**Demystifying GCNs: A Step-by-Step Guide to Building a Graph Convolutional Network Layer in PyTorch**

[[J Rosser](https://medium.com/@jrosseruk?source=post_page-----09bf2e788a51--------------------------------)](https://medium.com/@jrosseruk?source=post_page-----09bf2e788a51--------------------------------)

[J Rosser](https://medium.com/@jrosseruk?source=post_page-----09bf2e788a51--------------------------------)

·

[Follow](https://medium.com/m/signin?actionUrl=https%3A%2F%2Fmedium.com%2F_%2Fsubscribe%2Fuser%2F47962184be45&operation=register&redirect=https%3A%2F%2Fmedium.com%2F%40jrosseruk%2Fdemystifying-gcns-a-step-by-step-guide-to-building-a-graph-convolutional-network-layer-in-pytorch-09bf2e788a51&user=J+Rosser&userId=47962184be45&source=post_page-47962184be45----09bf2e788a51---------------------post_header-----------)

6 min read

·

Jan 19

38

1

Graph Neural Networks (GNNs) have emerged as a powerful class of neural networks, designed to capture the complexity and relational information inherent in graph-structured data.

Unlike traditional neural networks, GNNs excel in scenarios where data points are interconnected, such as social networks, molecular structures, and transportation systems.

What you can find in this article:

* 🧮 A (hopefully) easy-to-follow mathematical proof
* 🔥 PyTorch implementation
* 📓 Some extra resources

**Party Planning: Intro to GNNs!**

Let’s dive into a quick example to show why you might prefer using a GNN over a traditional neural network…

🌟 Imagine you’re planning a big, fun party with all your friends! 🎉

You’d like to choose a seating arrangement such that everyone has the best time possible, and because you’re cool 😎, you’re going to use machine learning to help!

With a traditional Neural Network, you’d consider each friend individually, possibly overlooking how they connect with others. This approach might lead to mismatched seating, where friends who’d enjoy each other’s company are separated.

🌐Enter Graph Neural Networks (GNNs)! ✨

Graph Neural Networks (GNNs) are like party planners with a superpower to see the connections between all your friends. They can look at the whole network of friendships, who knows whom, who shares interests, and who might enjoy meeting each other. A GNN will know to seat Donna next to Martha because they both know John Smith and have worked at UNIT.

Everyone has a great time because the GNN understands the complex web of relationships and interactions!

**Graph Convolutional Networks**

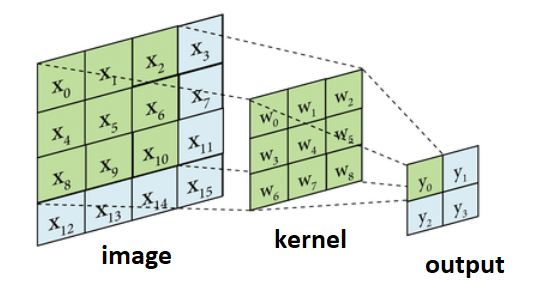
Alrighty, so what’s the deal with Graph Convolutional Networks (GCNs)?

Much like traditional neural networks, onions and ogres, GNNs are composed of layers.



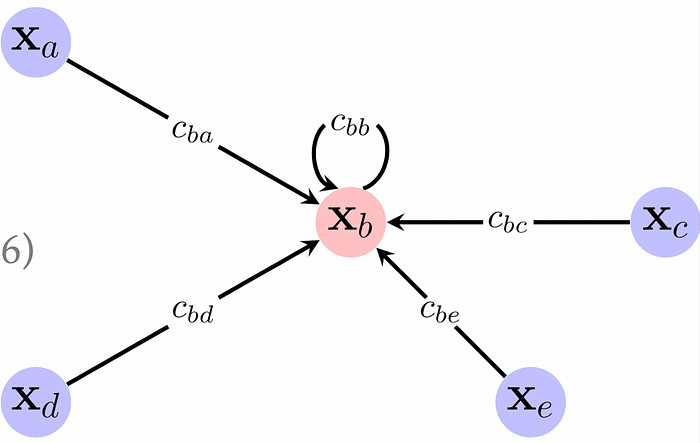
One of the fundamental layers in deep learning is the Graph Convolutional Network (GCN) layer, which can be thought of as being similar in function to a convolutional layer in a Convolutional Neural Network (CNN).

