Progess of the Project

Vincent Pai 2023/7/12

Outline

- **TRAM**
  - What’s TRAM
  - How to use TRAM
  - My automation code
- **Finding the models**
  - My task
  - Graphormer
  - Others may be useful
- **Future Plan**

TRAM

What is TRAM

- **Threat Report ATT&CK MAPPER** (TRAM)
  - TRAM is an open-source platform designed to advance research into automating the **mapping** of **CTI** reports to **MITRE ATT&CK.**
  - TRAM enables researchers to **test and refine** Machine Learning models for identifying ATT&CK techniques in prose-based threat intel reports and allows threat intel analysts to train ML models and validate ML results.
    - There's 4 ML models exist now and all of them are implemented as an **SKLearn** Pipeline

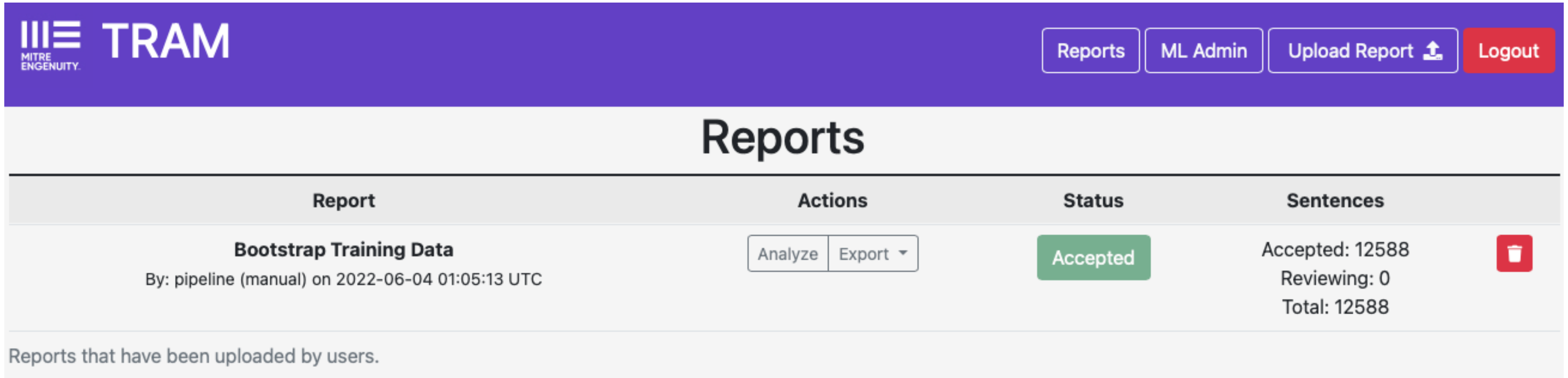
How to Use TRAM

- Need to use the Docker
  - Download the docker-compose.yml for TRAM

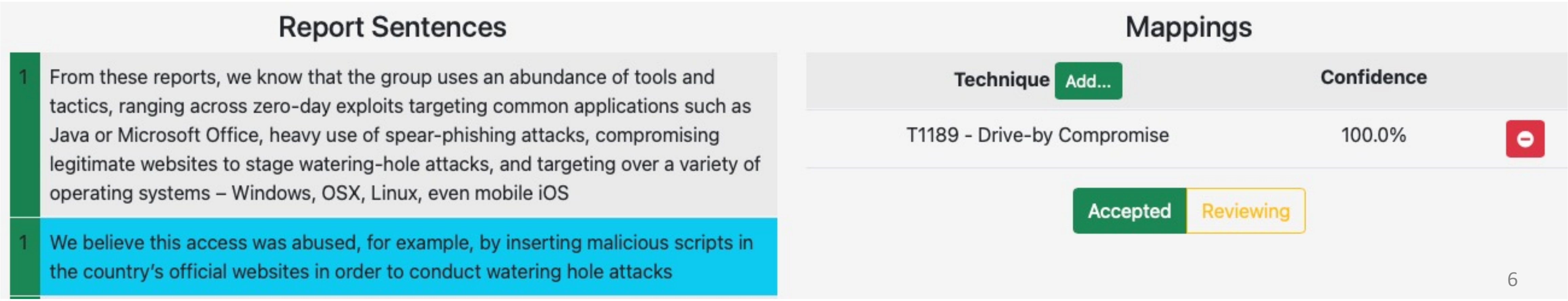
```yaml
version: '3.5'
services:
  tram:
    image: ghcr.io/center-for-threat-informed-defense/tram:latest
    environment:
      - DATA_DIRECTORY=/tram/data
      - ALLOWED_HOSTS=["example_host1", "localhost"]
      - DJANGO_SUPERUSER_USERNAME=djangoSuperuser
      - DJANGO_SUPERUSER_PASSWORD=LEGITPassword1234 # your password here
      - DJANGO_SUPERUSER_EMAIL=test@example.com # your email address here
```

