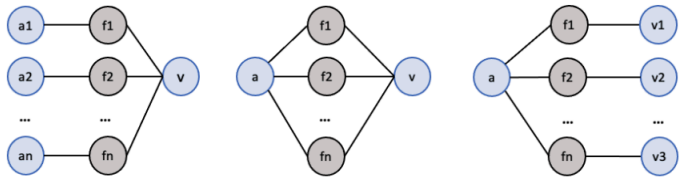
A compendium of thoughts: **Temporal Graph analysis for Anomaly Detection**
https://www.linkedin.com/in/satya-prakash-matukumalli/

Graph Theory is a field of mathematics that has been studied extensively over centuries. If we ask the question, why graphs? Why does it matter in the age of Data Science and Machine learning? I want you to read this paper which points to the fact that just looking at basic stats one would lose the context of its basic structure which was aptly demonstrated by Datasarus dataset. Graphs not only provide structural information but are also suitable for encoding rich information via features like directed/undirected, weighted/unweighted, mapping etc. The fundamental analysis of graphs involves node, link (edge) and graph (structure) predictions. Graph analysis has led to new forms of information retrievals implemented by graph databases. Entities and relationships are stored as nodes and edges in graph databases as tuples which are sometimes mentioned as semantic graphs or Knowledge graphs.

We focus on graph analysis using neural networks. How can we encode the relationship between the nodes along with node features? With the success of convolution networks in image classification many have adopted the same techniques on the adjacency matrix. This approach has not provided good results as graphs are not conformant as image pixels. But a true breakthrough came with massage passing features to connect nodes proposed in the paper "Semi-Supervised Classification with Graph Convolutional Network". Several architectures have been proposed over time to address dense, sparse, and larger graphs using future augmentation, virtual nodes, sampling neighborhood etc. based on compute and memory complexities. With advancements in NLP two architectures Transformers and Diffusion have gained prominence. We will see how these architectures will help in graph analysis later in this article. There are several libraries that implement GNN of which GraphSAGE, Graph Nets, Deep Graph Library and Pytorch Geometric are most popular. Another software worth noting is GraphScope which is a complete ecosystem for persistence, distributed compute, graph traversal and GNN in a flex Lego architecture to customize to your use case. This is also available with commercial support as well.

There are many applications of graph modes, we are interested in Cyber and Fraud. Both these use cases involve link and node analysis. Shown below are the subgraph patterns used in Cyber Security field. GNN 's plays a crucial role in all Cyber security fields including but not limited to, Source code vulnerability, Smart contacts vulnerability, Network intrusion detection, Malwares, Phishing, Deep Fakes and Privacy. For a complete survey of GNN's reader can refer to "Use of Graph Neural networks in adding Defensive Cyber Operations". Readers who are interested in Differential privacy can refer to my previous post.

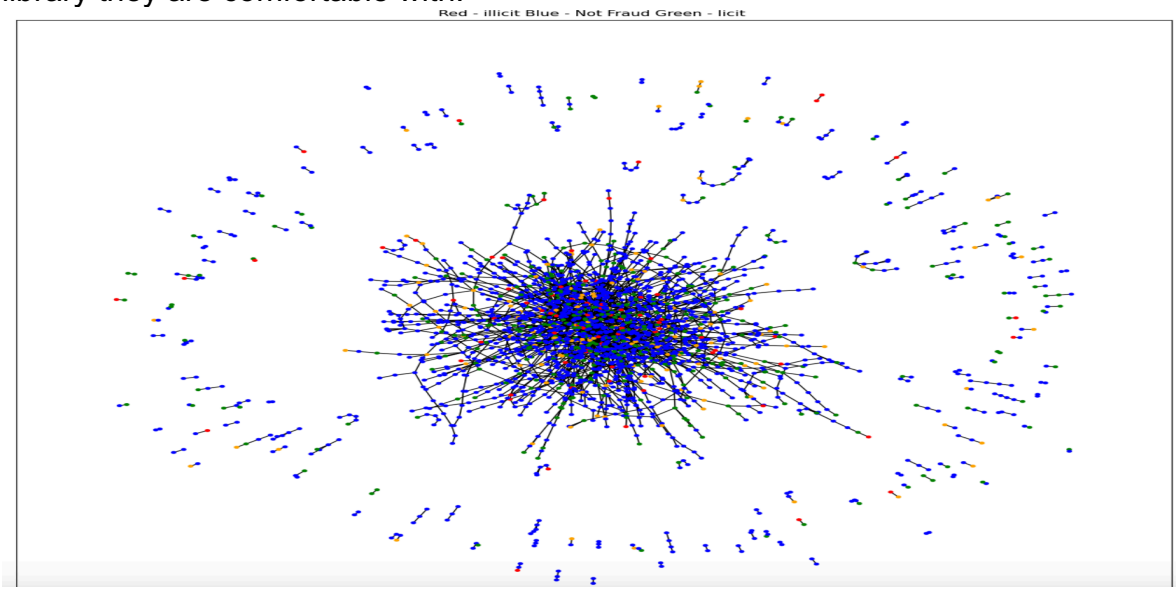


In case of Fraud and anomaly detection GNN models have proven very effective and evolved over the years with advancements in graph algorithms. A chronological survey of the models can be found in this GitHub repo. As mentioned earlier we provide a glimpse into attention and diffusion models for anomaly detection in my TemporalGraphAnamoly repo.

