GM Final Project Report

Graph Attention Networks

Team6:趙仰生、黃金興

Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. In International Conference on Learning Representations (ICLR), 2018.

https://iclr.cc/Conferences/2018/Schedule?showEvent=299&fbclid=IwAR10C784VV3gQAbCbGk0drcNgCvVxfDnb6cX1xeNEpDytmBHlOjBUJ3Derg

Paper Introduction:

They introduce an attention-based architecture to perform node classification of graph-structured data. The idea is to compute the hidden representations of each node in the graph, by attending over its neighbors.

The attention architecture has several interesting properties:

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

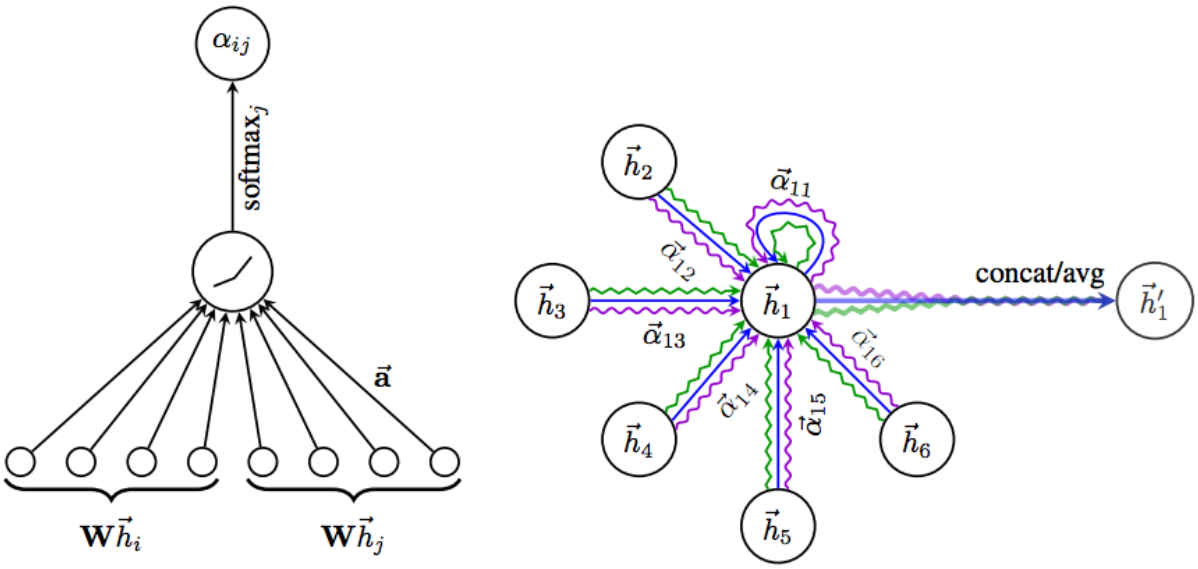
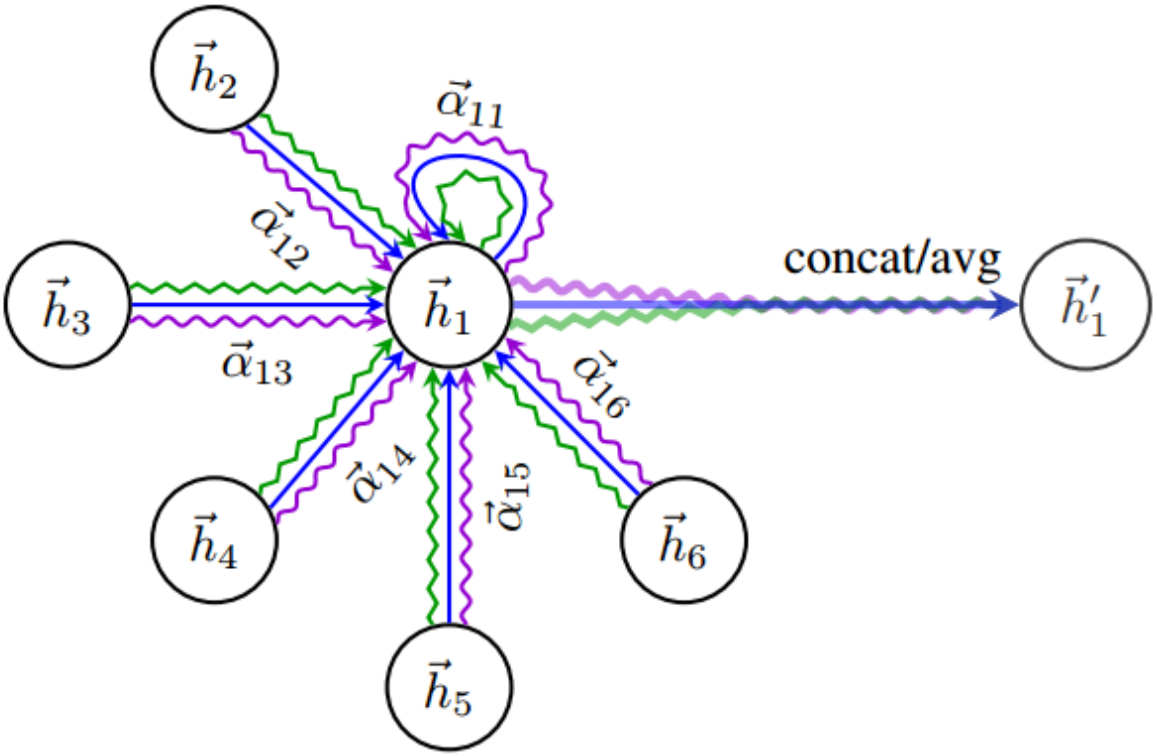


