ICMA 393: Graph Theory & Combinatorics Project Report

Nathan Sakchiraphong

March 2024

An Introduction to Graph Neural Networks

1 Introduction

Graphs are common data structures used to show relationships between things, like in social networks or recommendation systems, present in almost every facet our lives.

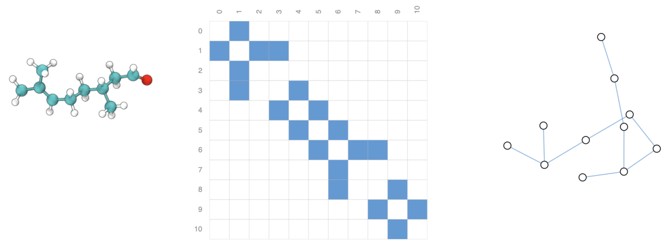


Figure 1: Representation of a Citronellal molecule as a Graph

Traditional machine learning often struggle with such data because of the relationships that data points might have with each other. Graph Neural Networks take advantage of the structure and have become the way to solve these kinds of questions. This presentation will hopefully serve as a gentle introduction to Graph Neural Networks and give you an intuitive understanding of how they work, with the help of a toy problem.

2 Motivation

I don't know about you but whenever I hear things about a new machine learning model or an advancement made to neural networks, it always seems like a magical black box to me. You put data in, and out comes a prediction. I don't really know whats going on between data going in and a prediction coming out. So I'm taking this as an opportunity to learn more about the behind the scenes of neural networks. That is why all the code for this project is written purely using numpy.

Also I'm a little salty about not being able to take Deep Learning this semester.

3 The Problem

The dataset that I will be using for the purpose of this presentation is called **Zachary's Karate Club**. It captures 34 members of a karate club, where the members are the nodes and the edges denote the interaction of members outside the club. The story goes that an argument occurred that split the club into 2.

In this problem we will be trying to predict which side a member chose in the argument based on the interactions that they had outside the club.

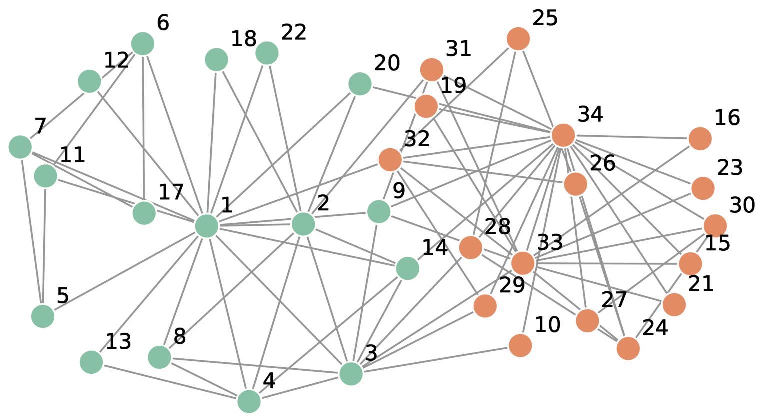


Figure 2: Zachary's Karate Club after the split

4 Approach

Lets start thinking if this question in the same way you would usually think of any other classification problem. In its simplest form, the data is nothing but:

Member	**Side**
1	A
2	B
. . .	. . .

But we also need to add the relationship that each member has with each other. one way to do this would be:

Member	**Side**	**Interacted with 1**	**Interacted with 2**	. . .
1	A	0	3	. . .
2	B	3	0	. . .
. . .	. . .	. . .	. . .	. . .

You can see that towards the right, you start to get something that might look familiar. An adjacency matrix of interactions. This is the edge feature of this dataset. Another major component and the one we're trying to predict here is the label or the node feature of the graph. Depending on whether you have directed or undirected relationships between data points, the adjacency matrix would look different, but this problem uses undirected edges so we have a symmetrical adjacency matrix.

Representing the interactions as columns would be pretty wasteful and difficult to understand. That is why such problems are often posed as graph problems.

5 GNNs

Now lets dive a little deeper and understand how a graph neural network can be used to solve the problem.

Consider a neural network that takes an image in its first layer. The pixels of the image are aggregated is then passed through a function of some kind, called the activation function, before becoming input for the next layer.

