Progess of the Project

Tsung-Min Pai

2023/9/16

Outline

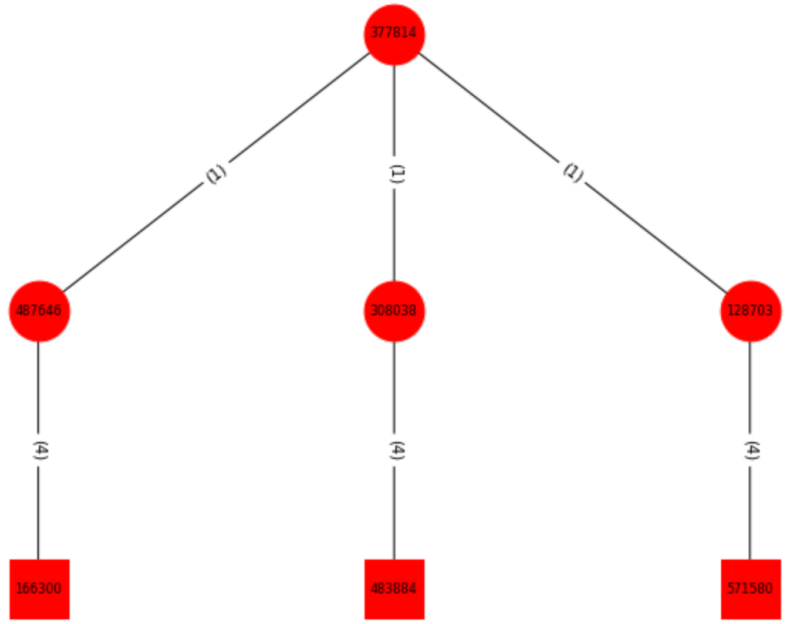
- **GNN**
  - **Experiment 1 and 2**
  - **Experiment 3**
- **Future Work**

Experiment 1 and 2

Experiment 1 and 2

- **Experiment 1:**
  - Dataset is 165 APs with 11 versions of embedding
  - Graph classification

- **Experiment 2:**
  - Experiment 1 + **benign** data
  - Benign made from benign.txt → 1000 graphs
  - Graph classification



(Smaple of the graph in exp1 and exp2)

Dataset

Format:

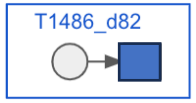

```
{"label": 10, "num_nodes": 3, "node_feat": [205565, 733769, 250773], "edge_attr": [23, 23], "edge_index": [[0, 0], [1, 2]]}
{"label": 11, "num_nodes": 3, "node_feat": [470650, 663446, 627322], "edge_attr": [23, 23], "edge_index": [[0, 0], [1, 2]]}
{"label": 15, "num_nodes": 2, "node_feat": [9863, 103498], "edge_attr": [23], "edge_index": [[0], [1]]}
{"label": 16, "num_nodes": 2, "node_feat": [157277, 753159], "edge_attr": [23], "edge_index": [[0], [1]]}
{"label": 22, "num_nodes": 36, "node_feat": [83068, 614681, 444724, 266227, 121794, 623948, 116790, 769462, 255741, 169794,
```

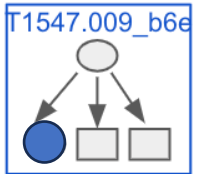

- Have 165 APs, each AP has 1000 variation → nodes are different but relations are same
- 0~99 test, 100~199 validation, 200~999 train → 1:1:8

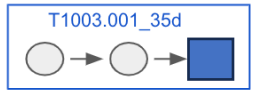
- Use transR_50, transE_50, transH_50, secureBERT… as embedding → **11 versions**
- **secureBERT** → dimension = 250, 150, 100, 50

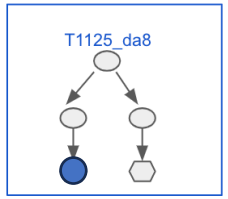
```
pca = PCA(n_components=DIM)
ent_embeddings = pca.fit_transform(ent_embeddings)
```

