AIML425 ASSIGNMENT 3

Benjamin McEvoy

300579954

1. INTRODUCTION

This document revises the concepts of Graph Neural Networks (GNNs), Graph Convolutional Networks (GCNs) repository or this google drive link: https://colab.research.google.com/drive/1ihUE6a0yU2caERkOEL-qt3eRSt3w_2HQ?usp=sharing

2. THEORY

This section outlines the background knowledge and topic, required for this paper which includes the descriptions and working of GNNs, GCN, TopKPooling and adjacency matrices.

2.1. Graph Neural Networks

A Graph Neural Network (GNN), is an optimisable transformation on all graph attributes (nodes, edges) that preserves the graph symmetries. Unlike traditional neural networks, GNNs operate on nodes and edges denoted $G = V, E$ to capture complex relationships, with nodes having a feature of $h_v^{(l)}$. They work through the processing of message passing, aggregation, and feature updating to make predictions on the graph. They are used in applications such as traffic and transportation networks, social network analysis, chemistry and biology [1].

2.1.1. Graph Convolutional Network

Graph Convolutional Network (GCN), originating from simplifying "proper" graph convolution based on graph Fourier transform, a GCN aims to learn a meaningful representation of nodes by capturing both local and global graph structures. Similar to Convolutional Neural Networks, GCNs use convolutional layers to apply filters over the graph structure, the filers take into account the local neighbourhood structure and update the node features accordingly. The original GCN denoted as

$$h_v^t = \sigma\left(W_t \sum_{u \in N_v \cup \{v\}} \frac{1}{\sqrt{|N_v||N_u|}} h_{t-1}^u\right)$$

W^t is shared across all nodes, and normalisation is required as the number of the neighbours varies but W^t doesn't. Neighbours with many neighbours around it will have a lower weight than compared to those without. It is well suited to the problem provided in the assignment (seen in Section 3. Experiments), for the simplicity it provides but the ability to aggregate local neighbourhoods.

2.1.2. TopKPooling

TopKPooling is the equivalent of k-max pooling (a generalisation of max-pooling), where each feature map is reduced to size k by picking units with the highest values. In GNNs the k-highest value can come from different nodes for each feature map, so TopKPooling projects all nodes to 1D and then selects the top K from that, keeping the most relevant nodes for the problem and discards the remaining [2].

An algorithm depiction of TopKPooling can be shown as:

H_t is a matrix with vertex embeddings in its rows at iteration t, A_t is the adjacency matrix at iteration t. p with $\|p\| = 1$ be a learnable projection vector. i be a vector of indices to the largest k elements of y [3].

1. $y = H_{t-1}p$
2. Find i
3. $\tilde{y} = \text{sigmoid}(y(i))$
4. $\tilde{H} = H(i, :)$ # keep top-K nodes
5. $H_t = \tilde{H} \odot (\tilde{y}\mathbf{1}^T)$ # gate deals with y being discrete
6. $A_t = A_{t-1}(i, i)$ # keep top-K nodes

2.2. Adjacency Matrix

An Adjacency Matrix (or connection matrix) is a matrix with rows and graphs labelled by graph vertices, in standard formulation, a NxN matrix, denoted by A. Where if, vertex i and vertex j are linked then $A_{ij} = A_{ji} = 1$, else $A_{ij} = A_{ji} = 0$.

3. EXPERIMENTS

This section outlines the setup, configuration, architecture and discussion of the results; providing insight into the utilisation of the GNN/GCN on the problem: "Tiger Walk".

256 tiger sensors are uniformly distributed, in a circular radius of 1000m. The tiger is dropped in the middle of the area and wanders s meter in a random direction till it wanders out of the area. Sensors are triggered within d meter if the tiger has passed it once. We want to know which sensors the tiger has visited, and the sensors giving a false positive with probability q (no false negatives).