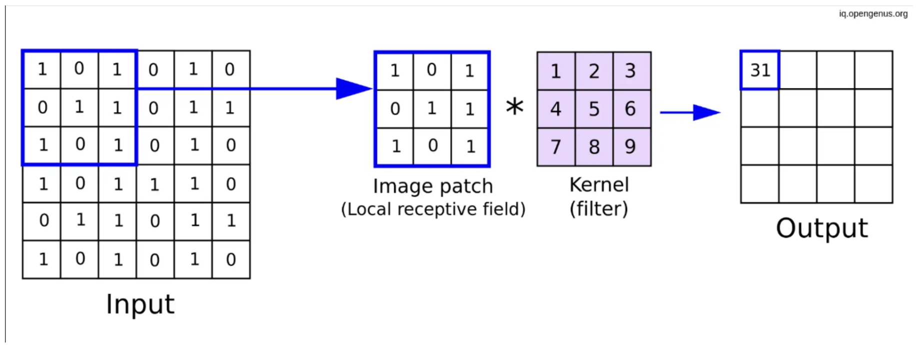


Figure 3: Traditional Neural Network: Input → Hidden Layer 1

When we think about it, this same principal can be applied to graphs as well. We can think of each pixel being a node its surround 8 pixels being the neighbourhood within which its being aggregated. In the context of a graph, this aggregation process is usually referred to as **Message Passing**. Just like aggregation that happens at each layer of a neural network, message passing works in the same way to provide a final understanding of each node in the context of the graph.

6 Message Passing

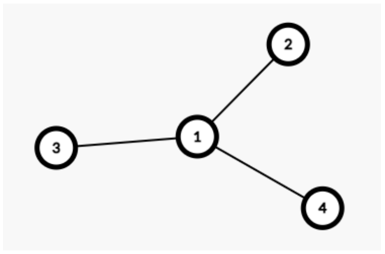


Figure 4: Example Graph

To explain this concept in a more mathematical way we can think of one round of message passing as a function f. Let x_1, be node 1 in the graph. So $f(x_1)$ is the function that will update the state of that node after message passing.

We can think of what $f(x_1)$ can be in a few steps. We know that the first step is an aggregation. This can differ based on the method so I'll just denote it with +. It's not really a sum though.

$$f(x_1) = x_1 + x_2 + x_3 + x_4$$

But we might want to give different importance values to each neighbour based on the weight of the edge. $c_{i,j}$ will be used here to denote the edge of the edge from i to j. These are usually fixed and come with the structure of the graph.

$$f(x_1) = c_{1,1}x_1 + c_{1,2}x_2 + c_{1,3}x_3 + c_{1,4}x_4$$

We also know that each node has its own set of features. For which we would like to give some importance to. We can all this W to be the set of learnable weights that will get updated in the neural network as it trains.

$$f(x_1) = c_{1,1}x_1W + c_{1,2}x_2W + c_{1,3}x_3W + c_{1,4}x_4W$$

Much like a traditional neural network, in order to solve more complex issues we would also like to add an activation function (σ) to give it some non-linearity.

$$f(x_1) = \sigma(c_{1,1}x_1W + c_{1,2}x_2W + c_{1,3}x_3W + c_{1,4}x_4W)$$

Finally, we can generalize this for all the nodes, not just x_1. Using N_i as the set of neighbours of i.

$$f(x_i) = \sigma(x_i, \sum_{j \in N_i} c_{i,j} \cdot x_j \cdot W)$$

This type of message passing gives rise to the convolution model of graph neural networks that I am focusing on today.

While this may seem tedious for each node to perform one by one, in practice, the entire process is streamlined using matrix multiplication.

$$\begin{bmatrix} c_{11} & c_{12} & c_{13} & c_{14} \\ c_{21} & c_{22} & c_{23} & c_{24} \\ c_{31} & c_{32} & c_{32} & c_{34} \\ c_{41} & c_{42} & c_{43} & c_{44} \end{bmatrix} \times \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} \times W = \begin{bmatrix} f(x_1) \\ f(x_2) \\ f(x_3) \\ f(x_4) \end{bmatrix}$$

$$f(x_1) = c_{1,1}x_1W + c_{1,2}x_2W + c_{1,3}x_3W + c_{1,4}x_4W$$

This gives us a rather simple final equation for message passing where A is the adjacency matrix with the weights, X is the matrix of node features and W are the learnable weights.

$$f(X) = \sigma(AXW)$$

7 Implementation

Setup

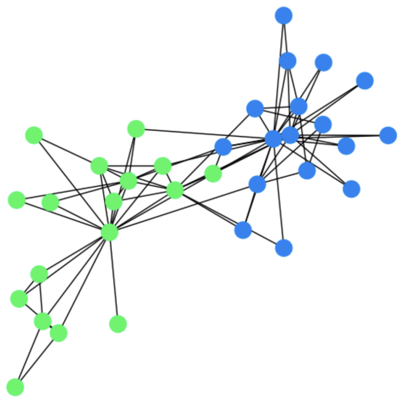


Figure 5: Zachary's Karate Club visualized with NetworkX

From earlier we now know that we need 3 things to start up our neural network.

A: An adjacency matrix where instead of using ones and zeros, the edge weights are used as values of connection. Node that message passing also includes the node itself so that diagonals will contain 1. Further normalization is done on it in accordance to Kipf and Welling's paper.

X: Node feature matrix, Since we have no node features other than the label, X is initialized as an identity matrix.

