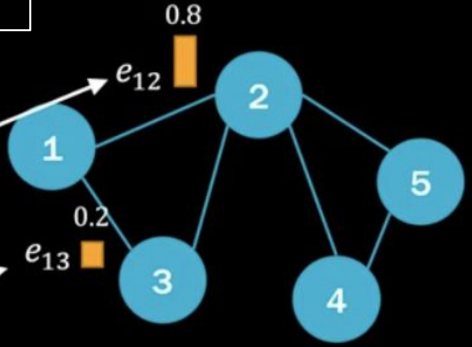


The children play in the garden.



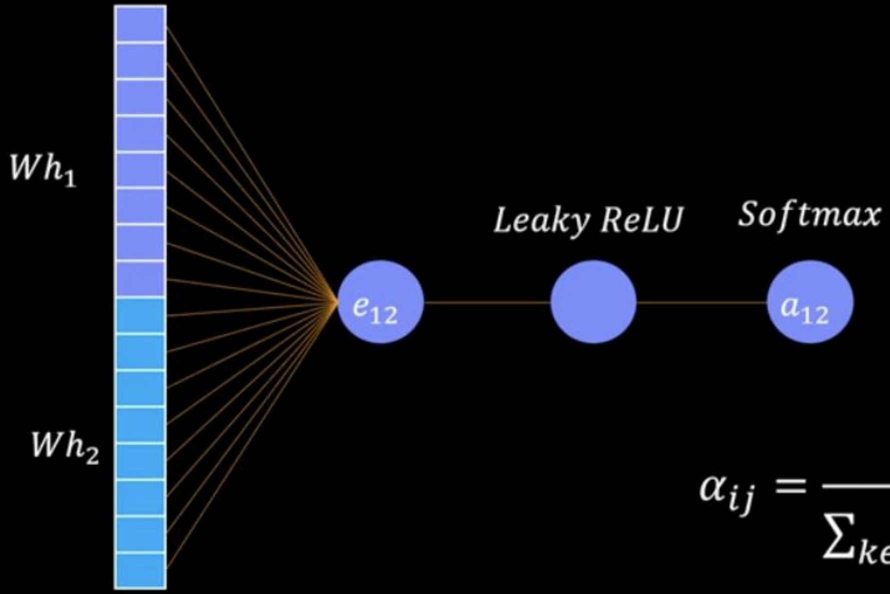
Masked attention - only focus on neighbor nodes

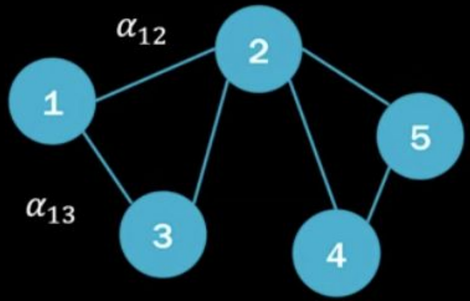
$$\underbrace{e_{ij} = a(Wh_i, Wh_j)}_{attention\ coefficient}$$



$$\underbrace{\alpha_{ij} = softmax_j(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{k\epsilon N(i)} \exp(e_{ik})} = \frac{\exp(a(Wh_i, Wh_j))}{\sum_{k\epsilon N(i)} \exp(a(Wh_i, Wh_j))}}_{normalized\ attention\ coefficient}$$