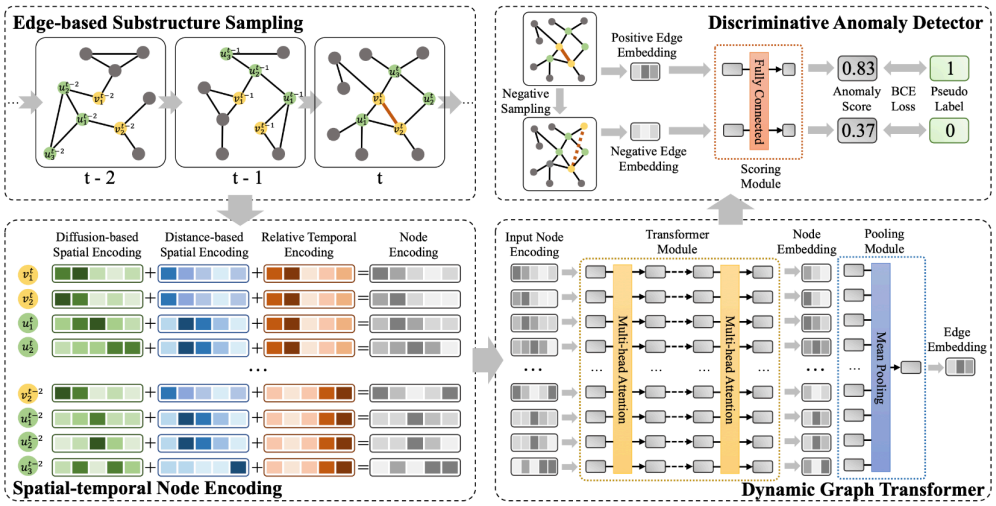
Graph Attention Transformers (GAT) – What is Attention? In graph context it is the relative importance of neighboring nodes. Intuitively nodes that are in Layer 1 have more impact compared to nodes in Layer 2 but does this mean the all nodes in layer 1 are equal or can we rank them. In other words, can we learn the importance of the neighboring nodes w.r.t node in question. GAT provides algorithms to learn the relative importance. This has been very successful. But the procedure to calculate attention is unconditionally. Can we do better than this? What if we compute attention dynamically based on node of interest and have a family of scoring functions and have some kind of boosting mechanism. This led to GATv2 explaining in the paper "How attentive are GAT's". GATs are computationally expensive, so they are better suited for fixed small and medium sized graphs and not suited for internet scale graphs.

To understand the concepts of GAT's we code an example from scratch with a couple of attention heads and ability to pass various aggregations. We use the bitcoin data provided in Kaggle. This is a classification problem. It has 3 classes – illicit, licit and unknown. The classes are skewed heavily like any other fraud dataset. This is a transaction data providing a buyer customer relation along with the nature of transaction like location, method of payment etc. We will merge the transaction data and relationship graph data to come up with node features and edge features. Our goal is to make a node level prediction. The actual GAT model is defined in customGATv2.py which defines message passing mechanism and Multilayer perceptron for classification. Train.py provides a helper function for training. We use the networkx package to draw a network showing transaction classification as per model with high accuracy. Once the reader understands how GAT works then they can directly GATv2 in PyG or another

library they are comfortable with.



In the previous example we consider the network as static. But very rarely networks are static. So how do we account for analysis of dynamic graphs? In the field of cyber security dynamic networks play an important role as mentioned previously. We consider using temporal graph transformers for anomaly detection. We take temporal graphs as an input and create diffusion based spatial embeddings, distance based spatial embeddings, relative temporal encoding and develop node features. These are passed to Dynamic Graph Transformers with multiple attention heads to get edge encodings. These edge encodings will be used to train the anomaly detection using Discriminative Anomaly Detector foredge prediction. For detailed treatment refer to paper "Anomaly detection in dynamic graphs using transformers".



For the actual implementation reader can refer to ADDyGraph.ipynb.

There is very rich literature related to graph neural networks. Readers should consider each architecture depending on the nature of the problem. Are they doing node prediction or edge prediction or subgraph matching? It also depends on the fact size of

the graph, how many hops to consider will have definite impact on the architectural choices. Another factor to consider is can you be able to implement the algorithm in a distributed fashion to address graphs of internet scale.