W: A set of randomly generated weights using Xavier's Initialisation

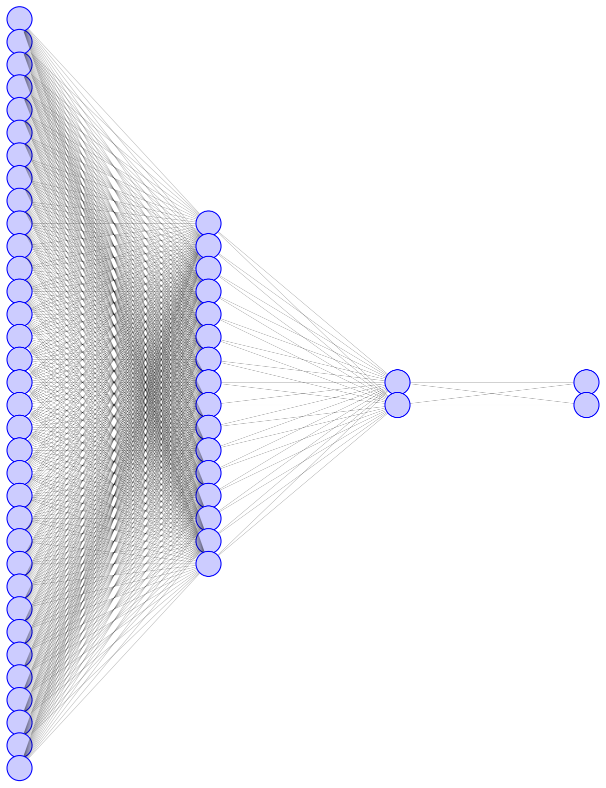
$$x = \sqrt{\frac{6}{inputs + outputs}}$$

Besides this we also need a loss function to be able to learn through back propagation. This loss function is called Cross-Entropy Loss and is generally used for classification models.

$$L(y, \hat{y}) = -\sum_i y_i \log(\hat{y}_i)$$

Neural Network

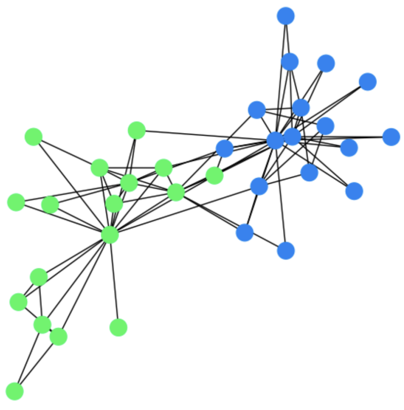
For more detail, I wouldn't mind sharing the Jupyter notebook, however to go over it briefly, The Neural Network is set up to have an input layer that takes in the whole dataset, 2 hidden layers of size 16 and 2 on which message passing is performed until we're left with two values. These two values go through final a Softmax layer that returns the normalized probability of the prediction.



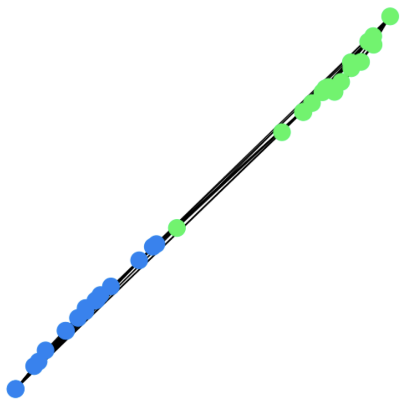
Training

```
//Setup

np.random.seed(100)
gcn_model = GCNNetwork(
    n_inputs=n_nodes,
    n_outputs=n_labels,
    hidden_sizes=[16, 2],
    activation=np.tanh,
)

train_nodes = np.array([0, 1, 32, 33]) # 4/34 vertices as training
test_nodes = np.array([i for i in range(labels.shape[0]) if i not in
     train_nodes]) # The rest is for testing
# Optimiser(learning_rate, weight_decay)
opt = Optimiser(2e-2, 2.5e-2, train_nodes)
```

```
//Training
# Stores node embeddings before passing them through the Softmax
     function.
# Used in visualisation
embeddings = []
accuracy = []
train_losses = []
test_losses = []

n_iters = 500
# keeps updating to the lowest value of loss to stop early in case there
     is no more improvement
loss_min = 1e6
# Running count of iterations without improvemnets
es_iters = 0
# Max allowed iterations without improvements
es_steps = 50

for iter in range(n_iters):

    # Forward Propagation
    embedding, y_pred = gcn_model.forward(A_hat, H)
    embeddings.append(embedding)

    opt.y_pred = y_pred
    opt.y_true = labels

    # Back Propagation
    gcn_model.output_layer.backward(opt)
    for layer in reversed(gcn_model.layers):
        layer.backward(opt)

    # Accuracy for non-training nodes
    predicted_label = np.argmax(y_pred, axis=1)
    actual_label = np.argmax(labels, axis=1)
    acc = (predicted_label == actual_label)[test_nodes]
    accuracy.append(acc.mean())

    # Cross-Entropy Loss Calculation
    loss = loss_calc(y_pred, labels)
    loss_train = loss[train_nodes].mean()
    loss_test = loss[test_nodes].mean()

    train_losses.append(loss_train)
    test_losses.append(loss_test)

    # Checking to see if the model has gone a significant number of
        iterations without improving
    if loss_test < loss_min:
        loss_min = loss_test
        es_iters = 0
    else:
        es_iters += 1

    if es_iters > es_steps:
        print("Early stopping!")
        break

    if iter % 100 == 0:
        print(f"iter: {iter + 1}, Train Loss: {loss_train:.3f}, Test
            Loss: {loss_test:.3f}")
```