  - Run some command in the same directory with the yml file to download the Docker images
  - Navigate to http://localhost:8000/ and login

How to Use TRAM

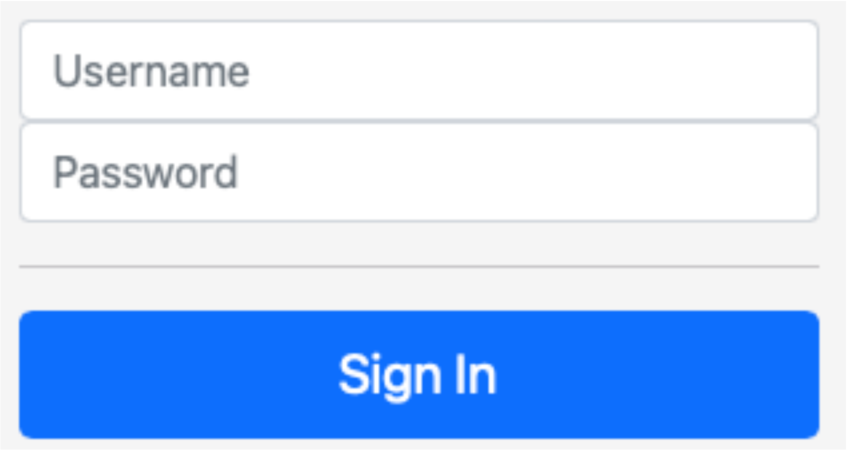


- After clicking the analyze:
  - Show the sentences and the corresponding MITRE ATT&CK technique

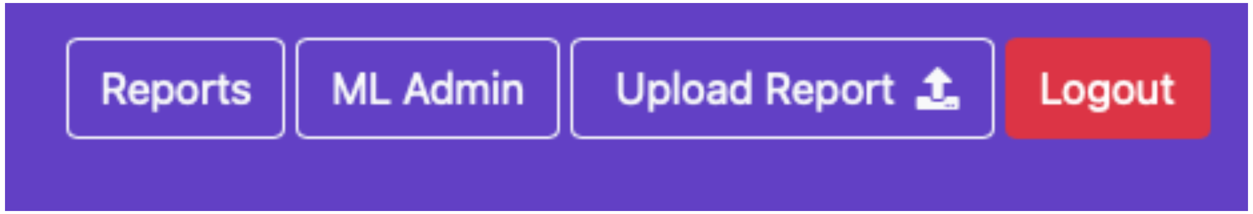


Automation - Tasks

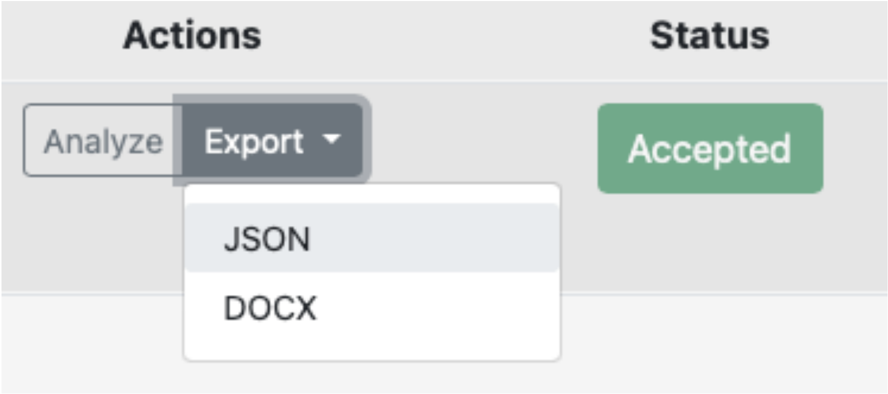
1.



