DETECTING TIGER MOVEMENT USING DISTRIBUTED SENSORS AND GRAPH NEURAL NETWORKS

Swati Swati

Victoria University of Wellington, NZ

1. INTRODUCTION

Graph Neural Networks (GNNs) are a class of neural networks designed to work directly with graph-structured data. Unlike traditional neural networks that operate on fixed-size input vectors, GNNs can handle data represented as graphs, which consist of nodes (vertices) and edges (connections between nodes). This makes GNNs particularly powerful for tasks where relationships and interactions between entities are crucial [2].

In this study, we explore the effectiveness of a GNN in improving the detection accuracy of tiger movements within a circular area monitored by uniformly distributed sensors. These sensors, which are prone to noise (false positives), are triggered when a tiger passes within a certain distance. Our goal is to determine which sensors the tiger visited and to enhance the detection accuracy using a GNN.

2. THEORY

In this section, we focus on the theoretical background relevant to the implemented system.

2.1. Graph Attention Neural Networks (GATs)

Graph Attention Networks (GATs) are a type of Graph Neural Network that incorporate attention mechanisms to weigh the importance of neighboring nodes when aggregating information. Unlike traditional GNNs that treat all neighbors equally, GATs assign different attention coefficients to each neighbor, allowing the model to focus on the most relevant nodes [3]. The attention mechanism is typically implemented using a self-attention layer, where attention coefficients α_{ij} between node i and its neighbor j are computed as:

$$\alpha_{ij} = \frac{\exp\left(\text{LeakyReLU}\left(\mathbf{a}^T[\mathbf{W}\mathbf{h}_i \parallel \mathbf{W}\mathbf{h}_j]\right)\right)}{\sum_{k\in\mathcal{N}(i)} \exp\left(\text{LeakyReLU}\left(\mathbf{a}^T[\mathbf{W}\mathbf{h}_i \parallel \mathbf{W}\mathbf{h}_k]\right)\right)} \quad (1)$$

where $\mathbf{W}$ is a learnable weight matrix, $\mathbf{h}_i$ and $\mathbf{h}_j$ are the input features of nodes i and j, $\mathbf{a}$ is the learnable attention vector, $\parallel$ denotes concatenation, and LeakyReLU is a non-linear activation function.

2.2. Graph Adjacency Matrix

The adjacency matrix is a concept in graph theory used to represent the connectivity between nodes in a graph. For a weighted graph with N nodes, the adjacency matrix A is an $N \times N$ matrix where each element A_{ij} represents the weight of the edge between nodes i and j [4]. For a graph with communication constraints, such as sensors communicating over a distance u, the adjacency matrix can be computed as:

$$A_{ij} = \begin{cases} 1 & \text{if distance}(i,j) \leq u \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

The adjacency matrix is crucial for various graph algorithms and applications, including graph neural networks, where it is used to define the relationships and connectivity patterns between nodes.

2.3. Loss Function

The Loss function used here is Binary Cross-Entropy Loss (BCELoss) which is commonly used in binary classification tasks. In the context of GATs, BCELoss measures the discrepancy between the predicted probabilities and the actual binary labels (visited or not visited) for each sensor. Mathematically, the BCELoss for a single prediction is defined as:

$$\text{BCELoss}(p, y) = -\left(y \log(p) + (1-y)\log(1-p)\right) \quad (3)$$

where p is the predicted probability that a sensor was visited, and y is the actual binary label (1 if visited, 0 if not). This loss function penalizes incorrect predictions more heavily, encouraging the model to produce probabilities close to the true labels.

3. EXPERIMENTS

In this section, we investigate the performance of a Graph Neural Network (GNN) designed to predict sensor activations based on a tiger's random walk in a circular area, despite presence the of noise (false positive). We set up an experimental framework to evaluate the effectiveness of the GNN in detecting sensor activations under varying conditions.

