GRAPH NEURAL NETWORKS APPLIED TO A MOTION DETECTION PROBLEM

Maria DaRocha

VUW, AIML425: Neural Networks and Deep Learning

1. INTRODUCTION

The following is my own work and supported by code available on GitHub and Google Collab.

The following paper is an investigation into designing a Graph Neural Network (GNN) that solves the (theoretical) problem of a tiger on a random walk in a circular area with a radius of 1,000 meters. Within this area, there are 256 uniformly distributed sensors that are triggered if a tiger passes within a set distance (in meters) at least once. Further, the sensors can yield false positives, but not false negatives. The goal is to determine which sensors the tiger visited before wandering out of the area [1].

2. THEORY

This paper defers to appendix references for the theoretical relevance of cross-entropy [2], loss minimisation [3], objective functions [4], and Convolutional Neural Networks (CNNs) [5].

2.1. GNNs

Graph Neural Networks (GNNs) can be understood as generalised Convolutional Neural Networks (CNNs) in which the network accepts, by design, irregular representations of data as input (i.e., a graph with a variable number of nodes and edges per node); a format that is intractable for regular convolutions along grid-like channels using a fixed kernel.[1]

A GNN receives, as input, a graph consisting of a set of vertices, edges, and (perhaps) a global property ($G = \{V, E, U\}$), that can itself possess features (embeddings, $h_v^{(l)}$) in the form of scalars (vectors) containing information about the global-context, nodes, and/or edges, such as weights or property information. GNNs may also use the graph's connectivity in learning, which is either represented as an adjacency matrix (i.e., for N vertices, the adjacency matrix is an $N \times N$ matrix denoted 'A'), or as adjacency lists (i.e., for each edge E (u,v), a jagged array of lists (tuple) stores, for each node, a list of that node's neighboring vertices; this is the preferred representation for large, sparse graphs).

To formulate predictions, the GNN performs iterative (layer-wise) computations resulting in, for each layer, updated embeddings ($h_v^{(l)}$) at either the graph-, node-, or edge-level. The general formula for a GNN with only node features can be described for node *v* at layer (or time) *t* by the update rule presented in *Equation 1* (*Appendix*) [1].

More sophisticated predictions can be made by using pooling within the GNN layer to make the learned embeddings aware of the graph's connectivity. This can be achieved by passing messages between parts of the graph, which happens in three steps reminiscent of a standard convolution (*Equation 2*, *Appendix*) [1], [6].

2.1. GATs

Graph Attention Neural Networks (GATs) are presently the most common and powerful formulations of GNN-architecture.[2] Further, knowledge of autoencoders [7], universal transformers [8], and attention mechanisms [9], [10] is helpful in making sense of a GAT's logic and structure. For a comprehensive background on these topics, this paper defers to the references cited.

The GAT architecture introduces an attention mechanism that assigns levels of importance to inputs from neighbouring nodes. The relative importance is then encoded as scalar attention coefficients, used to modulate the contribution of each neighbour at the time of feature aggregation. Attention coefficients are computed as a function of the features of the neighbours and the node undergoing updates (including edge features, if they exist). Mathematically, this is expressed as the set of functions presented in *Equation 3* (*Appendix*).

In GATs, the latent space is implicitly controlled through the attention mechanism, which is deterministic. That is, GATs do not impose a probabilistic constraint on the latent space in the same way that VAEs do (e.g., tending towards Gaussian to maximally pass information between layers).

[1] *Channels*: separated, lower-resolution aspects of an image; *Convolutions*: integral expressing the amount of overlap of one function, g, as it is shifted over another function, f. [5]

[2] GAT performs better than GCN (and GraphSAGE) and is more efficient at message-passing. [1]

Rather, the latent space in a GAT is shaped by the local structure of the graph and its attention coefficients. For models that are well-trained to perform binary (or multiclass) classification, we would expect a 2D representation of the latent space to show distinct clusters for nodes with similar labels to suggest meaningful encodings. An example of expected graphical behaviour on MNIST data can be seen in *Post-hoc counterfactual generation with supervised autoencoder* (*Figure 1, Appendix*) [11].