2.



3.



4. Postprocessing

Automation - programming

- Packages we need to import: **Selenium**, os, csv, json

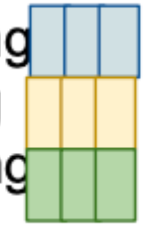
- **Upload.py**: Sign in and upload the files
- **Export.py**: After singing in, press the export and JSON for all the files
- **Postprocess.py**: Turn the all json files in a directory into the labeled csv file(Only the sentenses and the corresponding MITRE ATT&CK techniques.

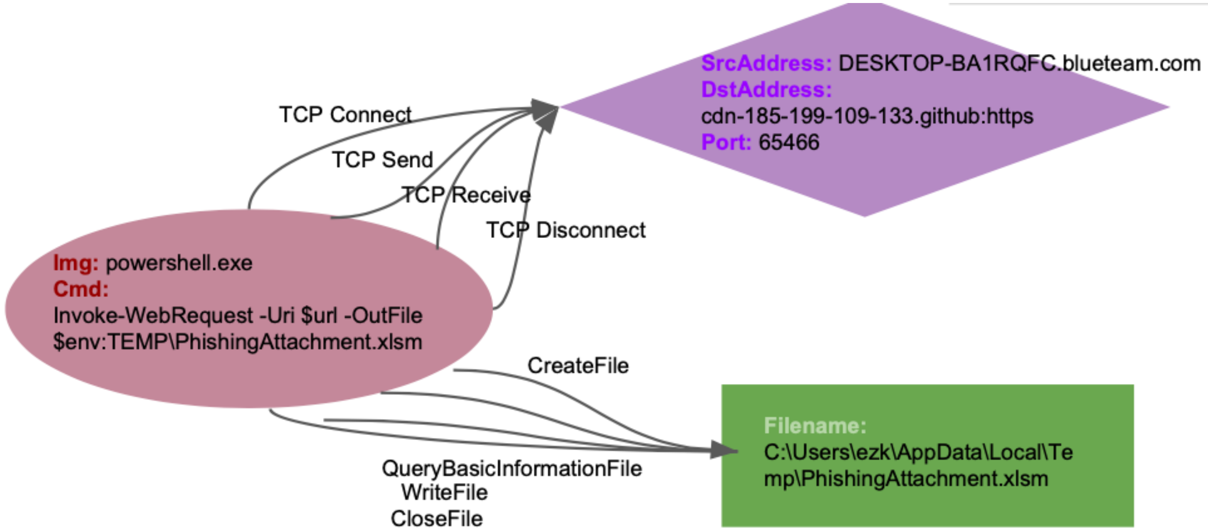
Finding the models

My task

目前可以取得

- src node 有一個 embedding
- relation 有一個 embedding
- dst node 有一個 embedding





目標：訓練一個 classifier 可以考慮到 provenance graph 中 interaction 的特性：

1. timestamp
2. 不同 dst node 是來自同個 src node
3. multi relation

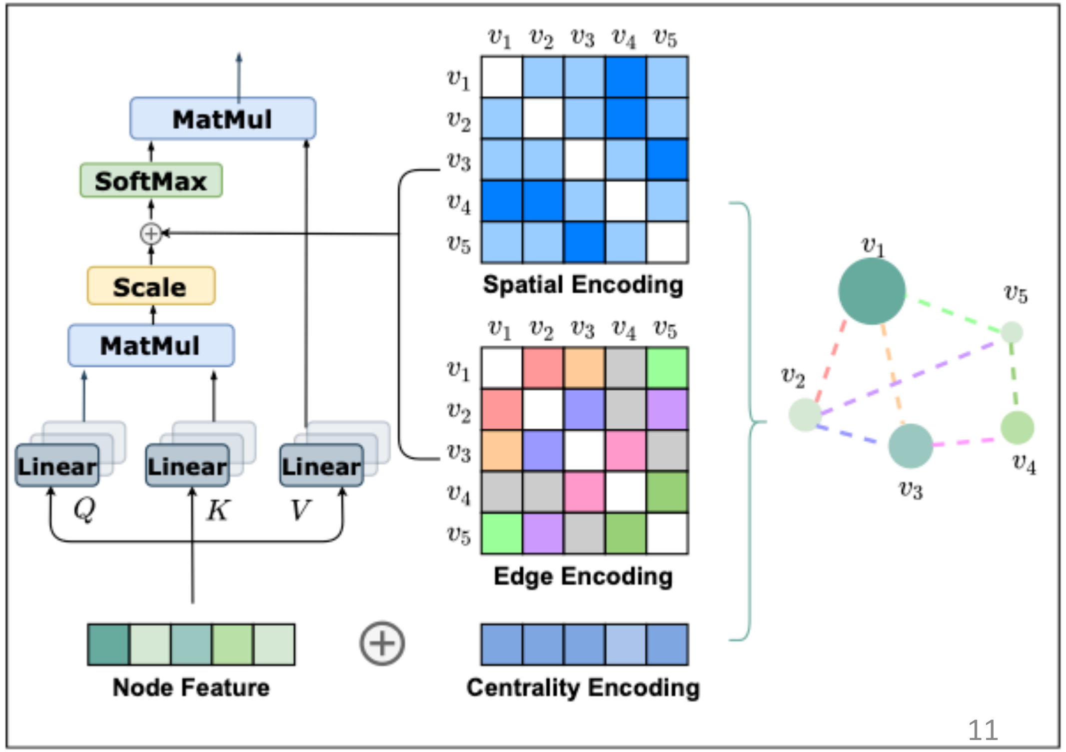
- Find a **multi-head self-attention NN** that can explain the **multi-relation** of out graph
- Find a **multi-label** classification

- **GNN** is my current searching direction – **GCN, GAT**

Graphormer

- Published by Microsoft
- Structural encoding: **Centrality** Encoding, **Edge** Encoding
- The only available graph transformer model
- Realize on **Transformer**

- Do layer normalization and feed-forward blocks **before** applying **multi-head self-attention** instead of **after**.
- Need to find out the input format and whether our datas could be the input.



Graphormer

- Paper
- Explanation of the paper
- How to use Graphormer to train a graph classification

Others may be useful

- **Graph Transformer** – improvement of the GNN
- **Graph attention network (GAT)** for node classification
- **Multilabel graph classification** using GAT

- MULTIHEADATTENTION
- Self-attention does not need O(n^2) memory

Future Plan

Plan of Next Week

- For **TRAM**
  - Try to use the real dataset to upload and then labeled them

- For **Model**