3.1. Setup

3.1.1. Sensor Generation

The sensors are generated using polar coordinates (radius r and angle θ) and then converted to Cartesian coordinates x, y using:

$$x = r \cdot \cos(\theta)$$

$$y = r \cdot \sin(\theta)$$

θ is sampled from a uniform distribution between 0 and 2π, ensuring that the sensors can be placed at any angle around the circle.

r is sampled from a uniform distribution between 0 and 1, then scaled by the square root and multiplied by the given radius. The square root ensures a uniform distribution of points within the circle. Then the x and y coordinates are stacked together and transposed to form a 2D array where each row represents the (x, y) coordinates of a sensor.

3.1.2. Sensor Adjacency Matrix

The adjacency matrix is found by creating a square matrix matrix of size nxn, where n is the number of sensors. Initialising all elements to 0, storing the adjacency information.

calculating the distance by iterating over each pair of sensors (i, j) where $i < j$ to avoid redundant calculations and self-loops, computing the Euclidean distance between sensor i and sensor j to compare the distance is less than or equal to u, set matrix[i, j] and matrix[j, i] to 1, indicating that the sensors are connected, however, if the distance is greater than u, leave matrix[i, j] and matrix[j, i] as 0, indicating no connection.

3.1.3. Tiger Walk Simulation

To simulate the tiger walk, the tiger moves in random directions influenced by a direction bias, with each step s of the given step size. The function tracks the tiger's position, checking if it is detected by any sensors. The simulation continues until the tiger reaches the boundary of the circle defined by radius.

3.1.4. Model Architecture

A GCN model structure is used, with three graph convolutional layers (GCNConv). The first layer transforms the input node features to 32-dimensional features, the second layer reduces these to 16 dimensions, and the final layer outputs the number of classes for the binary classification. In the forward method, each layer's output is passed through a ReLU activation function, except for the final layer, which uses a sigmoid activation function to produce the final class probabilities.

3.2. Results

In observation of Appendix Figure 1., the predicted graph indicates a close resemblance to the actual path of the triggered sensors. The observed d, q, and u values: 100, 300 and 0.1 respectively seem to be the best configuration of the ranges and probability and perform better in this set than compared to lower detection ranges.

Through observations of certain $d(50)$, $q(0.1)$ and $u(200)$ values in Appendix Figure 3., there was a 67% prediction accuracy for the sensor with 256 tiger sensors; but using the same model that is trained and inferring a 512 tiger sensor set, the prediction accuracy increased to 85%. Showing that it has improved over the new 512 sensor realisation, and has improved significantly.

Based on the graphs provided and those within the codebase, the experiments in predictions tend to be heavily weighted to continue within a certain direction, and so the model isn't utilised to its best ability. The accuracy provided in the heat map (Appendix Figure 2.) is greatly affected by the communication range and provides insight into the large weight of the adjacency matrix over the probability and detection ranges.

4. TOOLS

This assignment used the tools of Numpy [4], Matplotlib [5], Seaborn [6], SKLearn [7] and Pytorch [8] to carry out experimentation and construction of the graph neural network.

Google Colab [9] is where the code base is hosted and is provided with any available machine during runtime allocation.

Besides the utilisation of these tools [10], I declare that the code and paper provided is my work.

5. CONCLUSION

In conclusion, the results show that the model created can predict the sensors at a reasonable level. Tweaking the weights, algorithm, and addition of normalisation may result in more favourable and consistent accuracies across the different ranges provided in the appendix figures. Despite room for improvement of the code base, GNNs continue to provide popular applications in various sectors, demonstrating capability for complex tasks such as node classification.

6. REFERENCES

[1] José Suárez-Varela, Paul Almasan, Miquel Ferriol-Galmés, Krzysztof Rusek, Fabien Geyer, Xiangle Cheng, Xiang Shi, Shihan Xiao, Franco Scarselli, Albert Cabellos-Aparicio, and Pere Barlet-Ros, "Graph neural networks for communication networks: Context, use cases and opportunities," 2022.

[2] Harsha Kokel, "Understanding node attention in graph neural networks," https://harshakokel.com/posts/understanding-node-attention/, 2020, Accessed: 2024-09-13.

[3] Hongyang Gao and Shuiwang Ji, "Graph u-nets," *arXiv preprint arXiv:1905.05178*, 2019.

