Progess of the Project

Tsung-Min Pai

2023/9/7

Outline

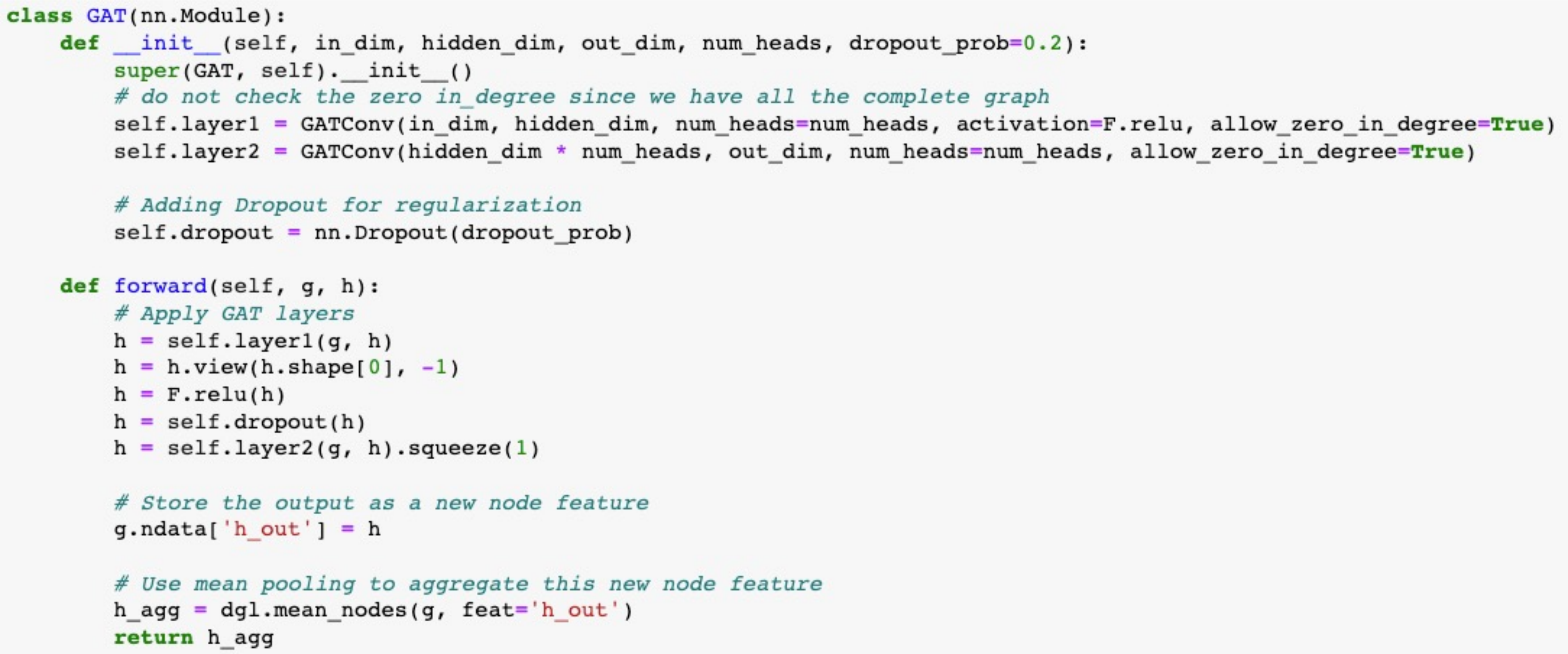
- **GAT**
  - **Experiment 1 and 2**
  - **Experiment 3**