Dataset

- Benign graphs for experiment 2:
  1. 400: Leaf nodes + their source nodes 

  2. 300: Leaf nodes + their source nodes with source nodes' neighbor nodes 

  3. 200: Leaf nodes + their 2 layer source nodes 

  4. 100: Leaf nodes + their 2 layer source nodes with source nodes' neighbor nodes 

  - For version 2 and 4 graphs:
    - Constrain the # of relation between the same nodes within 8
    - Constrain the # of the triplets in the graph within 32

Experiment 1 and 2

- Record the training message in a **log file**
- Also record the **classification report** supportedd by sklearn

- Log file:

```
08/29/2023, 09:04:02# labels of Validation: tensor([ 96,   4, 129, 111,  84, 162, 102, 103, 132, 114,  32, 110,  57,  85,
       149,  74], device='cuda:3') torch.Size([16])
08/29/2023, 09:04:02# predicted of Validation: tensor([ 96, 153, 113, 153,  84, 153, 153, 113, 161, 113,  32, 113, 153,  85,
       149, 153], device='cuda:3') torch.Size([16])
08/29/2023, 09:04:03# Validation Loss: 2.6715 | Validation Accuracy: 0.3879
```

- Classification report:

	precision	recall	f1-score	support
T1003.001_0ef4cc7b-611c-4237-b20b-db36b6906554	1.00	1.00	1.00	100
T1003.001_35d92515122effdd73801c6ac3021da7	1.00	1.00	1.00	100
T1003.002_5a484b65c247675e3b7ada4ba648d376	0.00	0.00	0.00	100
T1003.002_7fa4ea18694f2552547b65e23952cabb	1.00	1.00	1.00	100
T1003.003_9f73269695e54311dd61dc68940fb3e1	0.00	0.00	0.00	100
T1003.003_f049b89533298c2d6cd37a940248b219	0.00	0.00	0.00	100

Similar with the MLP, RNN:
Can barely predict the graph with only one triplet

Graph Attention Network - GAT

Model:

```
class GAT(nn.Module):
    def __init__(self, in_dim, hidden_dim, out_dim, num_heads, dropout_prob=0.2):
        super(GAT, self).__init__()
        # do not check the zero in_degree since we have all the complete graph
        self.layer1 = GATConv(in_dim, hidden_dim, num_heads=num_heads, activation=F.relu, allow_zero_in_degree=True)
        self.layer2 = GATConv(hidden_dim * num_heads, out_dim, num_heads=num_heads, allow_zero_in_degree=True)

        # Adding Dropout for regularization
        self.dropout = nn.Dropout(dropout_prob)

    def forward(self, g, h):
        # Apply GAT layers
        h = self.layer1(g, h)
        h = h.view(h.shape[0], -1)
        h = F.relu(h)
        h = self.dropout(h)
        h = self.layer2(g, h).squeeze(1)

        # Store the output as a new node feature
        g.ndata['h_out'] = h

        # Use mean pooling to aggregate this new node feature
        h_agg = dgl.mean_nodes(g, feat='h_out')
        return h_agg
```

Experiment 1 and 2 with GAT

- Total: 25 epochs
  - Optimizer = AdamW(model.parameters(), lr=5e-4)
  - Criterion = nn.CrossEntropyLoss()
  - Batch size = 16

- Except **secureBERT**, all about 35~40% test accuracy
- secureBERT family ≈ 12% test accuracy

- Experiment 1 and 2 have similar performance
  - since benign only has 1000 graph → kind of balance
  - If we make the benign graph much more than AP(real data) → imbalance

Graph Convolutional Network - GCN

Model:

```
class GCN(nn.Module):
    def __init__(self, in_feats, hidden_size, num_classes):
        super(GCN, self).__init__()
        self.conv1 = GraphConv(in_feats, hidden_size, allow_zero_in_degree=True)
        self.conv2 = GraphConv(hidden_size, num_classes, allow_zero_in_degree=True)

    def forward(self, g, inputs):
        h = self.conv1(g, inputs)
        h = torch.relu(h)
        h = self.conv2(g, h)

        g.ndata['h'] = h
        h_mean = dgl.mean_nodes(g, 'h')
        return h_mean
```