3.1. Experimental Setup

Initially, 256 sensors were uniformly distributed within a circular area with a radius of 1000 meters. The objective was to track multiple walks, 5 for this experiment, of the tiger which was dropped at the center of this area as it wandered randomly until it exited. Only the nearest sensor and all the sensors within the communication range *'u'* of that sensor were activated if the tiger comes within *'d'* meters of them at least once. But each sensor has a probability *'q'* of reporting a false positive, and hence this noise was added in the *Sensor Visited Matrix*.

The GAT model was employed with an input layer consisting of 2 neurons, corresponding to feature dimension of sensor locations, a hidden layer with 20 neurons and an output layer with a single neuron to provide the probability of a sensor being visited by the tiger. Four attention heads were utilized in the Graph Attention Convolution (GATConv) layer to capture the significance of each neighbor's information. The output was then passed through a ReLU activation function to introduce non-linearity. Finally, a Sigmoid activation function was applied to the output layer to ensure that the predictions were constrained between 0 and 1, representing the probability of sensor activation [1].

3.2. Training Process

Initially, the GAT model was trained on the complete graph data of the 'Sensor Visited Matrix' for 256 sensors, including noise, over 1000 epochs using the Adam optimizer and Binary Cross-Entropy Loss (BCELoss), with values for *d* as 50, *q* as 0.05, and *u* as 100. The predicted sensor matrix was then compared with the actual activated sensors, excluding false positives.

The graph data was then, subsequently split into training and validation sets with an 80-20 percent ratio. Hyperparameter tuning was performed to enhance the model's generalization and robustness. The model was retrained with the optimal hyperparameter values, as detailed in Table 1, and the Training vs Validation Loss Curve was plotted, as illustrated in Fig. 2.

The actual vs. predicted sensor visited graph is depicted in Fig. 1. It shows some introduction of False Negatives during prediction, indicating that sensors which were correctly activated initially were falsely predicted as not activated.

Performance evaluation was conducted for various *q*, *u*, and *d* values, as shown in Table 2. Accuracy vs. *d* was plotted for these *q* and *u* values, as demonstrated in Fig. 3.

The evaluation process was then extended to 512 uniformly distributed sensors using the same trained model i.e. the model's performance was assessed using the same *q*, *u*, and *d* values as for the 256 sensors, with Accuracy vs. *d* plotted for these parameters, as illustrated in Fig. 4. Finally, results were compared between the 256 and 512 sensor configurations.

3.3. Results

Initially, the GAT model achieved 92% accuracy on the complete graph data, suggesting overfitting. After an 80-20 training-validation split and hyperparameter tuning, accuracy decreased to around 80%, indicating better generalization.

For both the 256-sensor and 512-sensor configurations, accuracy increased with d. Lower q values (e.g., $q = 0.01$) led to more significant accuracy improvements with increasing d, while higher q values (e.g., $q = 0.1$) showed slower gains. Increasing u slightly improved performance, especially for lower d values, but the effect diminished at higher d.

In the 256-sensor setup, $q = 0.01, u = 150$ achieved over 0.7 accuracy at $d \approx 80$, whereas $q = 0.1, u = 50$ showed the lowest accuracy. Similarly, in the 512-sensor setup, $q = 0.01, u = 150$ reached over 0.77 accuracy at $d \approx 80$, while $q = 0.1, u = 50$ had the poorest performance.

Generally, the 512-sensor configuration outperformed the 256-sensor setup across various parameters, though there were some exceptions.

4. CONCLUSION

From the performed experiments, several key conclusions were drawn:

Increasing the detection radius d improves sensor coverage and reduces noise, enhancing overall model performance. Lower q values (e.g., $q = 0.01$) result in fewer false positives, allowing the model to learn more effectively and achieve greater accuracy with larger d. In contrast, higher q values (e.g., $q = 0.1$) increase false positives, leading to slower accuracy growth as d rises.

A greater communication range u allows sensors to connect over longer distances, providing richer data and improving performance. However, beyond a certain d, additional increases offer diminishing returns due to data saturation.