- **Future Work**

Graph Attention Network - GAT

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
```

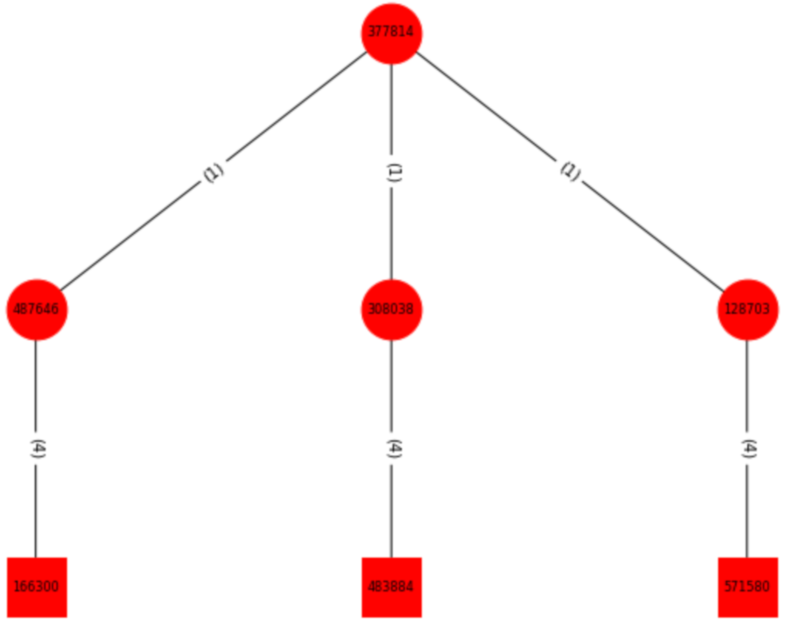
- Use the **new** verison of the dataset

Experiment 1 and 2

Experiment 1 and 2

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

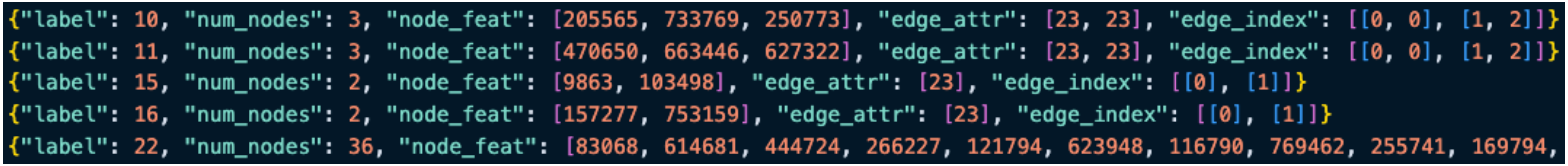
- **Experiment 2:**
  - Experiment 1 + **benign** data
  - Benign made from benign.txt → 1000 graphs
  - Graph classification



(Smaple of the graph in exp1 and exp2)

Dataset

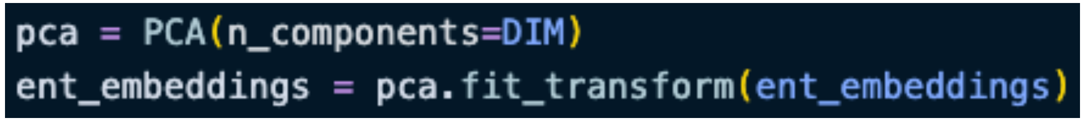
Format:



```
{"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

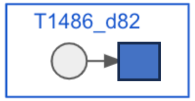
- Use transR_50, transE_50, transH_50, secureBERT… as embedding → **11 versions**
- **secureBERT** → dimension = 250, 150, 100, 50



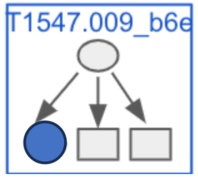
```
pca = PCA(n_components=DIM)
ent_embeddings = pca.fit_transform(ent_embeddings)
```

Dataset

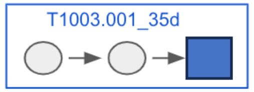
- Benign graphs for experiment 2:
  1. 400: Leaf nodes + their source nodes



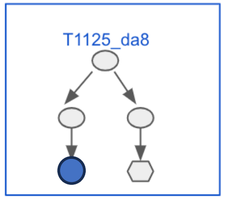
  2. 300: Leaf nodes + their source nodes with source nodes’ neighbor nodes