3. EXPERIMENTS

3.1. Experiment Set-Up

The problem described in the *Introduction* (above) was set up using Python in Jupyter Lab. The following describes the environment prepared for the GAT:

- A circular sensor area was sampled from uniform *r (weighted)* and θ, then represented as a set of coordinates (see *Appendix, Equation 4*).
- For the input graph's adjacency matrix (connectivity), two sensors were deemed capable of communicating (i.e., connected) if their pairwise distance was less than distance $\boldsymbol{u}$.
- The tiger's (non-neural) random walk was a stochastic path of increment (step size) $\boldsymbol{s}$, and the animal was said to have "left" the space (i.e., reached termination) once its total distance from origin (0, 0) exceeded 1,000 units (meters).
- During this period, if the tiger moved within a distance $\boldsymbol{d}$ (meters) of a sensor on the grid, that sensor's value was set to 1 to indicate it had been triggered.
- Noise injection was used to simulate a false positive rate $\boldsymbol{q}$ amongst the sensors.
- Each variable $\boldsymbol{u}$, $\boldsymbol{d}$, and $\boldsymbol{q}$ were arbitrarily initialised and later learned (optimized) by the GAT.
- An example visualisation of the set-up can be seen in *Figure 2* (*Appendix*).

3.2. GNN Architecture (GAT)

The problem outlines a binary classification task that aims to predict the probability $p(c_i = 1)$ that the tiger did $(c_i = 1)$ or did not $(c_i = 0)$ visit a sensor. The objective is therefore to minimise the Binary Cross Entropy (BCE) loss, defined in *Equation 5* (*Appendix*).

The node features consisted of sensor positions and the degree of each sensor (i.e., number of connections). The single edge feature was a scalar value (edge weight) representing the distance between two connected nodes. Both feature sets were passed to the GAT to guide learning the attention mechanism.

The GAT was instantiated with two Graph Attention Convolutional (GATConv) layers that handle the node embeddings (for which there were two input channels, sensor coordinates and degrees) and the edge embeddings, (for which there was one input channel, connection distance).

The third and final layer in the network was a fully connected layer for adjusting the probabilities based on $\boldsymbol{q}$, such that the system could better learn a correction mechanism to reduce the overall influence of false positives. ReLU activation was applied to the output of each GATConv layer, and the final fully connected layer used a sigmoid activation to output probabilities.

In this architecture, the final model only used one head for learning (self-attention).[3] To following the chain of convolutions in detail, refer to *Figure 3* (*Appendix*).

After the model gathered its predictions, a reasonable threshold was selected to convert the probabilities back into binary outputs.

4. CONCLUSION

The training, validation, and test losses displayed oscillation, but also demonstrated some amount of learning and convergence. The losses were tracked and can be seen in *Figure 4* (*Appendix*). The training loss decreased over time, but these losses were not substantial. The validation loss also fluctuated, but did not adhere to a decreasing trend, which could indicate overfitting. Finally, the test loss was often high and stable, which could indicate poor model generalisation.

The 2D representation of the latent space did not follow the clustering behaviour expected for this type of task (*Figure 5, Appendix*). There are clear clusters of red (triggered) and blue (not triggered) nodes, which would indicate the model has learned to distinguish between the two classes to some extent, but the circular arrangement and relatively even spacing suggests that the learned embeddings are perhaps more spatially uniform than driven by actual class separation.

The final graph for sensor predictions can be see in *Figure 6* (*Appendix*), where predicted sensor triggers have been highlighted in red. As evidenced by comparing *Figure 2* with *Figure 6*, the predictions shown on the output graph are too divergent for the model to be considered suitable for this task. If desired, please see informal discussion (*below*) for a response to the model's unsatisfactory behaviour.