Figure 1: **Left:** The attention mechanism $a(\mathbf{W}\vec{h}_i, \mathbf{W}\vec{h}_j)$ employed by our model, parametrized by a weight vector $\vec{\mathbf{a}} \in \mathbb{R}^{2F'}$, applying a LeakyReLU activation. **Right:** An illustration of multi-head attention (with $K = 3$ heads) by node 1 on its neighborhood. Different arrow styles and colors denote independent attention computations. The aggregated features from each head are concatenated or averaged to obtain $\vec{h}'_1$.

Paper's weaknesses :

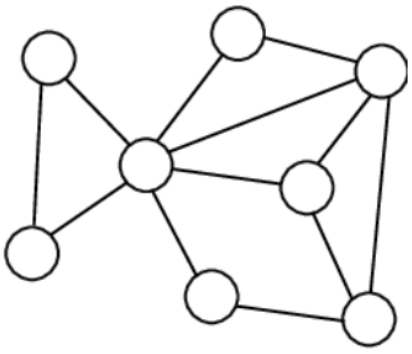
1. This method just apply for the first-order neighborhood => lacks of information from the other neighborhoods.



2. The similarity calculation method in this paper is so simple, just use: Concat + linear or Sum => the similarity maybe was not calculated well.

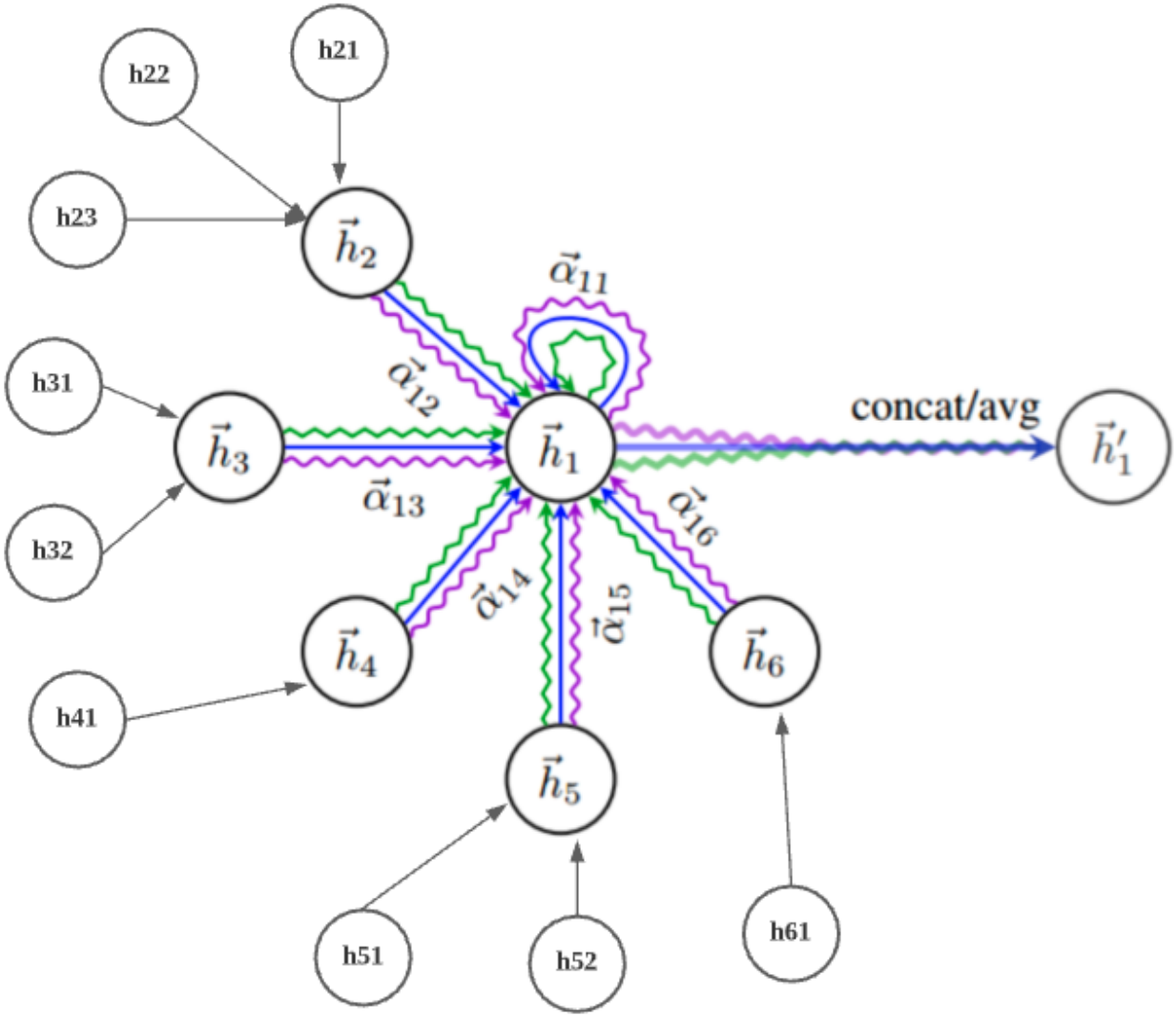
$$\alpha_{ij} = \frac{\exp\left(\text{LeakyReLU}\left(\vec{\mathbf{a}}^T[\mathbf{W}\vec{h}_i \| \mathbf{W}\vec{h}_j]\right)\right)}{\sum_{k\in\mathcal{N}_i}\exp\left(\text{LeakyReLU}\left(\vec{\mathbf{a}}^T[\mathbf{W}\vec{h}_i \| \mathbf{W}\vec{h}_k]\right)\right)}$$

3. The adjacency matrix is just constructed by the undirected graph => It can't reflect all characteristics of the graph.



Address the weaknesses:

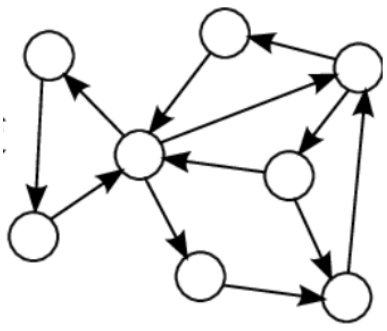
1. Apply attention to second-order or even third-order neighborhood.



2. Change this paper's similarity calculation method to dot production which is the most popular and successful method.