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



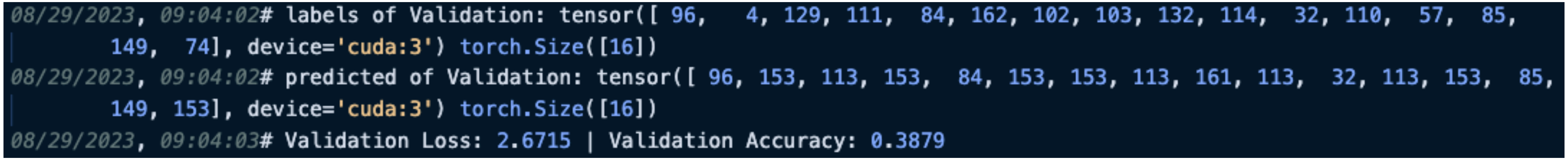
  4. 100: Leaf nodes + their 2 layer source nodes with source nodes’ neighbor nodes



  - 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

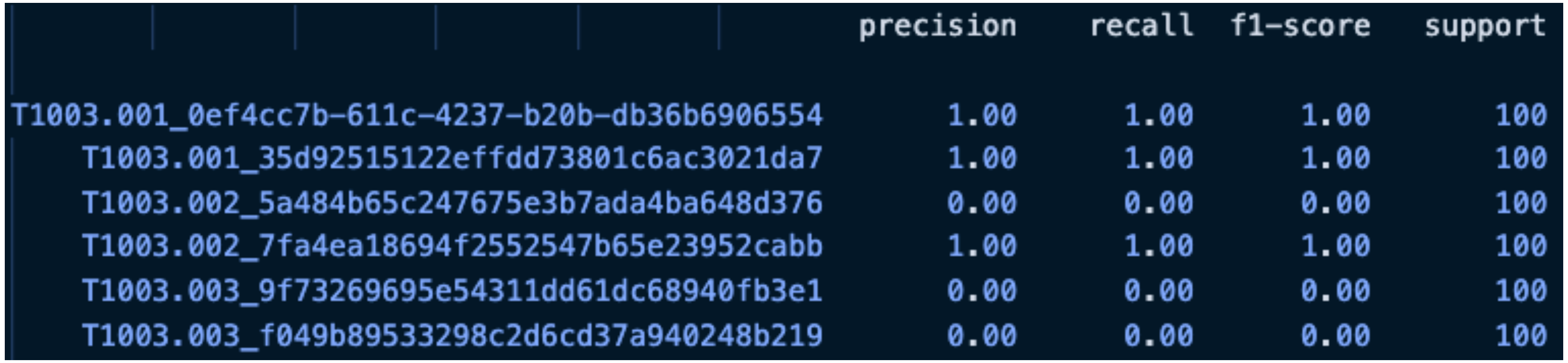
Experiment 1 and 2

- Record the training in a **log file**
- Also record the **classification report** supportedd by sklearn
- Give these files to Euni
- 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:
More triplets, more accurate