- Since the bad performance (**14%** accuracy on **transH_50** test), I didn't go any further with GCN

Graph SAmple and aggreGateE - GraphSAGE

Model:

```
class GraphSAGE(nn.Module):
    def __init__(self, in_dim, hidden_dim, out_dim):
        super(GraphSAGE, self).__init__()
        self.layer1 = dglnn.SAGEConv(in_dim, hidden_dim, 'lstm')
        self.layer2 = dglnn.SAGEConv(hidden_dim, out_dim, 'lstm')

    def forward(self, g, inputs):
        h = self.layer1(g, inputs)
        h = torch.relu(h)
        h = self.layer2(g, h)

        g.ndata['h'] = h
        h_mean = dgl.mean_nodes(g, 'h')
        return h_mean
```

- **In_dim**: dimension of the node embedding
- **out_dim**: # of the classes
- **Aggregate type**: mean, gcn, pool, lstm → performance of **lstm** is the best but take 5 times of the time in training

Experiment 1 and 2 with GraphSAGE

- Total: 25 epochs
  - Optimizer = AdamW(model.parameters(), lr=5e-4)
  - Criterion = nn.CrossEntropyLoss()
  - Batch size = 16

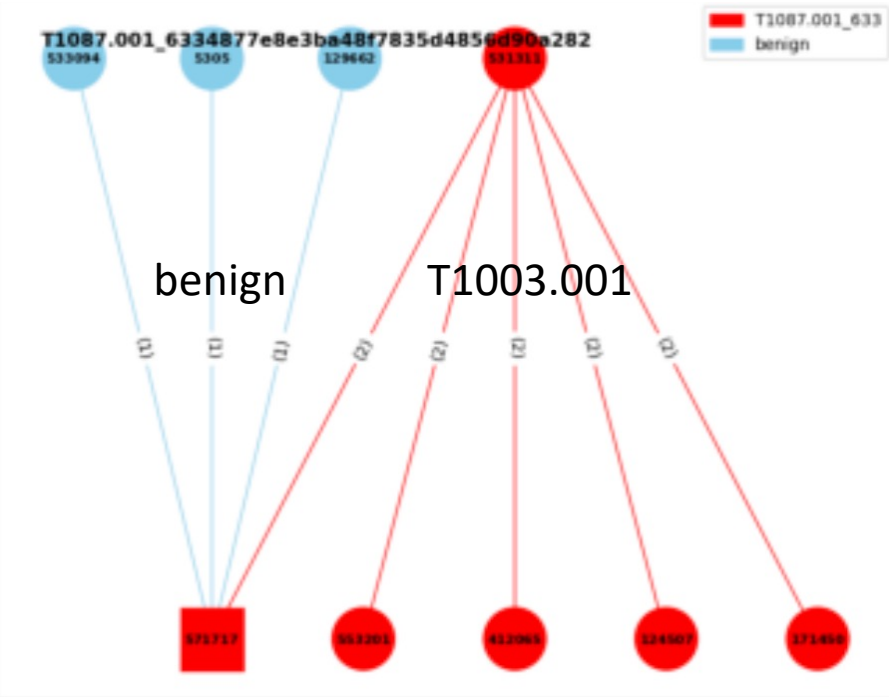
- All about 60~62% test accuracy → increase 20% compared to GAT
- All secureBERT family ≈ 60% test accuracy

- Experiment 1 and 2 have similar performance
  - since benign only has 1000 graph → kind of balance
  - If we make the benign graph much more than AP(real data) → imbalance

Experiment 3

Experiment 3

- **Experiment 3:**
  - Consider the **neighbor** benign nodes
  - Edge classification
    - I think it's more like an triplet classification?
  - Given a graph → label the triplets with the benign or the specific AP