[4] Charles R Harris, K Jarrod Millman, Stéfan J van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J Smith, et al., "Array programming with numpy," *Nature*, vol. 585, no. 7825, pp. 357–362, 2020.

[5] John D Hunter, "Matplotlib: A 2d graphics environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.

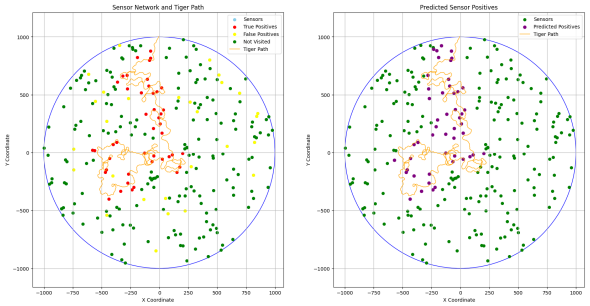
[6] Michael L. Waskom, "Seaborn: statistical data visualization," https://seaborn.pydata.org, 2021, Accessed: 2024-09-13.

[7] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[8] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al., "Pytorch: An imperative style, high-performance deep learning library," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

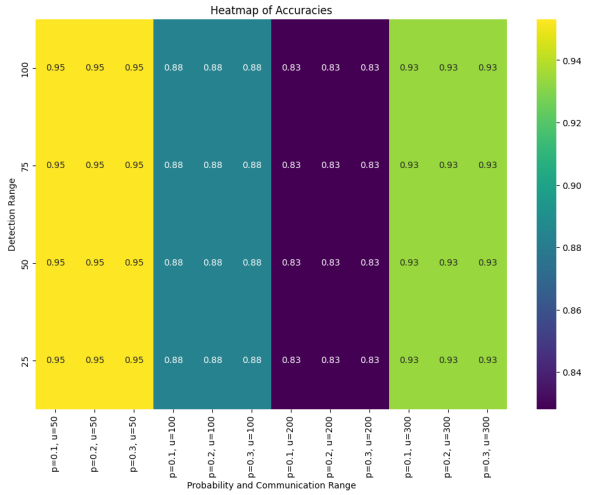
[9] Google, "Google colaboratory," Online, 2023, Available at: https://colab.research.google.com.

[10] Inc. Grammarly, "Grammarly," https://www.grammarly.com, 2024, Accessed: 2024-08-12.



(a) Detected Sensors and Predicted Sensors

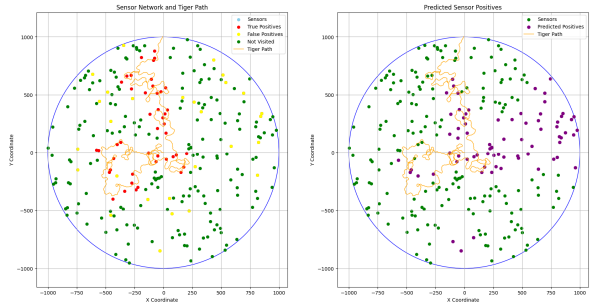
Fig. 1: Sensor Prediction with: u=300, d=100 and p=0.1



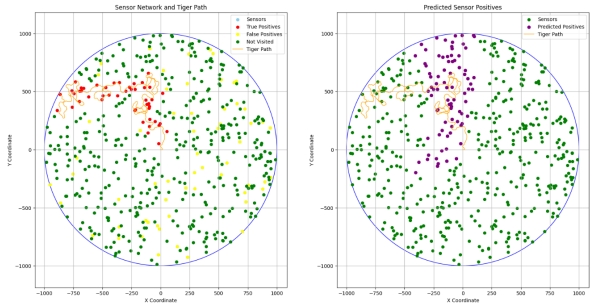
(a) Comparisons between different d u and p ranges

Fig. 2: Heatmap of Accuracies

Appendix



(a) 256 Tiger Sensors



(b) 512 Tiger Sensors

Fig. 3: Detected Sensors and Predicted Sensors - Tensor Comparison