$$\text{softmax}(\frac{QK^T}{\sqrt{d_k}})$$ Dot product

3. Change undirected graph to directed graph.



Experiment Datasets:

“Cora” and “Citeseer”. Below are their profile.

	Cora	Citeseer
Task	Transductive	Transductive
# Nodes	2708 (1 graph)	3327 (1 graph)
# Edges	5429	4732
# Features/Node	1433	3703
# Classes	7	6
# Training Nodes	140	120
# Validation Nodes	500	500
# Test Nodes	1000	1000

Experiment Setting:

For the first two tasks:

We use the open-source of this paper in the link:
(1) https://github.com/PetarV-/GAT
(2) https://github.com/Diego999/pyGAT
Because the original (1) was written in TensorFlow, and we are not too familiar with TensorFlow, so I use (2) source, it was written in PyTorch. Although this source is not official, it was implemented closer to the paper than the office code, they used the *concat+linear* method in the attention calculation, not just the simple *add* vector like the official code (1). And this source (2) code got the performance higher than the official, this source code said they can get accuracy around 84.4 ~ 85.2 for Cora (original code 83.0). But both source code have the limitation, they don't support for the Citeseer dataset, just support Cora dataset, so that we need to implement GAT for Citeseer dataset by ourself, and the performance just get 70.10 (lower than 72.5, the result in paper). We will use that for the baseline for easy comparison with our experiments.