Experiment 1 and 2

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

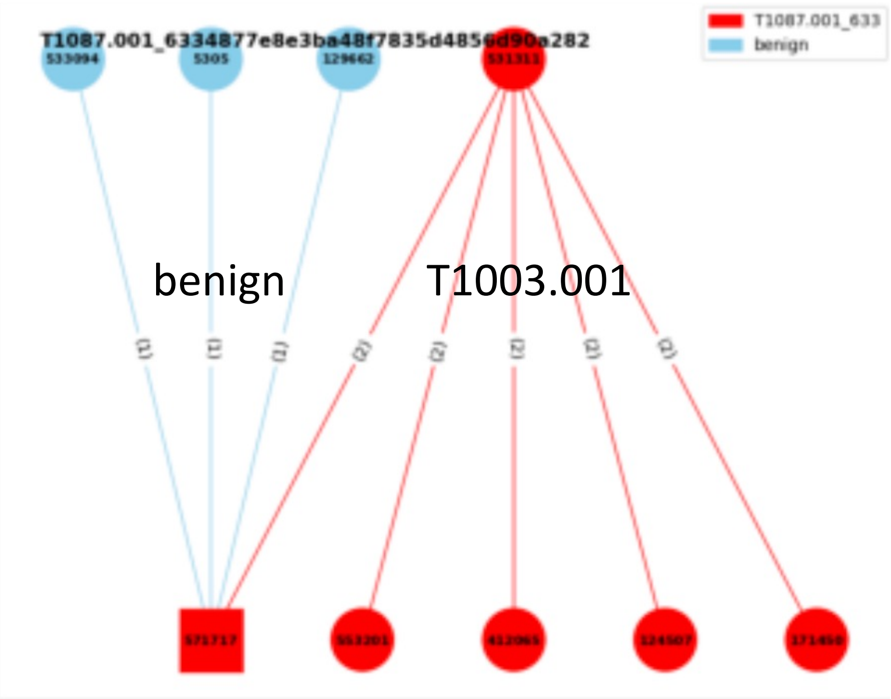
- Except **secureBERT**, all about 35~40% test accuracy
- secureBERT family ≈ 12% test accuracy

- 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

Experiment 3

Experiment 3

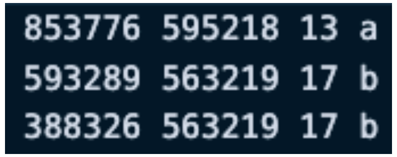
- **Experiment 3:**
  - Consider the **neighbor** benign nodes
  - Edge classification
    - I think it’s more like an triplet classification?
  - Given a graph → label the triplets with the benign or the specific AP



benign

T1003.001

Source txt file:

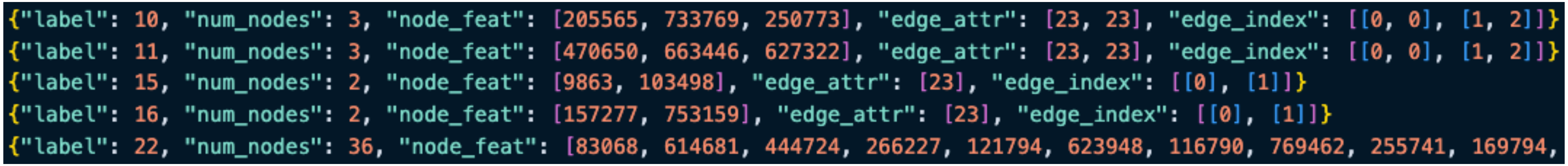


```
853776 595218 13 a
593289 563219 17 b
388326 563219 17 b
```

- a means attack pattern
- b means benign

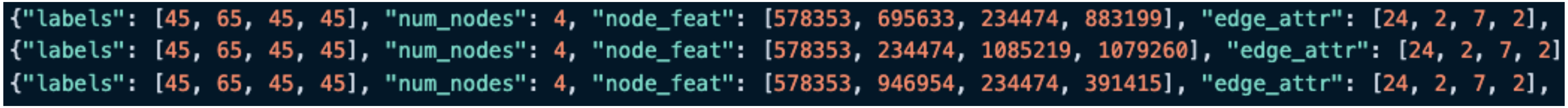
Dataset

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**, which is **multi-label** classification
- # of labels = # of triplets

- Haven't successfully trained → some tensor error

Future Work

Future Work

- **GAT**
  - Successfully run the experiment 3
  - Try GCN or different architecture of the model
  - Improve the performance of the model (if available)

Thanks!!