In a CNN, the convolutional layer gathers and processes information from surrounding pixels, known as the “receptive field,” to create a condensed, lower-dimensional representation:



Source: <https://medium.com/@thepyprogrammer/2d-image-convolution-with-numpy-with-a-handmade-sliding-window-view-946c4acb98b4>

A GCN layer acts similarly, however in this case instead of neighbouring pixels, we aggregate information from neighbouring nodes in the graph (as well as the node itself):

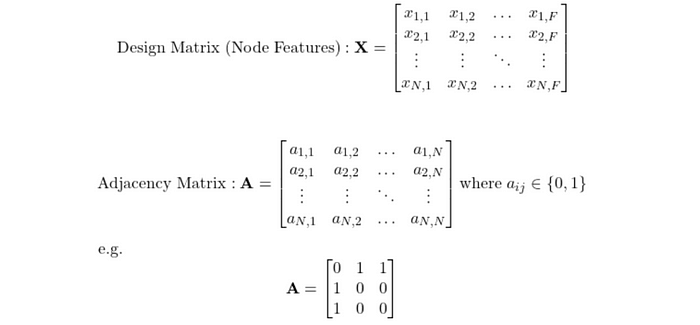


Source: <https://a-j.gitbook.io/geometric-deep-learning/graphs-ii>

**Deriving the equation for Graph Convolutional Networks!**

A great video on this subject: <https://www.youtube.com/watch?v=CwHNUX2GWvE&ab_channel=FedericoBarbero>

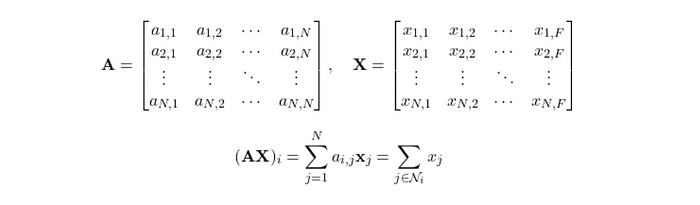
The input to a GNN is a graph! We can represent this as a (design) matrix of node features and an Adjacency Matrix comprising of 1s indicating connections between nodes and 0s otherwise:



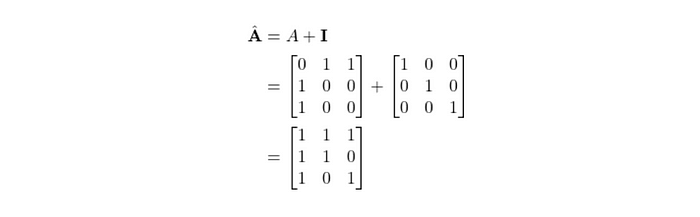
So this example adjacency matrix would be for the graph:

Node 1 -- Node 2  
 |  
 Node 3

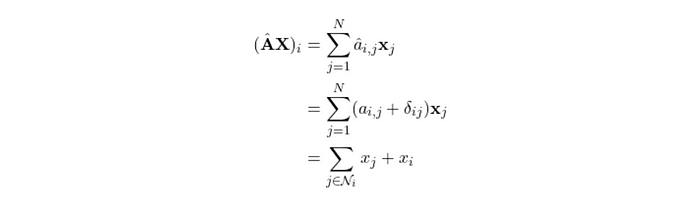
If we were to multiply **A** by **X**, we could convince ourselves that the result is the sum of all neighbours of the nodes for each feature. Let’s consider node **i**, we can see that multiplying **A** with **X** results in just summing the features of the neighbours **j**.



Now, the eagle-eyed amongst you might be thinking, but what about the node itself’s feature vector, that must be important! We can include this by adding 1s down the diagonal, and mathematically this is represented by adding an identity matrix:

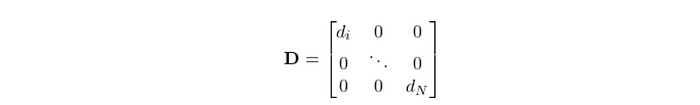


Such that:

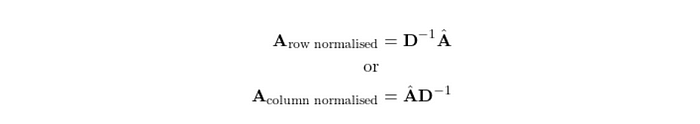


Very nice! However we’ve got another problem, we need to normalise this summation as some nodes may have hundreds of neighbours and others might just have one or two!

One way to do this is to divide by the number of neighbours for each node, known as the node degree. Our first intuition would be to create a diagonal degree matrix **D**where the diagonals represent node degree:



And so the following would normalise our equation with respect to columns or rows:



Intuitively, row normalisation is effectively taking the mean value of the neighbours, and column normalisation instead takes into account the number of neighbours the neighbours have.