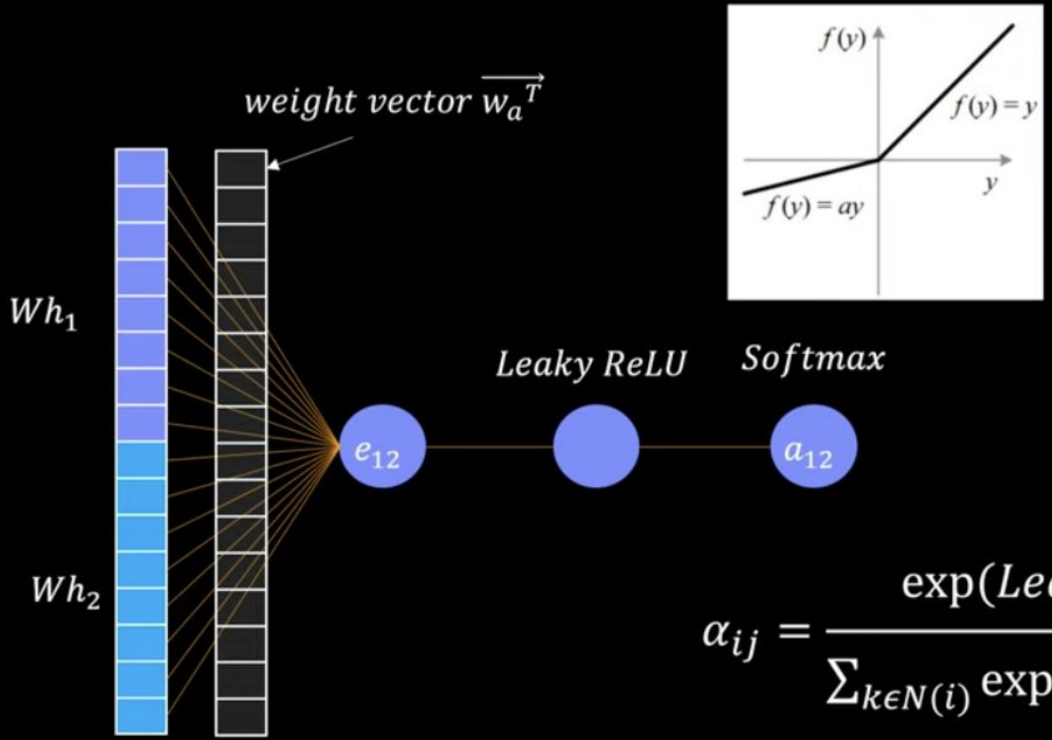
Using a neural network

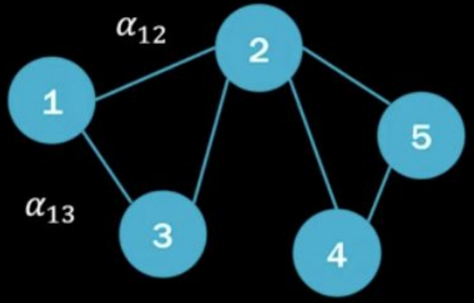




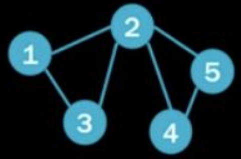
Node feature embeddings are concatenated

$$\alpha_{ij} = \frac{\exp(LeakyReLU\left(\overrightarrow{w_a^T}\ [Wh_i||Wh_j]\right))}{\sum_{k\epsilon N(i)} \exp(LeakyReLU(\overrightarrow{w_a^T}\ [Wh_i||Wh_j]))}$$





$$\alpha_{ij} = \frac{\exp\left(LeakyReLU\left(\overrightarrow{w_a^T}\,[Wh_i||Wh_j]\right)\right)}{\sum_{k \epsilon N(i)} \exp(LeakyReLU(\overrightarrow{w_a^T}\,[Wh_i||Wh_j]))}$$

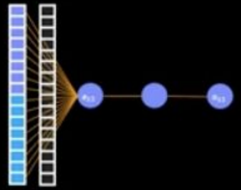


$$h_i' = \sigma\Big(\sum_{j \in N(i)} \alpha_{ij}\, W * h_j\Big)$$

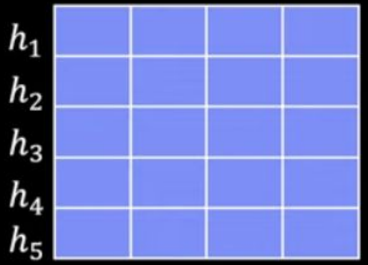
1	1	1	0	0
1	1	1	1	1
1	1	1	0	0
0	1	0	1	1
0	1	0	1	1

adjacency matrix

[5, 5]

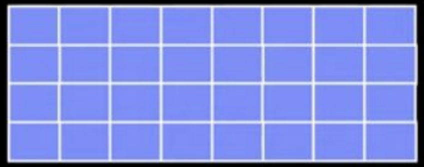


masked attention



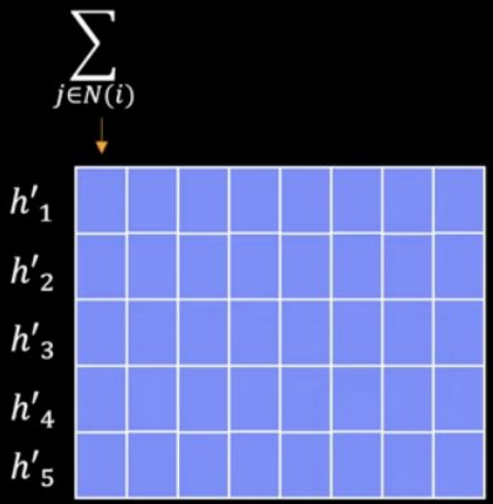
features per node

[5, 4]



