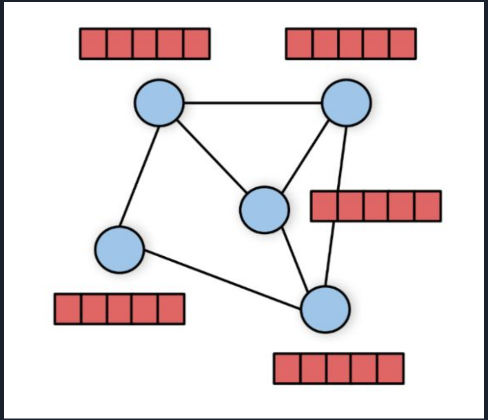
GRAPH ATTENTION NETWORKS

Andrea Morelli 1845525

Neural Network Project AY 2021/2022

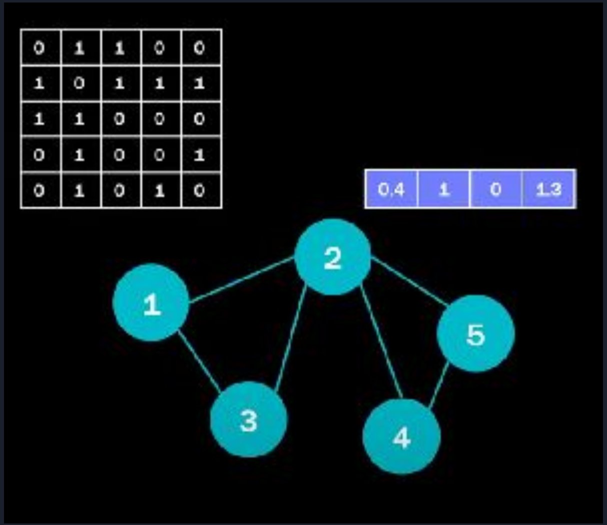
GRAPH ATTENTION NETWORK



What is a graph attention network?

It is a neural network architecture that operates on graph data and leveraging on graph convolutional networks and self attention layers allows to improve the accuracy of previous techniques on common graph datasets including the cora dateset,citeseet,pubmed and ppi.

INPUT OF THE NETWORK

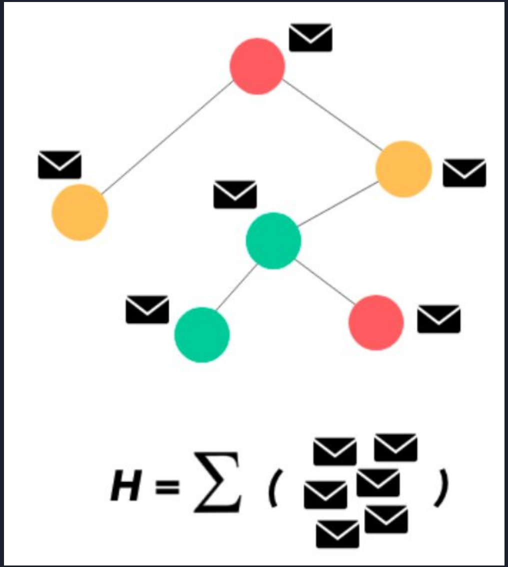


0	1	1	0	0
1	0	1	1	1
1	1	0	0	0
0	1	0	0	1
0	1	0	1	0

0,4	1	0	1.3

In the case of graph data the input is a graph with nodes that have a n dimensional vector representing its embeddings and an adiacency matrix representing the connections between the different nodes of the graph.

GRAPH CONVOLUTIONAL LAYERS



In the case of graph convolutional layer to produce the hidden representation of a node x we would apply a sort of message passing ,where there are parameters to optimize ,between the nodes x and its neighbors, and this operation is done in parallel in all the nodes.
The nodes features are all stacked in a matrix and this matrix is used to derive the hidden features for all the nodes using a W matrix of learnable weights and the adiacency matrix.