Finally, increasing the number of sensors generally enhances model accuracy by providing more data points.

5. STATEMENT OF TOOLS USED

The tools used in this study include PyTorch for neural network implementation, torch.geometric for GNN implementation, Plotly for graph visualization, and Google Colab for executing the code. The code was written by me, and I verify that it is my original work. The code can be accessed at the following link: *Final-Code*.

6. REFERENCES

[1] Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric, 2019.

[2] Jianchao Lu et all Jiaqi Ge, Gaochao Xu. GraphSensor: A Graph Attention Network for Time-Series Sensor, 2024.

[3] Prof Bastiaan Kleijn. Graph attention network. Victoria University Lecture in L8_GNN_short.pdf, 2024.

[4] Prof Bastiaan Kleijn. Graph neural network: the theory. Victoria University Lecture in L8_GNNs.pdf, 2024.

7. APPENDIX

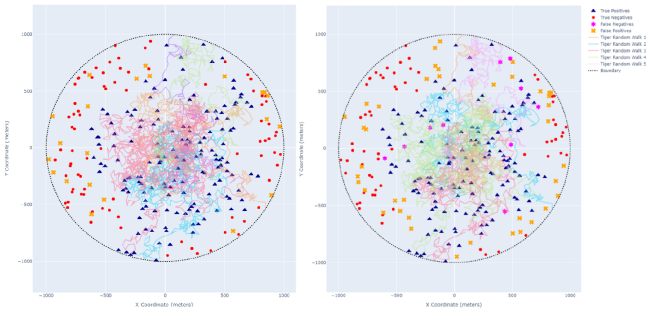

Fig. 1: Comparison of Actual Sensor Activation Graph containing Noise Vs GAT Model Predicted Sensor Activation Graph for 256 Sensor Configuration .

Table 1: GAT Model parameters optimization.

Parameter	Values	Best Value
heads	1, 2, 4, 8	4
hidden_dim	8, 16, 18, 20	20
learning_rate	0.0, 0.001, 0.01	0.001
epochs	100, 200, 500, 1000	500
patience	10, 20, 30	20

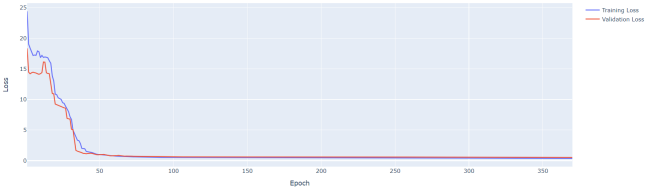

Fig. 2: Training vs Validation loss curve of GAT model.

Table 2: Performance Evaluation on q, u, d.

Parameter	Values
q	0.01, 0.05, 0.1
u	50, 100, 150
d	10, 20, 30, 40, 50, 60, 70, 80, 90, 100

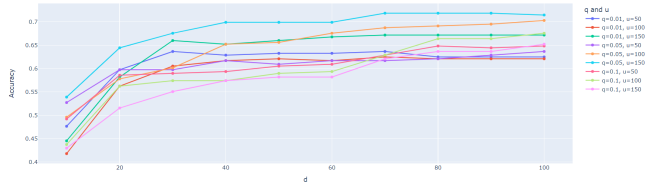

Fig. 3: Accuracy vd d for various q and u values on 256 Sensor Configuration.

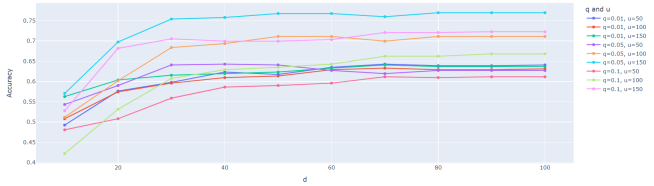

Fig. 4: Accuracy vs d for various q and u values on 512 Sensor Configuration.