  - Find out what the input of the Graphormer should be
  - If Graphmor is feasible, read the paper
  - Try to implement or use the simplest model to train

  - if Graphormer is not feasible, try some more models

Appendix

List of Models and Useful Techniques

- Graphormer
- Graph Transformer
- Graph attention network (GAT) for node classification
- Multilabel graph classification using graph attention networks
- MULTIHEADATTENTION

- Self-attention does not need O(n^2) memory

Graphormer’s Input

- By using the centrality encoding in the input, the softmax attention can catch the node importance signal in the queries and the keys.
- Someone use the **ogbg-mohiv** dataset

Transformer. The Transformer architecture consists of a composition of Transformer layers [49]. Each Transformer layer has two parts: a self-attention module and a position-wise feed-forward network (FFN). Let $H = \left[h_1^\top, \cdots, h_n^\top\right]^\top \in \mathbb{R}^{n\times d}$ denote the input of self-attention module where d is the hidden dimension and $h_i \in \mathbb{R}^{1\times d}$ is the hidden representation at position i. The input H is projected by three matrices $W_Q \in \mathbb{R}^{d\times d_K}, W_K \in \mathbb{R}^{d\times d_K}$ and $W_V \in \mathbb{R}^{d\times d_V}$ to the corresponding representations Q, K, V. The self-attention is then calculated as:

$$Q = HW_Q, \quad K = HW_K, \quad V = HW_V, \tag{3}$$

$$A = \frac{QK^\top}{\sqrt{d_K}}, \quad \mathrm{Attn}\left(H\right) = \mathrm{softmax}\left(A\right)V, \tag{4}$$

where A is a matrix capturing the similarity between queries and keys. For simplicity of illustration, we consider the single-head self-attention and assume $d_K = d_V = d$. The extension to the multi-head attention is standard and straightforward, and we omit bias terms for simplicity.