GRAPH CONVOLUTIONAL LAYERS

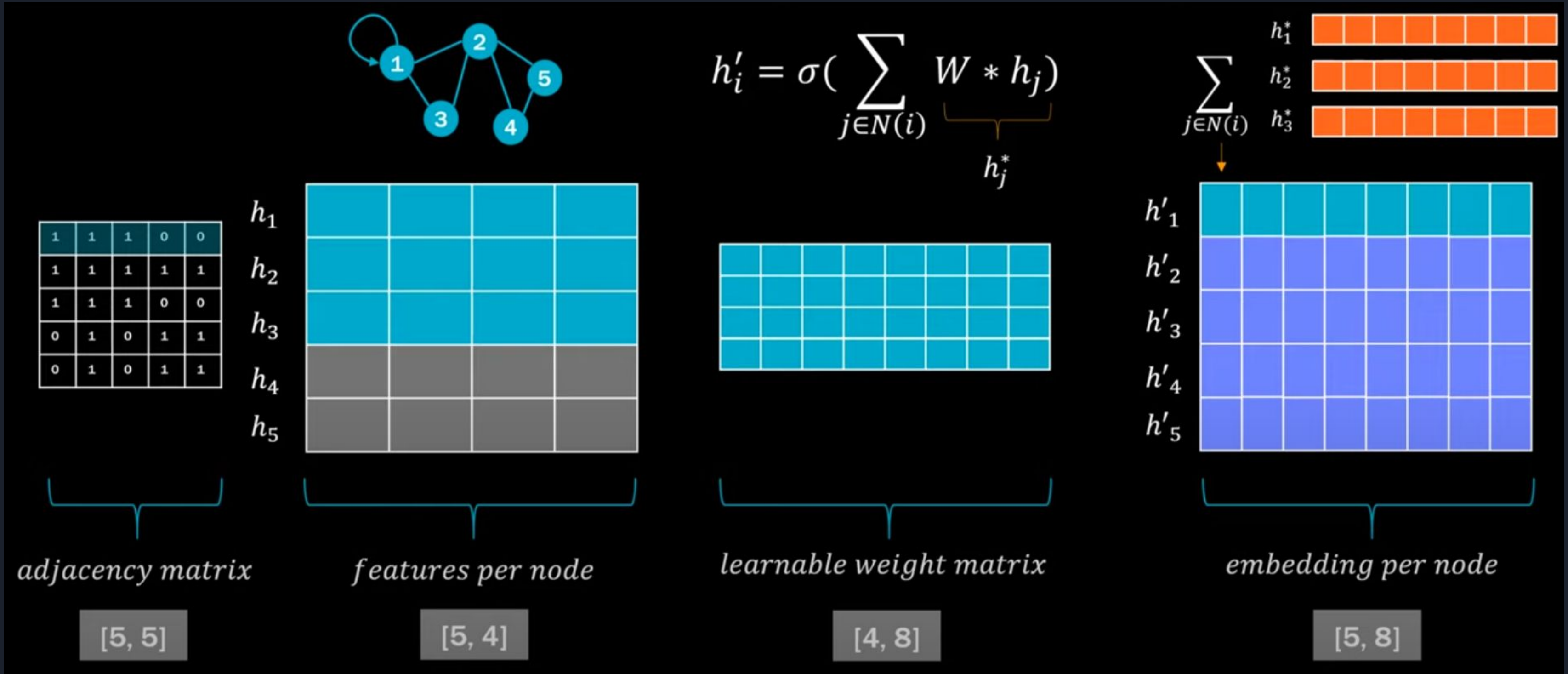
The general formula to obtain the hidden features at layer i is:

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

Where W is the weight matrix,hj are the node embeddings from the last layer of the neightboor j,N is the set of neighbors of the node i and sigma is an activation function.
The aggregation function at the end can be a simple sum or for example an average or a different type of formula even one with learnable parameters.
In practise all of those operations are done in parallel using matrix multiplication and gpu for example in the pytorch geometric package and can be summarized as multiplying the features nodes with the weight matrix and then with the adiacency matrix and at the final step apply an activation function.