[3] Where more than one attention head was used, outputs of all heads were concatenated along the feature dimension.

5. APPENDIX

5.1. Equations

(1) $\boldsymbol{h_v^t = f(h_v^{t-1}, \{h_u^{t-1} | u \in \mathcal{N}_v\}, t)}$, where…

$\mathcal{N}_v$ represents the neighborhood of v, f is a neural network layer (e.g., a graph convolution or an attention mechanism), and prediction tasks occur at the (A) graph-, (B) node-, or (C) edge-level, where the objective is to [6]:

(A) predict a single property for a whole graph,
(B) predict some property for each node in a graph, and/or
(C) predict the property or presence of edges in a graph.

Equation 1: General Formulation GNN with Node Features. [1], [6]

---

(2) $\boldsymbol{m_{uv} \leftarrow f_{uv}(h_u, h_v, e_{vu})}$
(3) $\boldsymbol{h_v \leftarrow f_v(h_v, \sum_{u \in \mathcal{N}_v} m_{uv})}$
(4) $\boldsymbol{e_{uv} \leftarrow U(e_{vw}, h_u, h_v)}$, where…

f_{uv} and f_v are small neural networks, e_{uv} and U are usually not present, and general-form message passing is non-trivial.

A "plain English" description of message-passing:

Step 1: *For each node in the graph, gather all that node's neighbouring embeddings (or messages).*

Step 2: *Aggregate all messages using an aggregate function (e.g., concatenation or sum).*

Step 3: *All pooled messages are passed through an update function (usually a learned neural network).*

Equation 2: Message-Passing GNNs. [1], [6]

---

(5) $\boldsymbol{h_v \leftarrow \sigma(\sum_{u \in \mathcal{N}_v} a_{vu} W h_u)}$
(6) $\boldsymbol{a_{vu} \leftarrow \frac{exp(a_{vu})}{\sum_{w \in m\mathcal{N}_v} exp(a_{vw})}}$
(7) $\boldsymbol{a_{vu} \leftarrow a(h_v, h_u, e_{uv})}$
(8) $\boldsymbol{e_{uv} \leftarrow U(e_{vw}, h_u, h_v)}$, where…

a is a small neural network, a_{vu} is a scalar that tells us how important a particular neighbour u is, and generalises to a multi-head case, which is almost always used.

Equation 3: Graph Attention Networks (GATs). [1], [6]

---

(9) $\boldsymbol{r = \sqrt{U(0,1)} * 1000}$
(10) $\boldsymbol{\theta = U(0, 2\pi)}$
(11) $\boldsymbol{\left(x = r cos(\theta), y = r sin(\theta)\right)}$

Equation 4, Sampling Sensor Locations within the Circular Area and Converting to Cartesian Coordinates.

(12) $\boldsymbol{\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N}\left[c_i log\left(p(c_i)\right) + (1 - c_i) log(1 - pc_i)\right]}$,
where…

c_i is the ground truth (did/did not visit), $\mathrm{p}(c_i)$ is the predicted probability that sensor i was visited, and N is the total number of sensors.

Equation 5, Binary Cross-Entropy (BCE) Loss.