Source txt file:

```
853776 595218 13 a
593289 563219 17 b
388326 563219 17 b
```

- a means attack pattern
- b means benign

Experiment 3

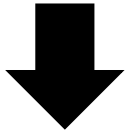
- Concept:
  1. Use the node embedding from the GraphSAGE model
  2. Concatenate with the edge embedding
  3. Train the MLP model to do the triplet classification

- Current Difficulty:
  1. Format of the dataset

Dataset

Format in experiment 1 and 2:


```
{"label": 10, "num_nodes": 3, "node_feat": [205565, 733769, 250773], "edge_attr": [23, 23], "edge_index": [[0, 0], [1, 2]]}
{"label": 11, "num_nodes": 3, "node_feat": [470650, 663446, 627322], "edge_attr": [23, 23], "edge_index": [[0, 0], [1, 2]]}
{"label": 15, "num_nodes": 2, "node_feat": [9863, 103498], "edge_attr": [23], "edge_index": [[0], [1]]}
{"label": 16, "num_nodes": 2, "node_feat": [157277, 753159], "edge_attr": [23], "edge_index": [[0], [1]]}
{"label": 22, "num_nodes": 36, "node_feat": [83068, 614681, 444724, 266227, 121794, 623948, 116790, 769462, 255741, 169794,
```

Format in experiment 3:


```
{"labels": [45, 65, 45, 45], "num_nodes": 4, "node_feat": [578353, 695633, 234474, 883199], "edge_attr": [24, 2, 7, 2],
{"labels": [45, 65, 45, 45], "num_nodes": 4, "node_feat": [578353, 234474, 1085219, 1079260], "edge_attr": [24, 2, 7, 2]
{"labels": [45, 65, 45, 45], "num_nodes": 4, "node_feat": [578353, 946954, 234474, 391415], "edge_attr": [24, 2, 7, 2],
```


- From graph classification to **edge classification**
- # of labels = # of triplets
- Haven’t successfully trained → # of labels of each data(graph) are different

Official Edge Classification Sample of DGL

```
import dgl.function as fn
class DotProductPredictor(nn.Module):
    def forward(self, graph, h):
        # h contains the node representations computed from the GNN defined
        # in the node classification section (Section 5.1).
        with graph.local_scope():
            graph.ndata['h'] = h
            graph.apply_edges(fn.u_dot_v('h', 'h', 'score'))
            return graph.edata['score']
```

```
class SAGE(nn.Module):
    def __init__(self, in_feats, hid_feats, out_feats):
        super().__init__()
        self.conv1 = dglnn.SAGEConv(
            in_feats=in_feats, out_feats=hid_feats, aggregator_type='mean')
        self.conv2 = dglnn.SAGEConv(
            in_feats=hid_feats, out_feats=out_feats, aggregator_type='mean')

    def forward(self, graph, inputs):
        # inputs are features of nodes
        h = self.conv1(graph, inputs)
        h = F.relu(h)
        h = self.conv2(graph, h)
        return h
```

Official Edge Classification Sample of DGL

```
class Model(nn.Module):
    def __init__(self, in_features, hidden_features, out_features):
        super().__init__()
        self.sage = SAGE(in_features, hidden_features, out_features)
        self.pred = DotProductPredictor()

    def forward(self, g, x):
        h = self.sage(g, x)
        score = self.pred(g, h)
        output = torch.softmax(score, dim=1)
        print(output)
        return output
```

```
src = np.random.randint(0, 100, 500)
dst = np.random.randint(0, 100, 500)
# make it symmetric
edge_pred_graph = dgl.graph((np.concatenate([src, dst]), np.concatenate([dst, src])))
# synthetic node and edge features, as well as edge labels
edge_pred_graph.ndata['feature'] = torch.randn(100, 10)
edge_pred_graph.edata['feature'] = torch.randn(1000, 10)
edge_pred_graph.edata['label'] = torch.randn(1000)
# synthetic train-validation-test splits
edge_pred_graph.edata['train_mask'] = torch.zeros(1000, dtype=torch.bool).bernoulli(0.6)
```