To get the best of both worlds, we implement symmetric normalisation as follows:



This takes into account both the current node’s number of neighbours and the neighbours’ number of neighbours.

Great stuff! Our equation is taking shape!

Now, we just need some parameters for our machine learning model to learn, hopefully it makes sense to just pop a weight matrix in there like we do with linear regression etc.!



And because we know that adding non-linearity allows for a much better representation, we can add a ReLU ontop!

Finally…



**PyTorch Implementation**

Now comes the PyTorch implementation!

In the \_\_init\_\_ method, we initialise **A, D,**and **W**.

In the forward pass, we construct **H** from the components.

import torch  
import torch.nn as nn  
import torch.nn.functional as F  
  
class GCNLayer(nn.Module):  
 """  
 GCN layer  
  
 Args:  
 input\_dim (int): Dimension of the input  
 output\_dim (int): Dimension of the output (a softmax distribution)  
 A (torch.Tensor): 2D adjacency matrix  
 """  
  
 def \_\_init\_\_(self, input\_dim: int, output\_dim: int, A: torch.Tensor):  
 super(GCNLayer, self).\_\_init\_\_()  
 self.input\_dim = input\_dim  
 self.output\_dim = output\_dim  
 self.A = A  
  
 # A\_hat = A + I  
 self.A\_hat = self.A + torch.eye(self.A.size(0))  
  
 # Create diagonal degree matrix D  
 self.ones = torch.ones(input\_dim, input\_dim)  
 self.D = torch.matmul(self.A.float(), self.ones.float())  
  
 # Extract the diagonal elements  
 self.D = torch.diag(self.D)  
  
 # Create a new tensor with the diagonal elements and zeros elsewhere  
 self.D = torch.diag\_embed(self.D)  
   
 # Create D^{-1/2}  
 self.D\_neg\_sqrt = torch.diag\_embed(torch.diag(torch.pow(self.D, -0.5)))  
   
 # Initialise the weight matrix as a parameter  
 self.W = nn.Parameter(torch.rand(input\_dim, output\_dim))  
  
 def forward(self, X: torch.Tensor):  
  
 # D^-1/2 \* (A\_hat \* D^-1/2)  
 support\_1 = torch.matmul(self.D\_neg\_sqrt, torch.matmul(self.A\_hat, self.D\_neg\_sqrt))  
   
 # (D^-1/2 \* A\_hat \* D^-1/2) \* (X \* W)  
 support\_2 = torch.matmul(support\_1, torch.matmul(X, self.W))  
   
 # ReLU(D^-1/2 \* A\_hat \* D^-1/2 \* X \* W)  
 H = F.relu(support\_2)  
  
 return H  
  
if \_\_name\_\_ == "\_\_main\_\_":  
  
 # Example Usage  
 input\_dim = 3 # Assuming the input dimension is 3  
 output\_dim = 2 # Assuming the output dimension is 2  
  
 # Example adjacency matrix  
 A = torch.tensor([[1., 0., 0.],  
 [0., 1., 1.],  
 [0., 1., 1.]])   
  
 # Create the GCN Layer  
 gcn\_layer = GCNLayer(input\_dim, output\_dim, A)  
  
 # Example input feature matrix  
 X = torch.tensor([[1., 2., 3.],  
 [4., 5., 6.],  
 [7., 8., 9.]])  
  
 # Forward pass  
 output = gcn\_layer(X)  
   
 print(output)  
 # tensor([[ 6.3438, 5.8004],  
 # [13.3558, 13.7459],  
 # [15.5052, 16.0948]], grad\_fn=<ReluBackward0>)

**And… we’re done! 🎉🎉🎉**

Thank you so much for reading this far, please don’t hesitate to comment any corrections or any other cool things!

Seeya in the next one!

**Resources**

* <https://geometricdeeplearning.com/lectures/>
* <https://www.youtube.com/watch?v=CwHNUX2GWvE&ab_channel=FedericoBarbero>

[Graph Neural Networks](https://medium.com/tag/graph-neural-networks?source=post_page-----09bf2e788a51---------------graph_neural_networks-----------------)

[Graph Convolution Network](https://medium.com/tag/graph-convolution-network?source=post_page-----09bf2e788a51---------------graph_convolution_network-----------------)

[Machine Learning](https://medium.com/tag/machine-learning?source=post_page-----09bf2e788a51---------------machine_learning-----------------)

[Artificial Intelligence](https://medium.com/tag/artificial-intelligence?source=post_page-----09bf2e788a51---------------artificial_intelligence-----------------)