5.2. Figures

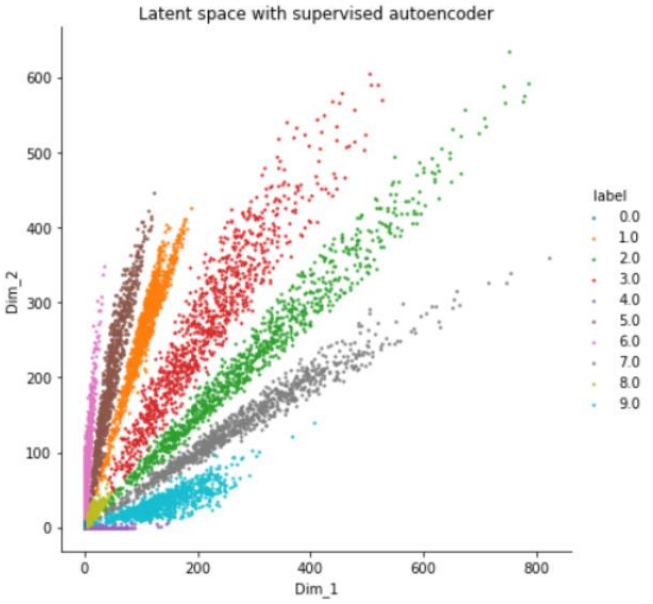

Figure 1, Latent Space with Supervised Autoencoder [11].

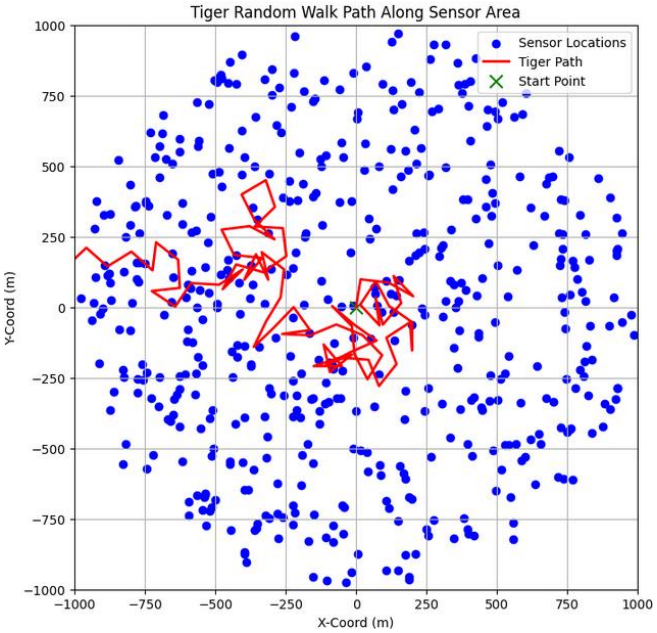

Figure 2, Simulated Tiger Random Walk Amongst 512 Uniformly Distributed Sensors.

Layer 1: GAT Layer #1 (GATConv)

Initial Input:

- Node features $[N, 3]$, where N = # of nodes (512) and 3 = input feature size
- Edge features $[e, 1]$, where e = # of edges and 1 = input feature size
- Edge index $[2, e]$, for node connections (COO format, PyTorch Geometric feature)

Occurrence:

1. Attention mechanism computes attention coefficients b/w neighbours based on their features and edge attributes
2. For each node, information from the node's neighbours is aggregated using the learned coefficients
3. Each $[N, 3] \rightarrow [N, 8]$, transformed into a new feature vector of size 8, set arbitrarily but kept intentionally small, given the computational cost and difficulties with kernel crashes
4. Applies ReLU activation

Output:

- Output node feature size = $[N, 8]$,
- Edge features remain $[e, 1]$, and
- Edge index is still $[2, e]$

Layer 2: GAT Layer #2 (GATConv)

Input:

- Output of ***Layer 1***

Occurrence:

1. Same as above, but for $[N, 8]$
2. Same as above, but for $[N, 8]$
3. $[N, 8] \rightarrow [N, 8]$, transformed into a new feature vector of size 8, no concatenation (self-attention)
4. Applies ReLU activation

Output:

- Output node feature size = $[N, 8]$,
- Edge features remain $[e, 1]$, and
- Edge index is still $[2, e]$

Layer 3: Fully Connected Layer

Input:

- Output of ***Layer 2***

Occurrence:

1. $[N, 8] \rightarrow [N, 1]$, reduces the 8-dim feature vector for each node to a single output
2. Applies sigmoid activation

Final Output:

- $[N, 1]$, a scalar value (probability) for each node (i.e., the probability that the sensor is triggered

Figure 3, GAT Convolutions.