Appendix

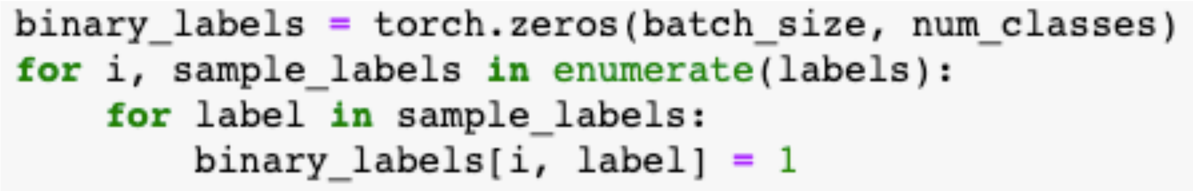
Useful Links

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

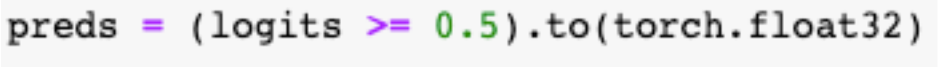
https://blog.csdn.net/uncle_ll/article/details/82778750

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

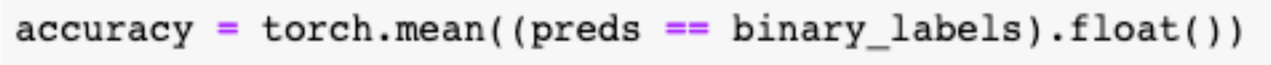
Experiment 3



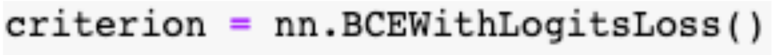
```
binary_labels = torch.zeros(batch_size, num_classes)
for i, sample_labels in enumerate(labels):
    for label in sample_labels:
        binary_labels[i, label] = 1
```



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



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



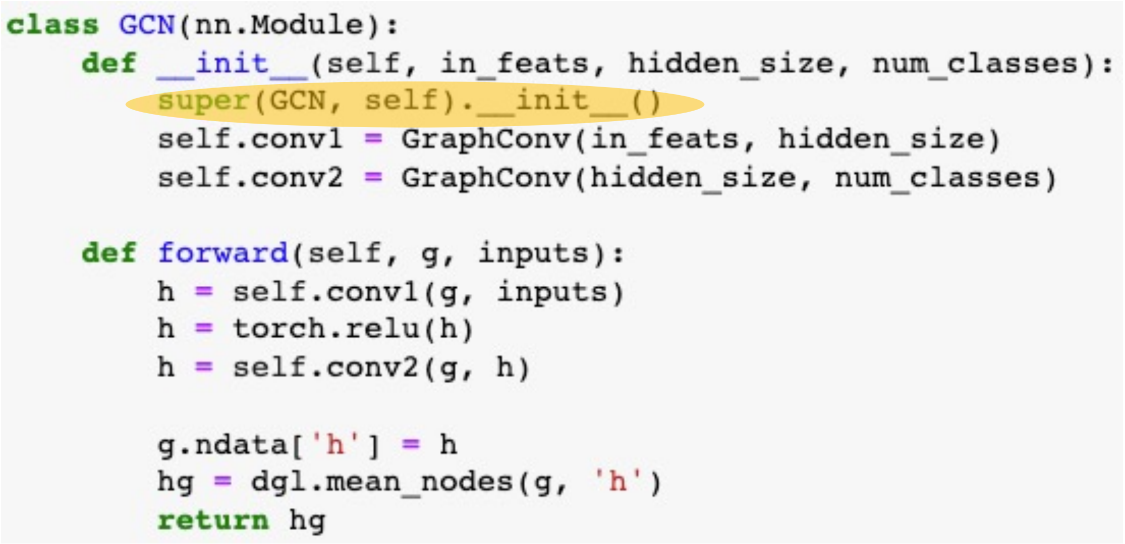
```
criterion = nn.BCEWithLogitsLoss()
```

- 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:

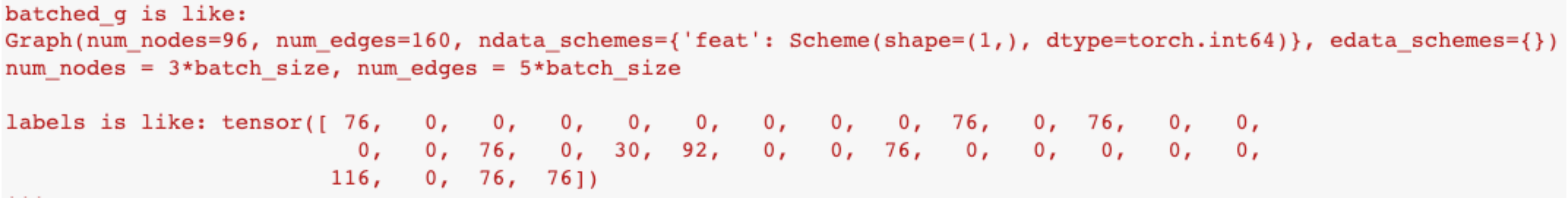


```
class GCN(nn.Module):
    def __init__(self, in_feats, hidden_size, num_classes):
        super(GCN, self).__init__()
        self.conv1 = GraphConv(in_feats, hidden_size)
        self.conv2 = GraphConv(hidden_size, num_classes)

    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
```