Official Edge Classification Sample of DGL

```
label:  tensor([88, 67, 53, 32, 15,  9, 39, 81, 78, 21, 79, 89, 51, 76, 56, 57,  7,  0,
                 0, 63, 52, 73, 54, 85, 24, 23, 17, 90, 48, 36, 20,  6, 23, 39, 35, 64,
                51, 18, 69, 58, 73, 38, 99, 34, 64, 80, 50, 91, 48, 92, 58, 23, 17,  7,
                28, 25, 22, 96, 20, 54, 65, 54,  8, 95, 13, 69,  4, 67, 96, 43, 96, 90,
                80, 22, 45,  5, 98, 88, 94, 88,  7, 59, 47, 68, 95, 91, 23, 34, 31, 70,
                51, 70, 39, 53, 11, 34, 49, 77, 65, 39,  0, 43, 70, 69, 27, 22, 28, 10,
                96, 57, 54, 40, 74, 25,  2, 59, 29, 98, 99, 64, 43, 87,  3, 85, 61, 45,
                54,  6, 17, 52, 66, 96, 96, 72,  1, 64, 78, 10, 15, 77, 35, 57, 85, 34,
                84, 28, 68, 35, 84, 67, 67,  8, 16,  9, 53, 58, 49, 41,  7, 40, 38, 47,
                45, 89,  8,  8,  3, 37, 70, 36, 85, 30, 13, 20, 32, 77, 28, 57, 25, 34,
                64, 98, 44, 86, 91, 68, 12, 99, 53, 15, 46, 18, 10, 71, 71, 67, 95, 90,
                61, 80, 73, 59, 72, 23, 27, 65, 37, 12, 68, 30, 79, 46, 40, 70,  8, 64,
                79, 62, 64, 88, 48, 12, 49, 74, 23,  7, 28,  0, 69, 87, 99, 56, 94, 40,
                27, 63,  1, 40, 49, 75, 58,  2, 36, 49,  5, 13, 35, 44, 80, 72, 49, 97,
                43, 89, 63, 47, 64,  2, 85,  7, 50, 24,  6, 22,  9, 20, 21, 57, 58, 51,
                39, 48,  7, 13, 49, 51, 72, 34, 68, 19, 19, 54, 26, 91, 14, 21,  2, 44,
                74,  9, 43, 28, 70, 68, 27, 87, 32, 23, 29, 21, 42, 14, 12, 70, 94,  9,
                53, 57, 77, 26,  5, 81,  9, 94, 75, 14, 17, 79, 83, 16, 28, 65, 85, 54,
                31, 37, 24, 96, 52, 77, 59,  7, 43, 60, 37,  3,  5, 51, 24, 44, 89, 16,
                 5, 73, 82, 21, 18, 30, 83, 83,  6,  6, 19,  2, 25,  4, 68, 41, 20, 17,
                20, 82, 76, 93, 37, 11, 12, 74, 47, 97, 98,  9, 92,  7, 64, 92, 33, 50,
                43, 17, 28, 25, 27, 69, 24, 63, 86, 86, 42, 54, 59, 52, 42, 55, 82, 59,
                75, 37, 57, 79, 50, 70, 24, 10,  8, 89, 81, 71, 76,  9, 23, 69, 31, 91,
                63, 74, 26, 38, 77, 56, 72, 92, 69, 35, 27, 65,  2, 79, 51, 89, 11,  6,
                65, 87, 95, 24, 96, 73, 47, 75, 22, 57, 78, 44, 12,  1, 14, 97, 23, 83,
                56, 86, 56, 35, 46,  3, 95,  5, 18, 92, 57, 69,  9, 38, 91, 60, 89,  5,
                63, 15, 28, 60, 48, 36, 74, 27, 61,  3, 27, 53, 74, 78, 34, 18, 32, 58,
                63,  6, 40, 98, 30,  1, 56, 17, 17,  7, 21, 91, 54, 37])
shape torch.Size([500])
```

- Shape: [500]

```
pred:  tensor([[1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
```

- Shape: [500, 1]

- The result seems to be not suitable for our task

Future Work

Future Work

- **GNN**
  - Successfully run the experiment 3
  - Improve the performance of the model (if available)

Thanks!!