GRAPH CONVOLUTIONAL LAYERS

In practise the convolutional layer is implemented as follows:



SELF ATTENTION

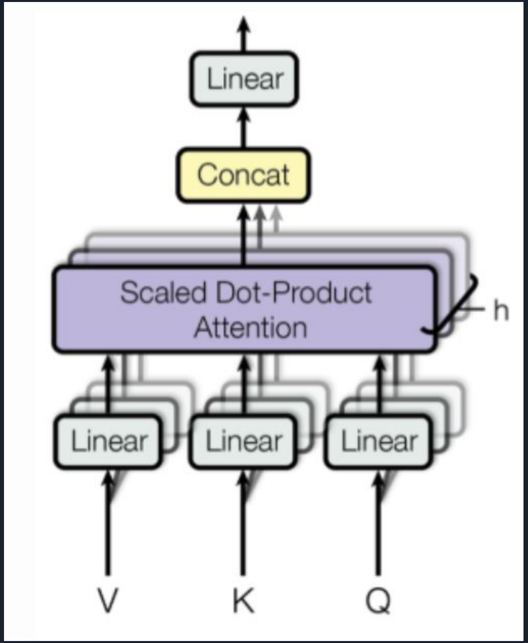
The convolutional layer can be estended using self attention.

One of the most important features of the graph attention network is the use of self attention, a mechanism commonly used in state of the art models in natural language processing , capable of learning how to give attention to different parts of the inputs to give the best possible results. The node i in this case not only has to work with its neighbors as in the graph convolutional layers but has also to learn attention coefficients to give to the neighboors, to weight the features coming from them.

The attention coeffcent that nodes i gives to node j can be seen as $e_{ij} = a(Wh_i, Wh_j)$

where W is the parameters matrix and hi and hj are the nodes embeddings.

SELF ATTENTION



This is what a self attention layer looks like: from the features in input we apply a linear transformation with the matrices V (values),K(keys) and Q(queries)
and we apply a dot product between the queries and keys obtained.Then we normalize those and weight with the attention coefficients obtained the values to obtain the output.
Before seeing how the attention coefficients are obtained in details in the paper let's see how the normalization is performed

SELF ATTENTION

Before describing how the attention function a is implemented lets describe the normalization applied to the coefficients of node i neighboors:
this is the formula used to normaliza that is a simple softmax in order to normalize the coefficients of the neighbors to have a value between 0 and 1 and weighted by their importance

$$\alpha_{ij} = \mathrm{softmax}_j\left(e_{ij}\right) = \frac{\exp\left(e_{ij}\right)}{\sum_{k\in N(i)} \exp\left(e_{ik}\right)} = \frac{\exp\left(a\left(Wh_i, Wh_j\right)\right)}{\sum_{k\in N(i)} \exp\left(a\left(Wh_i, Wh_j\right)\right)}$$

the softmax function firstly exponentiates all the eij attention coefficients to obtain values between 0 and 1 and then returns the percentage of its attention with respect to the attention given to all the neighboors togheter.

Now let's see how the attention coefficients aij are calculated.