For the third task:

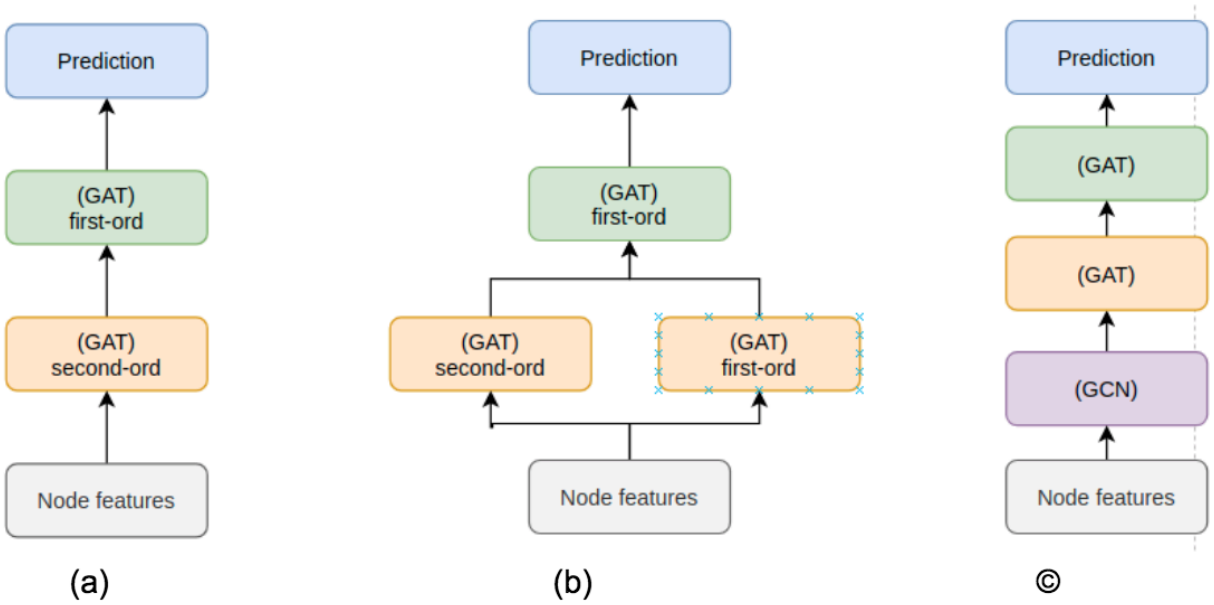
For this part, we build the model and experiment in pytorch. Since the experiment in the paper only train and test on undirected graph, we try this technique(Graph attention) on directed graph, including "Cora" and "Citeseer".
Since our GPU(2060 super 6G) is not good enough, we set:
"hidden node = 4", "K(multi-head) = 3",
and the the hyper parameters as below:
"learning rate = 0.005", "weight_decay = 5e-4", "optimizer = Adam", "criterion = CrossEntropyLoss".
The training process will stop in 350 epochs or there's no improvements during 10 epochs.

Experiment Result:

A. Neighbor Problems

This paper just uses first-order, we think that is not enough, so we tried to use the second-order or even third-order. We tried many approaches:

1. Change the original, first-order to second-order or third-order (a)
2. Use 2 branches, ex: one is first-order, one is second-order, and aggregate the output (b)



In the first approach, we use the second-order instead of the first-order, so the model can learn in larger areas. It helps enlarge the receptive field, the node can learn the information from its neighbor and the second neighbor. We think it can make the information richer and it should be better. We use the middle module for second-order, and the output GAT module just keeps the first-order. The result is depend to the dataset (the middle column in below table):

- Cora dataset: it also lower than the original
- Citeseer dataset: it can get a better result, **71.30 (higher 1.2),**

We saw the Cora performance decrease much as the receptive field becomes larger, and Citeseer performance increase as the receptive field becomes larger. So we continue to try the other case:

instead the middle GAT first-order module to GAT mixed first and second-order (column 3).
=> the performance of Citeseer is also higher than the original, but lower than the second-order case. However with Cora, performance is increase compare with the second-order
However, when we use the third-order for Citeseer, the accuracy was dropped, Cora is a terrible drop. Because it is too large.

	first-order	second-order (in middle) (%)	Mixer second first-order (%)	third-order (in middle) (%)
Citeseer	70.10	**71.30**	**70.90**	69.50
Cora	84.80	82.80	84.60	78.50

In the second approach, we used 2 branches with different order, and then aggregated it by the addition. This approach doesn't drop performance too much for Cora, but the Citeseer also increases because of the enlarged receptive field.