Appendix

Useful Links

https://zhuanlan.zhihu.com/p/107737824

https://zhuanlan.zhihu.com/p/315800604

https://blog.csdn.net/uncle_ll/article/details/82778750

https://docs.dgl.ai/en/1.1.x/guide_cn/minibatch-edge.html#guide-cn-minibatch-edge-classification-sampler

https://docs.dgl.ai/en/0.8.x/generated/dgl.nn.pytorch.conv.SAGEConv.html

https://docs.dgl.ai/en/0.8.x/generated/dgl.nn.pytorch.conv.GATConv.html

https://www.modb.pro/db/111133

Experiment 3

```
binary_labels = torch.zeros(batch_size, num_classes)
for i, sample_labels in enumerate(labels):
    for label in sample_labels:
        binary_labels[i, label] = 1
```

```
preds = (logits >= 0.5).to(torch.float32)
```

```
accuracy = torch.mean((preds == binary_labels).float())
```

```
criterion = nn.BCEWithLogitsLoss()
```

- Total: 25 epochs
  - Optimizer = AdamW(model.parameters(), lr=5e-4)
  - Criterion = nn.CrossEntropyLoss()
  - Batch size = 4

Graph Convolutional Network - GCN

Graph Convolutional Network - GCN

Model:

```
class GCN(nn.Module):
    def __init__(self, in_feats, hidden_size, num_classes):
        super(GCN, self).__init__()
        self.conv1 = GraphConv(in_feats, hidden_size)
        self.conv2 = GraphConv(hidden_size, num_classes)

    def forward(self, g, inputs):
        h = self.conv1(g, inputs)
        h = torch.relu(h)
        h = self.conv2(g, h)

        g.ndata['h'] = h
        hg = dgl.mean_nodes(g, 'h')
        return hg
```

- Use the **old** verison of the dataset
- Use **DGL** to be our library
- DGL data format:

```
batched_g is like:
Graph(num_nodes=96, num_edges=160, ndata_schemes={'feat': Scheme(shape=(1,), dtype=torch.int64)}, edata_schemes={})
num_nodes = 3*batch_size, num_edges = 5*batch_size

labels is like: tensor([ 76,   0,   0,   0,   0,   0,   0,   0,   0,  76,   0,  76,   0,   0,
                          0,   0,  76,   0,  30,  92,   0,   0,  76,   0,   0,   0,   0,   0,
                        116,   0,  76,  76])
```