SELF ATTENTION

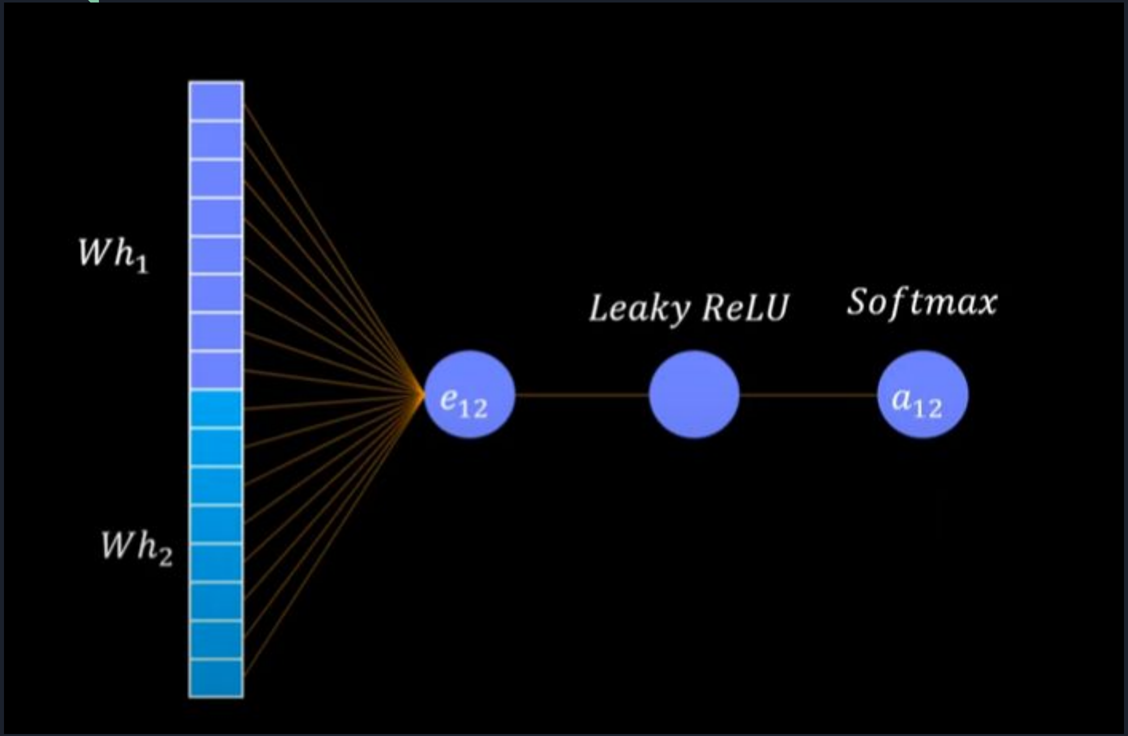
There are different ways to get the attention coefficients, for example in the original paper "attention is all you need" its calculated by using the dot product of the features learned applying the a linear trasformation to the embeddings as in the transformers layers.
However in this paper they opted for a different type of attention mechanism:

they firstly stack the features obtained by multiplying the embeddings with W and then they apply a fully connected layer to get a value that is then passed by a leaky relu activation function before the softmax normalization.
In the softmax normalization there is the need to apply a mask matrix in order to normalize only on the neightboors of node i and not include all the others.
The formula applied at the end to obtain the coefficients normalized is the following:

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

the LeakyReLu is used to cut off all the negative values by a large amount but the entire one and is helpful to emphatize positive relationships found by the model.

SELF ATTENTION



Here is a picture that describe what has been seen in the previous slide:

GRAPH ATTENTION LAYER

Merging the things seen before allows to implement the graph attention layer where there is the use of the graph convolutional layer and the attention coefficients to weight the neighbors messages like a linear combination.
Here is the formula that describes how to obtain the node features:

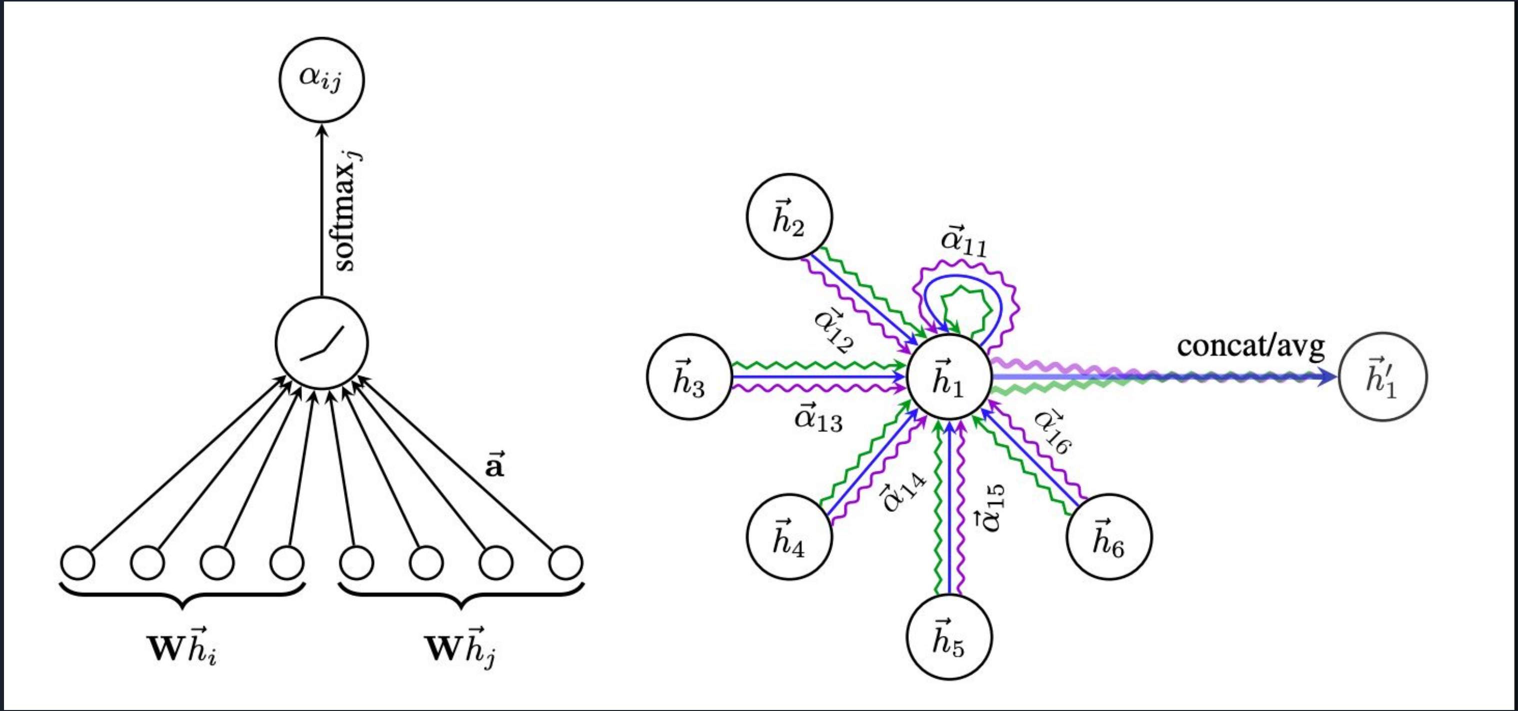
$$h'_i = \sigma\left(\sum_{j \in N(i)} \alpha_{ij} W * h_j\right)$$

the aij are values between 0 and 1 that sums to one and are applied in the summation before the activation function

To summarize ,the learnable weights are the ones of the matrix W used to transform the embeddings and the ones of the attention mechanism (fully connected layer).
In practise we use multihead attention ,so we learn multiple attention coefficients between i and j and we use those to weight the message passing of the neightboors and obtain the features of the output.

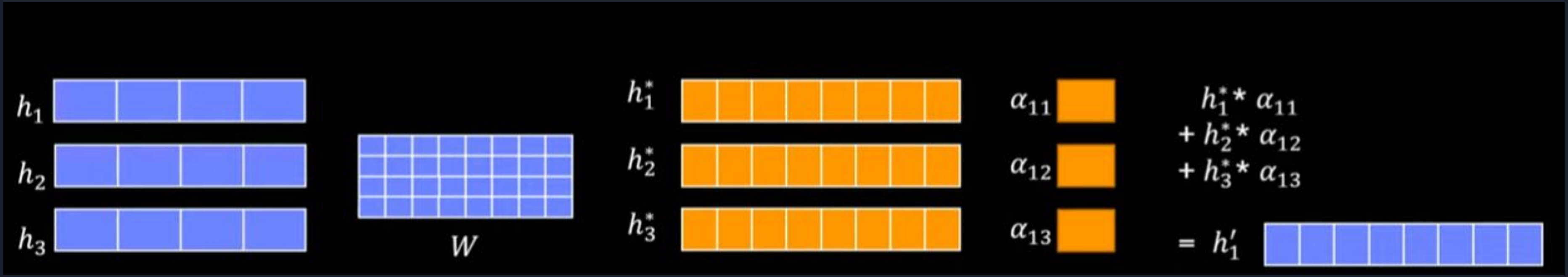
GRAPH ATTENTION NETWORK

A graph attention network is a stack of graph attention layers.
Here is a picture of the entire graph attention layer:



GRAPH ATTENTION NETWORK

Here is a picture of how the operations are performed in practise by libraries like torch geometric to do everything in parallel:



THANKS FOR YOUR ATTENTION