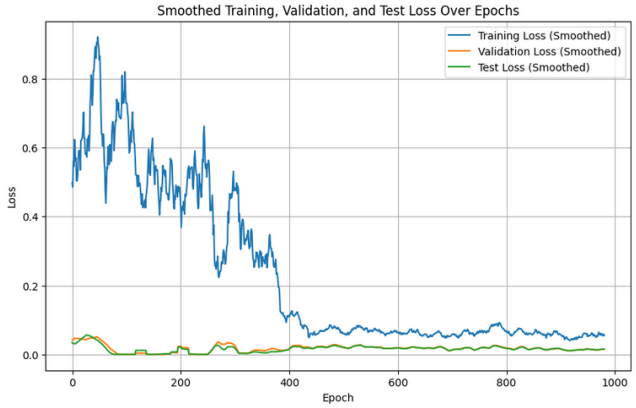

Figure 4, Model Losses Over Epochs (MA-Smoothed).

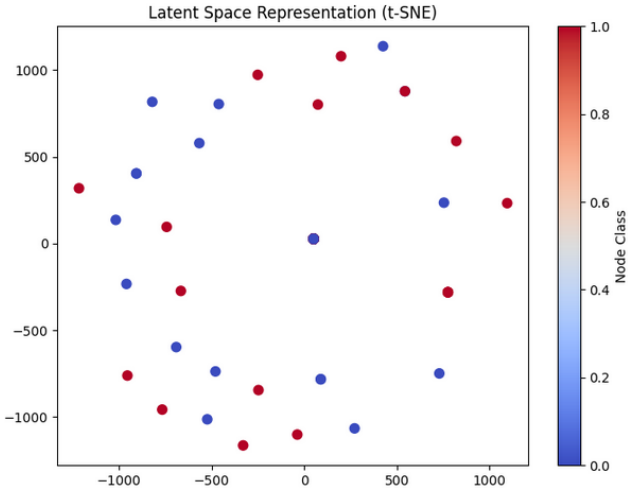

Figure 5, 2D Latent Space Representation.

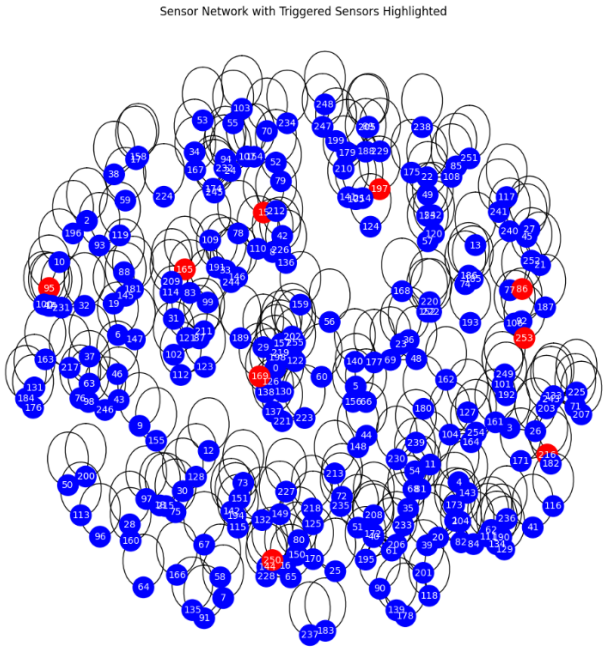

Figure 6, Sensor Network with Highlighted Triggers.