	GAT-original	2 branches (first + second)
Citeseer	70.10	71.10
Cora	84.80	83.80

In GCN+GAT, We also try to combine the GCN with GAT, because We think the GCN will be more stable in the lower layer which just takes embedding the node information to the feature vector, and GAT is used to extract the feature and perform the prediction. As the paper *"CoAtNet: Marrying Convolution and Attention for All Data Sizes"*[1] explains, attention has a dynamic parameter, so it needs more datasets and it is not good to use on some first layers, just suitable for some last layers. We also tried to change some layers

[1] Zihang Dai, Hanxiao Liu, Quoc V. Le, Mingxing Tan, CoAtNet: Marrying Convolution and Attention for All Data Sizes

and drop, but the best performance We got was just 69.20 for Citeseer, lower than the original (70.10).

	GAT-original	GCN+GAT
Citeseer	70.10	69.20
Cora	84.80	83.30

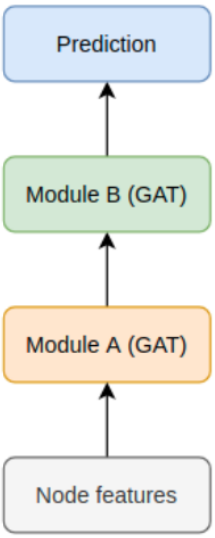
Conclusions:

We see the performance depends on the characteristic of the dataset, ex: the complex, the number of nodes, the connection between nodes. In 2 dataset Cora and Citeseer, the result is totally different.

With the Citeseer, the information from the second-order nodes have an important role, it helps the node en-rich the information. Inversely, with the Cora dataset, the first-order is enough; it doesn't need to enlarge the receptive field. We look back to the size of 2 dataset, the Cora dataset just has 1433 nodes, and the Citeseer has 3703 nodes, greater 2.5 times Cora. So we guess that the large dataset needs more information from the further node, not just first-order nodes, and the small dataset will easily get noise when enlarging the neighbor receptive field.

B. Attention method

The original model like the diagram below:



The model has 2 layers: the first layer is multi-head GAT, it takes the input from the node features and the output is the hidden feature. Here, the number of hidden features is 8, the number of heads is 8. The next layer also is a multi-head GAT, it takes the output from the previous layer, with feature input is 64 (8*8), and output is the class number.

The original method they used was the concat + linear, accuracy for Cora is 84.80 and CiteSeed is 70.10, the speed is very slow, it takes around: 0.2-0.3s for each epoch.

The most popular method for attention is dot production, it takes around 0.02s for each epoch, faster **1x** times than the original one, and the result is also the same, I already try 5 times, and the best one I can get is 85.1, it is near with the source code I used.

Method	Concat+linear speed (s)	dot-product speed (s)
Citeseer	0.2982	0.0234
Cora	0.206	0.0224

The result for dot-product is also not bad, it also fluctuates depending on the initialization value, like the concat+linear methods, I just see the add method is worse. It is hard to say which is the best and what is the worst. However, the performance is very different between these methods.
So, the dot-product method is the best one, it can get good results and the speed is also fast, faster concat+linear 5 times. It can satisfy the accuracy and the speed, so that is why it is the most popular method nowaday.

Method	Concat+linear (%)	dot-product (%)
Citeseer	70.10	70.10
Cora	84.80	84.70 (max 85.10)

Conclusions:
For the attention method, we can get the same result between *concat+linear* (original) and dot-product, but the speed of concat+linear is significantly slower than the speed of dot-product. So that the dot-product is better.

C. Directed Graph

We directed try on different types of graph. The result is as below:

	Undirected(%)	directed(%)
Citeseer	59.5	63.4
Cora	76.2	76.6

Conclusions:

As we can see, the performance of directed graph is better than undirected graph, we think this is because “Attention” technique is directional.

And also the accuracy of ours is a little bit lower than the paper’s claim, this should be the factor of resource. In the paper’s experimental setting, the “hidden node” and “K(multi-head)” is far larger than our setting, which is too huge to our GPU.

However, in our setting, the performance of directed graph is indeed better than undirected graph, therefore, GAT is still better used on directed one.