Graphormer's Implementation

- Based on the classic Transformer -> do I think the input may be same as the transformer(embedding).

3.2 Implementation Details of Graphormer

Graphormer Layer. Graphormer is built upon the original implementation of classic Transformer encoder described in [49]. In addition, we apply the layer normalization (LN) before the multi-head self-attention (MHA) and the feed-forward blocks (FFN) instead of after [53]. This modification has been unanimously adopted by all current Transformer implementations because it leads to more effective optimization [43]. Especially, for FFN sub-layer, we set the dimensionality of input, output, and the inner-layer to the same dimension with d. We formally characterize the Graphormer layer as below:

$$h'^{(l)} = \mathrm{MHA}(\mathrm{LN}(h^{(l-1)})) + h^{(l-1)} \tag{8}$$

$$h^{(l)} = \mathrm{FFN}(\mathrm{LN}(h'^{(l)})) + h'^{(l)} \tag{9}$$

Transformer Architecture

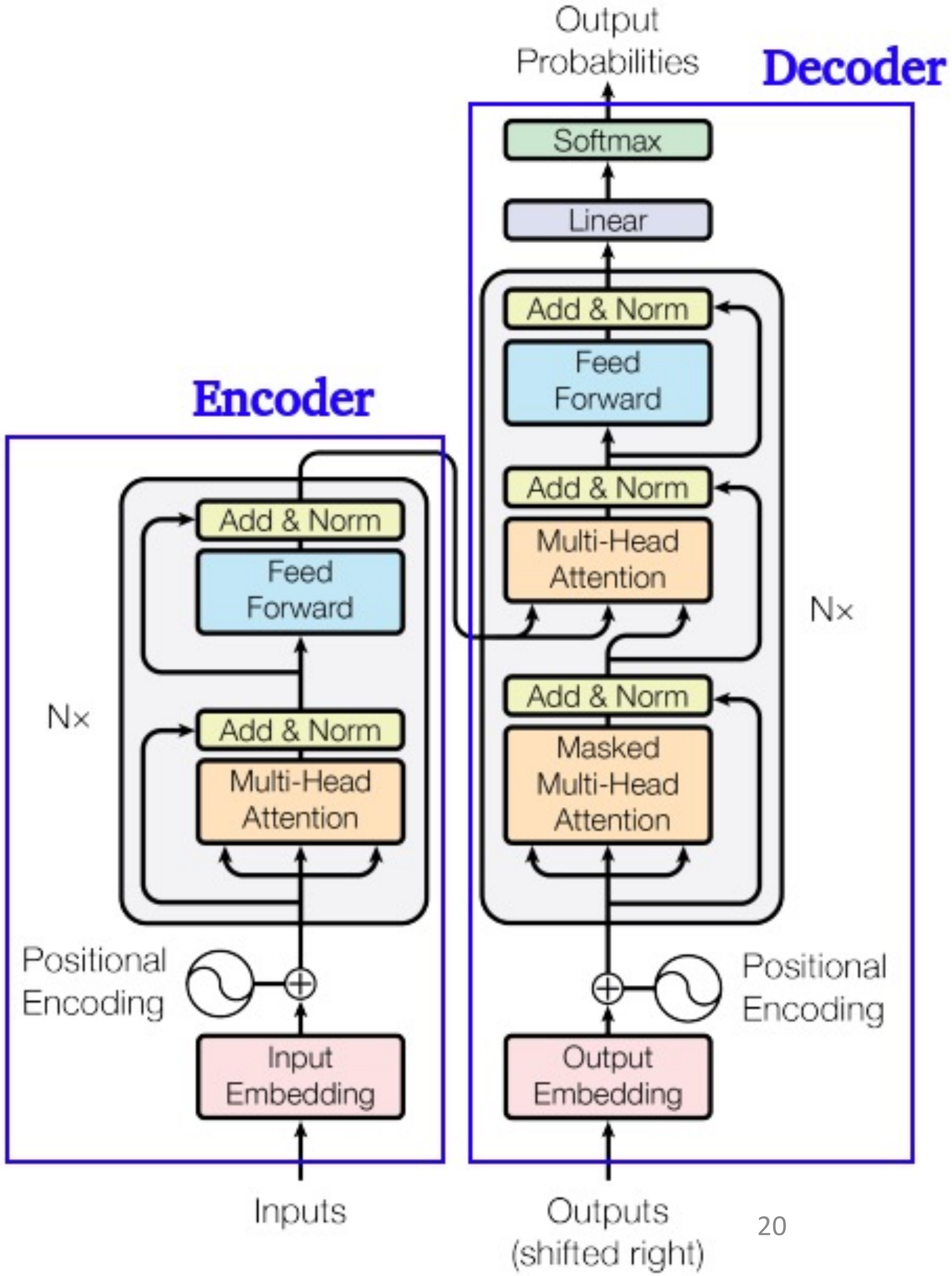
- Input is a sequence (node's embedding)