(a) Original Embedding



(b) Learned Embedding

Figure 6: Neural Network Output

8 Results

Loss

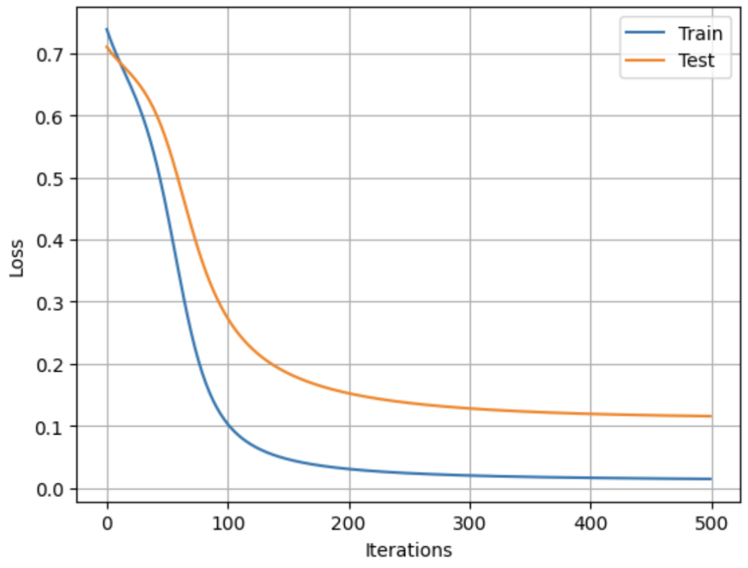


Figure 7: Train vs Test Loss during Training

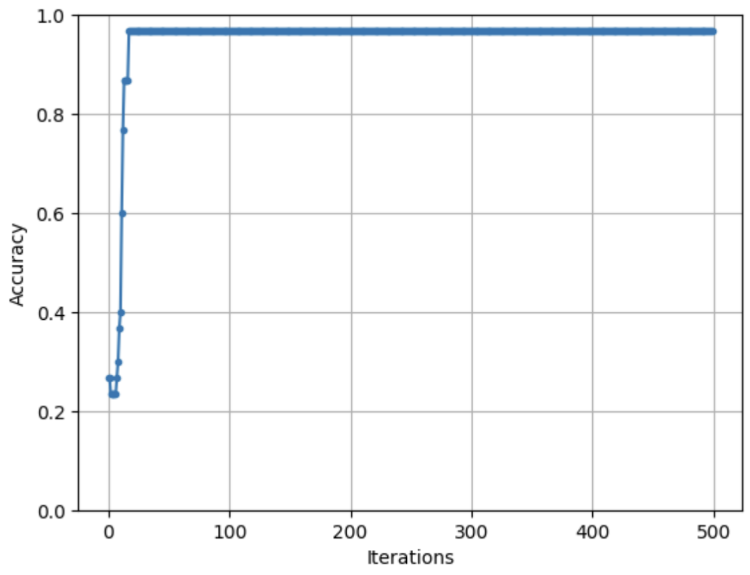


Figure 8: Test accuracy, Peaked at 0.967

9 Resources

- Graph Neural Networks - a perspective from the ground up
  Alex Foo
  https://www.youtube.com/watch?v=GXhBEj1ZtE8
- Graph Convolutional Networks (GCN): From CNN point of view
  Soroush Mehraban
  https://www.youtube.com/watch?v=eLcGehfjvgs
- Graph Convolutional Networks (GCNs) made simple
  WelcomeAIOverlords
  https://www.youtube.com/watch?v=2KRAOZIULzw
- Intro to Graphs and Label Propagation Algorithm in Machine Learning
  WelcomeAIOverlords
  https://www.youtube.com/watch?v=OI0Jo-5d190
- Graph Convolutional Networks using only NumPy
  WelcomeAIOverlords
  https://www.youtube.com/watch?v=8qTnNXdkF1Q
- And many more.....