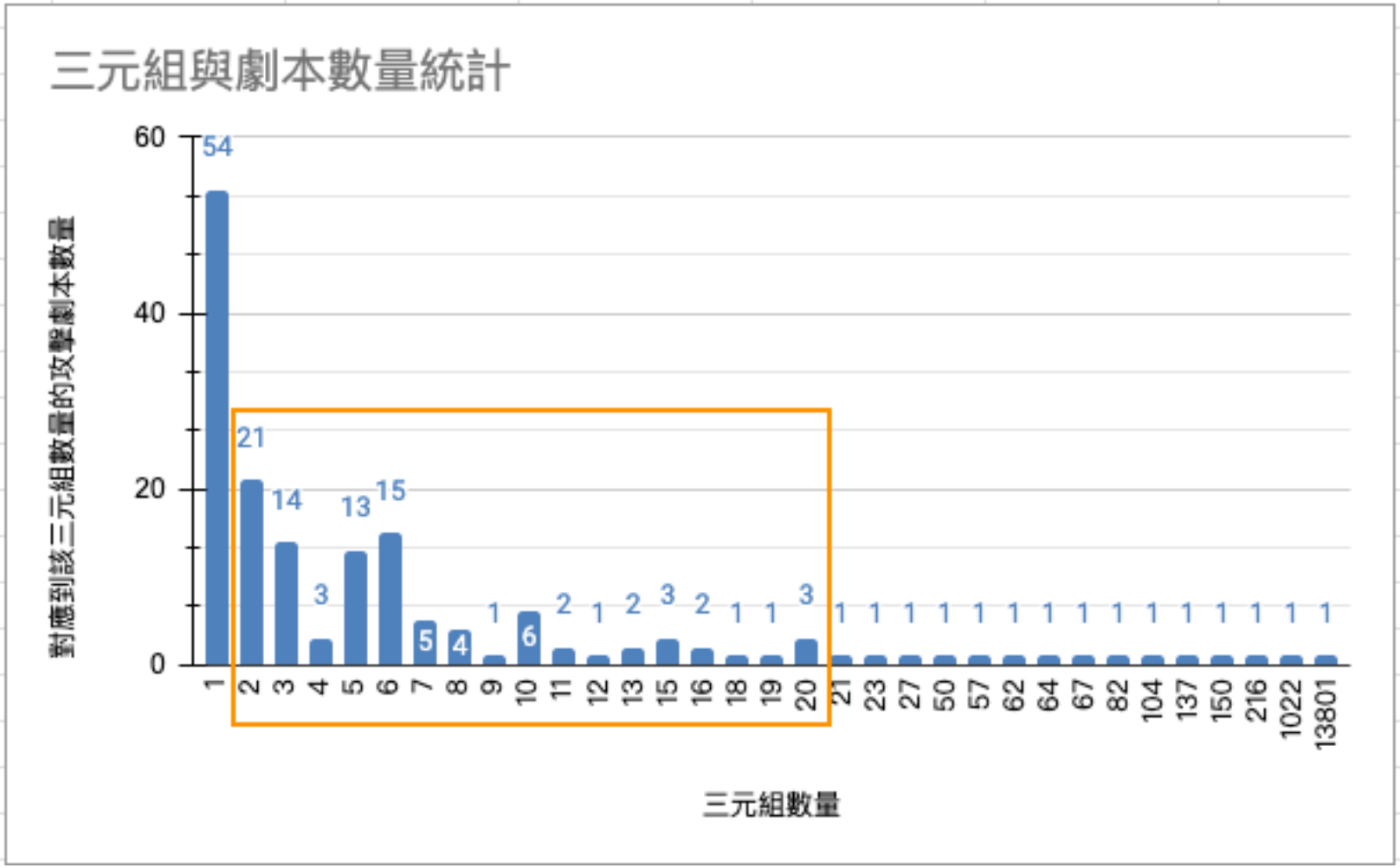
Result:

```
  0%|          | 0/120 [00:00<?, ?it/s]
Epoch 0 | Train Loss: 2625.5943 | Train Accuracy: 0.4763
  1%|          | 1/120 [00:56<1:52:21, 56.65s/it]
Validation Loss: 494.0275 | Validation Accuracy: 0.6642
 99%|▉| 119/120 [1:51:06<00:55, 55.13s/it]
Validation Loss: 0.9964 | Validation Accuracy: 0.6642
Epoch 119 | Train Loss: 0.9625 | Train Accuracy: 0.6644
100%|▉| 120/120 [1:52:03<00:00, 56.03s/it]
Validation Loss: 0.9965 | Validation Accuracy: 0.6642

Test Accuracy: 66 %
```

- GAT applied on the old data has the similar result

Appendix


三元組與劇本數量統計
對應到該三元組數量的攻擊劇本數量
三元組數量
60
40
20
0
54
21
14
3
13
15
5
4
1
6
2
1
2
3
2
1
1
3
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
2
3
4
5
6
7
8
9
10
11
12
13
15
16
18
19
20
21
23
27
50
57
62
64
67
82
104
137
150
216
1022
13801