Transformer. The Transformer architecture consists of a composition of Transformer layers [49]. Each Transformer layer has two parts: a self-attention module and a position-wise feed-forward network (FFN). Let $H = \left[h_1^\top, \cdots, h_n^\top\right]^\top \in \mathbb{R}^{n\times d}$ denote the input of self-attention module where d is the hidden dimension and $h_i \in \mathbb{R}^{1\times d}$ is the hidden representation at position i. The input H is projected by three matrices $W_Q \in \mathbb{R}^{d\times d_K}$, $W_K \in \mathbb{R}^{d\times d_K}$ and $W_V \in \mathbb{R}^{d\times d_V}$ to the corresponding representations Q, K, V. The self-attention is then calculated as:

$$Q = HW_Q, \quad K = HW_K, \quad V = HW_V, \tag{3}$$

$$A = \frac{QK^\top}{\sqrt{d_K}}, \quad \mathrm{Attn}\left(H\right) = \mathrm{softmax}\left(A\right)V, \tag{4}$$

where A is a matrix capturing the similarity between queries and keys. For simplicity of illustration, we consider the single-head self-attention and assume $d_K = d_V = d$. The extension to the multi-head attention is standard and straightforward, and we omit bias terms for simplicity.



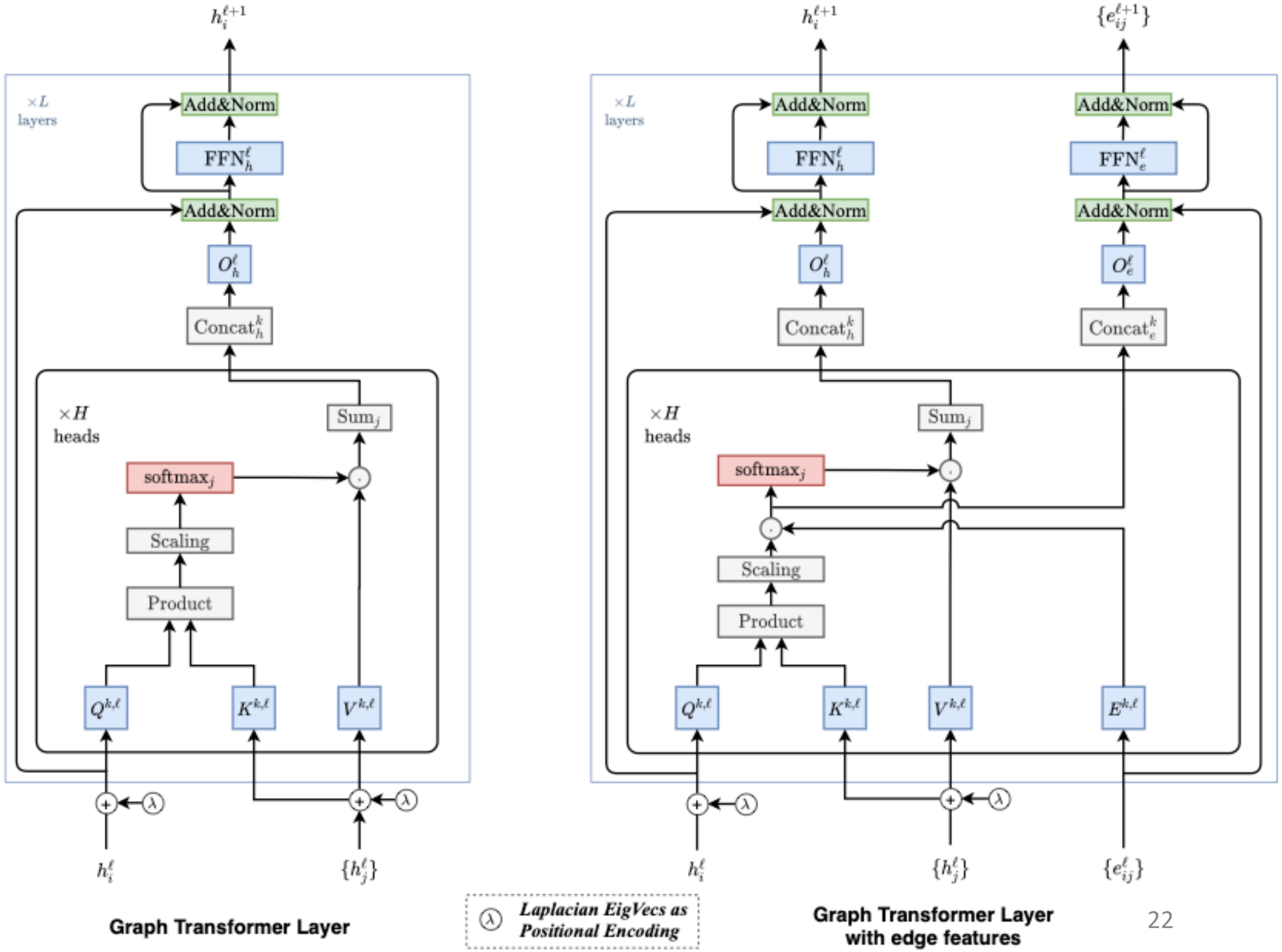
Graphormer’s Encoding

- Centrality Encoding: 可能有某個 node 特別重要，we simply add centrality encoding to the node features as the input.

- Edge Encoding in attention: 在許多圖形任務中，邊緣也具有結構特徵，例如在分子圖中，原子對之間的特徵可以描述它們之間的鍵類型。這些特徵對於圖形表示非常重要，將它們與節點特徵一起編碼到網絡中是必不可少的。在先前的研究中，主要有兩種邊編碼方法。
  - 第一種方法是將邊特徵添加到相關節點的特徵中。
  - 第二種方法是對於每個節點，將其相關邊的特徵與節點特徵一起在聚合中使用。
  - 然而，這樣使用邊特徵只將邊的信息傳播到相關的節點，可能不是利用邊信息表示整個圖形的有效方法。 為了更好地將邊特徵編碼到注意力層中，我們在Graphormer中提出了一種新的邊編碼方法。注意機制需要估計每對節點(vi, vj)之間的相關性，我們認為連接它們的邊應該在相關性中考慮，就像[34, 51]中那樣。對於每對有序節點(vi, vj)，我們找到從vi到vj的(其中之一)最短路徑SPij = (e1, e2, ..., eN)，並計算沿該路徑的邊特徵與可學習嵌入之間點積的平均值。所提出的邊編碼通過一個偏差項將邊特徵納入注意模塊。

Graph Transformer - Improvement of GNN

- Improved Graph Transformer, which extends the key design components of the NLP transformers to arbitrary graphs.



Graph Transformer - Improvement of GNN

- https://arxiv.org/pdf/2012.09699v2.pdf
- https://github.com/graphdeeplearning/graphtransformer

Graph attention network (GAT) for node classification

- In this tutorial, we will implement a specific graph neural network known as a GAT to **predict labels** of scientific **papers** based on what **type of papers cite** them.