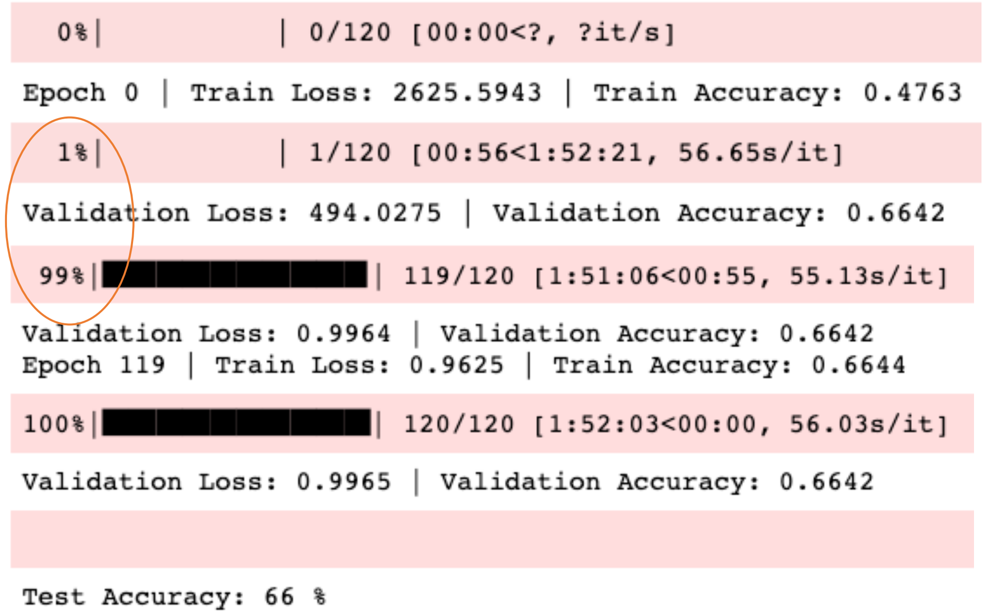
- Use the **old** verison of the dataset
- Use **DGL** to be our library
- DGL data format:



```
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:

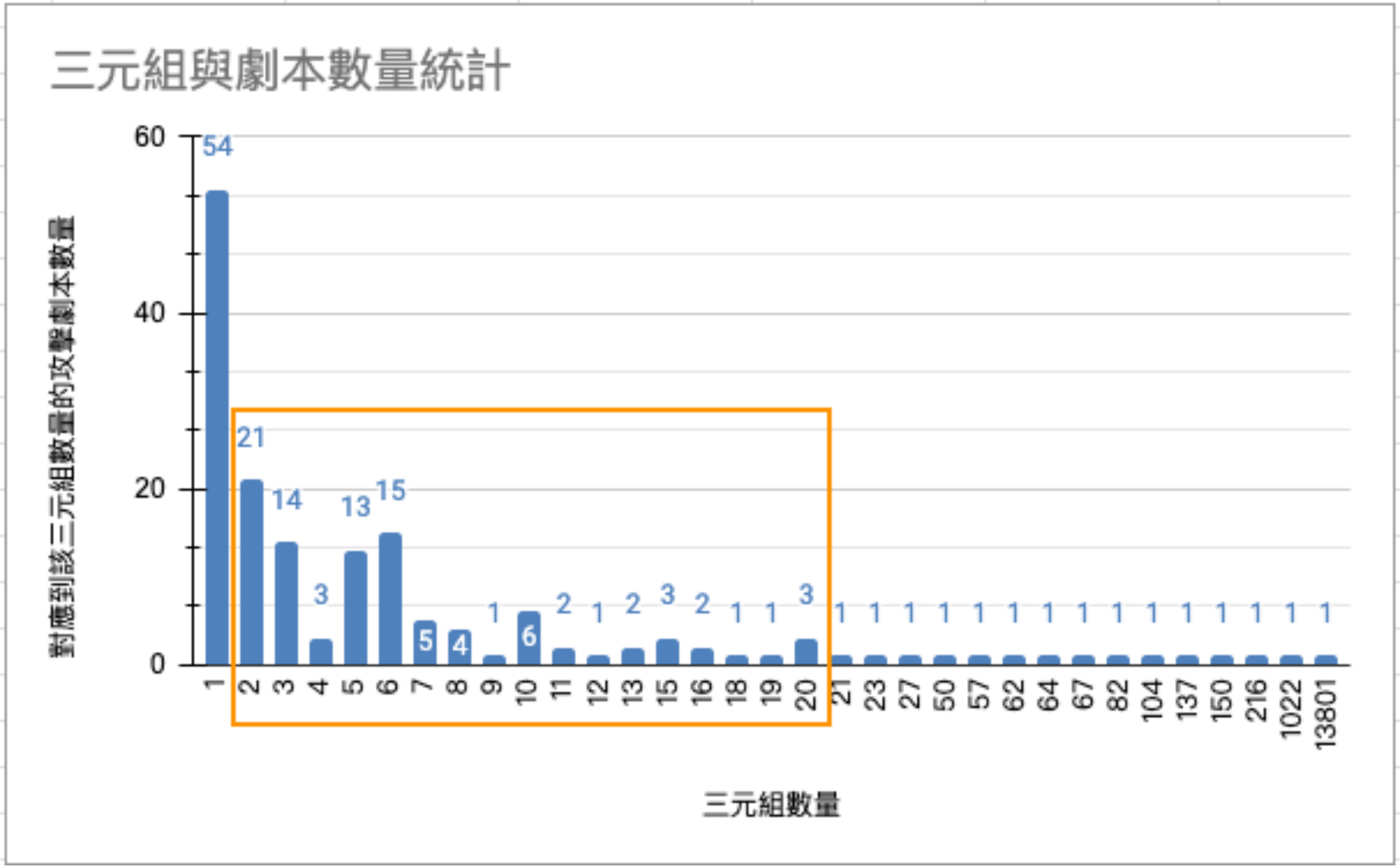


```
  0%|          | 0/120 [00:00<?, ?it/s]
Epoch 0 | Train Loss: 2625.5943 | Train Accuracy: 0.4763
  1%|          | 1/120 [00:56<1:52:21, 56.65s/it]
Validation Loss: 494.0275 | Validation Accuracy: 0.6642
 99%|█████████▉| 119/120 [1:51:06<00:55, 55.13s/it]
Validation Loss: 0.9964 | Validation Accuracy: 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
0
20
40
60
三元組數量
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
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