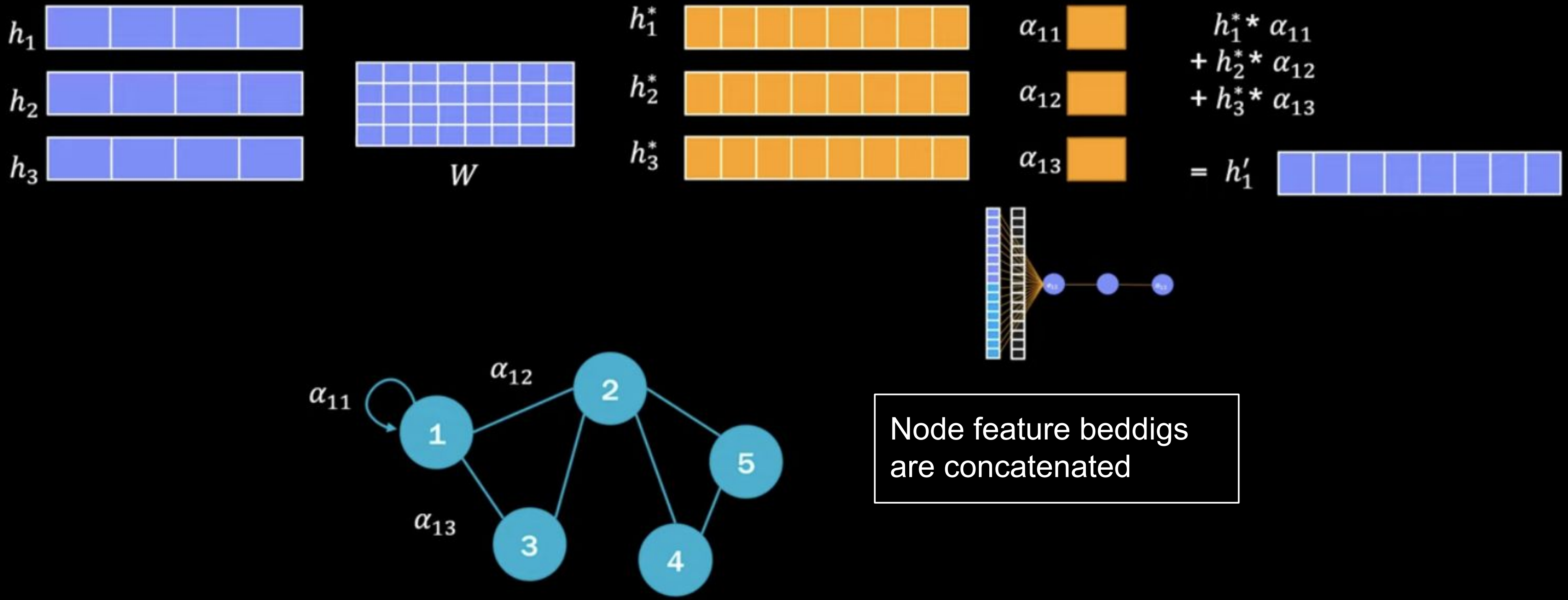
learnable weight matrix

[4, 8]

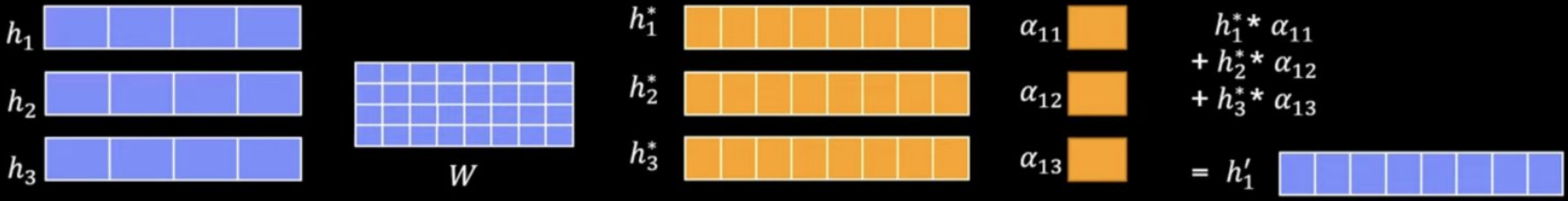


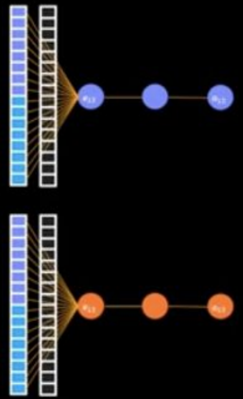
embedding per node

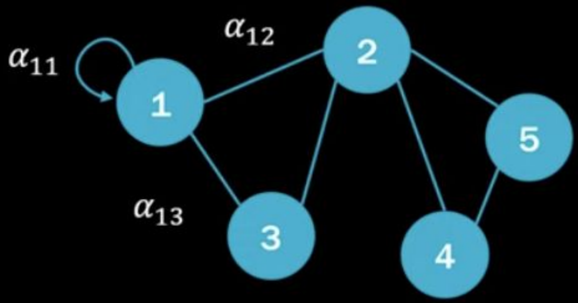
[5, 8]



Node feature beddigs are concatenated







Multihead Attention