6. REFERENCES

[1] B. Kleijn, "Graph Neural Networks: AIML425 [Lecture 6 Notes]," *Master of Artificial Intelligence, Victoria University of Wellington*, Sep. 2024, Accessed: Sep. 07, 2024. [Online]. Available: https://ecs.wgtn.ac.nz/foswiki/pub/Courses/AIML425_2024T2/LectureSchedule/GNNs.pdf

[2] B. Kleijn, "Brief Review Probability Theory and Basic Information Theoric Quantities as Used in Deep Learning: AIML425 [Lecture 1 Notes]," *Master of Artificial Intelligence, Victoria University of Wellington*, pp. 16–20, Jul. 2024, Accessed: Jul. 18, 2024. [Online]. Available: https://ecs.wgtn.ac.nz/foswiki/pub/Courses/AIML425_2024T2/LectureSchedule/probability.pdf

[3] M. DaRocha, "Demonstrating the Basic Principles of Generative Models for (FCNN) Gaussian and Uniform Mappings," *VUW, AIML425: Neural Networks and Deep Learning*, Aug. 2024, Accessed: Sep. 10, 2024. [Online]. Available: https://github.com/Marianette/A2-AIML425

[4] B. Kleijn, "Objective Functions and Regularisation: AIML425 [Lecture 4 Notes]," *Master of Artificial Intelligence, Victoria University of Wellington*, pp. 4–11, Aug. 2024, Accessed: Aug. 14, 2024. [Online]. Available: https://ecs.wgtn.ac.nz/foswiki/pub/Courses/AIML425_2024T2/LectureSchedule/lect4ObjF.pdf

[5] B. Kleijn, "Convolutional Neural Networks: AIML425 [Lecture 5 Notes]," *Master of Artificial Intelligence, Victoria University of Wellington*, Aug. 2024, Accessed: Sep. 08, 2024. [Online]. Available: https://ecs.wgtn.ac.nz/foswiki/pub/Courses/AIML425_2024T2/LectureSchedule/lect5CNN.pdf

[6] J.-B. Vincent, A. de G. Matthews, and P. Manurangsi, "A Gentle Introduction to Graph Neural Networks," *Distill*, 2021, doi: 10.23915/distill.00030.

[7] B. Kleijn, "Autoencoders: AIML425 [Lecture 7 Notes]," *Master of Artificial Intelligence, Victoria University of Wellington*, Sep. 2024, Accessed: Sep. 08, 2024. [Online]. Available: https://ecs.wgtn.ac.nz/foswiki/pub/Courses/AIML425_2024T2/LectureSchedule/autoEnc.pdf

[8] M. Dehghani, S. Gouws, O. Vinyals, J. Uszkoreit, and Ł. Kaiser, "Moving Beyond Translation with the Universal Transformer," 2018.

[9] H. Lee, "Transformers: AIML431 [Lecture 14 Notes]," *Department of Computer Science & Information Engineering of National Taiwan University*, 2024, Accessed: Sep. 10, 2024. [Online]. Available: https://ecs.wgtn.ac.nz/foswiki/bin/viewfileauth/Courses/AIML431_2024T2/LectureSchedule/Transformer%20(v5).pdf

[10] L. Li and K. Shapovalenko, "Transformers: AIML431 [Lecture 15 Notes]," *Transformers [Lecture 19], Introduction to Deep Learning, Carnegie Mellon University*, vol. 11, no. 785, Apr. 2024, Accessed: Sep. 10, 2024. [Online]. Available: https://deeplearning.cs.cmu.edu/S24/index.html

[11] V. Guyomard, F. Fessant, T. Bouadi, and T. Guyet, "Post-hoc Counterfactual Generation with Supervised Autoencoder," 2021, pp. 105–114. doi: 10.1007/978-3-030-93736-2_10.

7. INFORMAL DISCUSSION

Further investigation into model correction or improvement is still required.

Several approaches were trialed to rectify the performance of the model, such as effecting the architecture to include multiple attention heads (up to 3) and batch normalisation. Some of these attempts were retained for marking consideration under the final *'Debugging'* section of the .ipynb file.

Unfortunately, most of these attempts still resulted in either similar performance or kernel crashes and, after a considerable amount of time was spent on testing different methods to improve performance or reduce the model's complexity (compute), these issues still could not be identified or resolved within the specified timeframe.