(Multi-head) graph attention layer

The GAT model implements multi-head graph attention layers. The `MultiHeadGraphAttention` layer is simply a concatenation (or averaging) of multiple graph attention layers (`GraphAttention`), each with separate learnable weights `W`. The `GraphAttention` layer does the following:

Graph attention network (GAT) for node classification

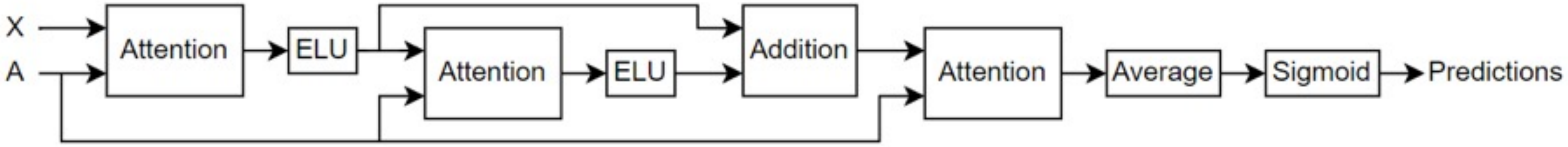
- https://keras.io/examples/graph/gat_node_classification/
- For original GAT: https://arxiv.org/pdf/1710.10903.pdf

- other: https://towardsdatascience.com/graph-attention-networks-in-python-975736ac5c0c

Multilabel graph classification using graph attention networks

- The model uses a masked **multi-head self-attention** mechanism to aggregate features across the **neighborhood** of a node, that is, the set of nodes that are directly connected to the node.
- The **mask**, which is obtained from the adjacency matrix, is used to prevent *attention* between nodes that are not in the same neighborhood.

Define the model. The model takes as input a feature matrix X and an adjacency matrix A and outputs categorical predictions.



Multilabel graph classification using graph attention networks

- https://www.mathworks.com/help/deeplearning/ug/multilabel-graph-classification-using-graph-attention-networks.html

MULTIHEADATTENTION

- Allows the model to jointly attend to information from different representation subspaces

$$\mathrm{MultiHead}(Q, K, V) = \mathrm{Concat}(head_1, \ldots, head_h)W^O$$

$$\text{where } head_i = \mathrm{Attention}(QW_i^Q, KW_i^K, VW_i^V).$$

- Determine mask type and combine masks if necessary.

```
merge_masks(attn_mask, key_padding_mask, query)
```

MULTIHEADATTENTION

- https://pytorch.org/docs/stable/generated/torch.nn.MultiheadAttention.html
- https://arxiv.org/abs/1706.03762

Self-attention does not need O(n^2) memory

- Only need O(log n) space complexity (usually considered to be O(n^2))

Sequence length	$n = 2^8$	2^{10}	2^{12}	2^{14}	2^{16}	2^{18}	2^{20}
Size of inputs and outputs	160KB	640KB	2.5MB	10MB	40MB	160MB	640MB
Memory overhead of standard attention	270KB	4.0MB	64MB	1GB	OOM	OOM	OOM
Memory overhead of memory-eff. attn.	270KB	4.0MB	16MB	17MB	21MB	64MB	256MB
Compute time on TPUv3	0.06ms	0.11ms	0.7ms	11.3ms	177ms	2.82s	45.2s
Relative compute speed	$\pm 5\%$	$\pm 5\%$	$-8 \pm 2\%$	$-13 \pm 2\%$	-	-	-

Table 2: Memory and time requirements of self-attention during **inference**.

Self-attention does not need O(n^2) memory

- https://arxiv.org/pdf/2112.05682.pdf
- https://github.com/google-research/google-research/blob/master/memory_efficient_attention/memory_efficient_attention.ipynb

Question

- QA? BERT? Can it explain the multi-relation?
- If use GNN: Graph Convolutional Networks (GCNs), Graph Attention Networks (GATs), Graph Isomorphism Networks (GINs), GraphSAGE.

- https://pytorch-geometric.readthedocs.io/en/latest/

Question

- Basically, the only useful model that can be directly imported in the realm of graph classifier is Graphomer.