Progess of the Project

Tsung-Min Pai

2023/9/22

Outline

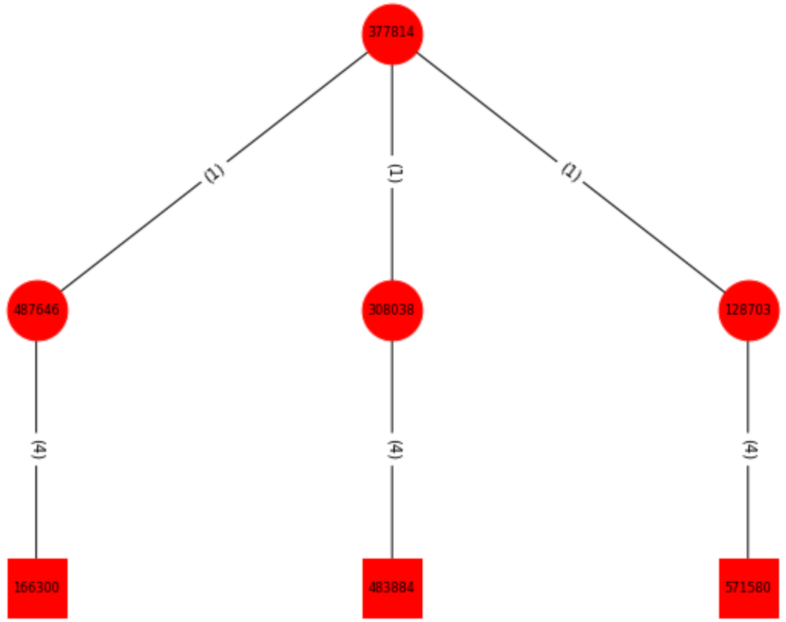
- **GNN**
  - **Experiment 1 and 2**
  - **Experiment 3**
- **Future Work**

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)

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, 'pool')
        self.layer2 = dglnn.SAGEConv(hidden_dim, out_dim, 'pool')

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

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

- **In_dim**: dimension of the node embedding
- **out_dim**: # of the classes
- **Aggregate type**: mean, gcn, pool, lstm → performance of **pool and lstm** are the similar → pool is **faster**

Experiment 1 and 2 with GraphSAGE

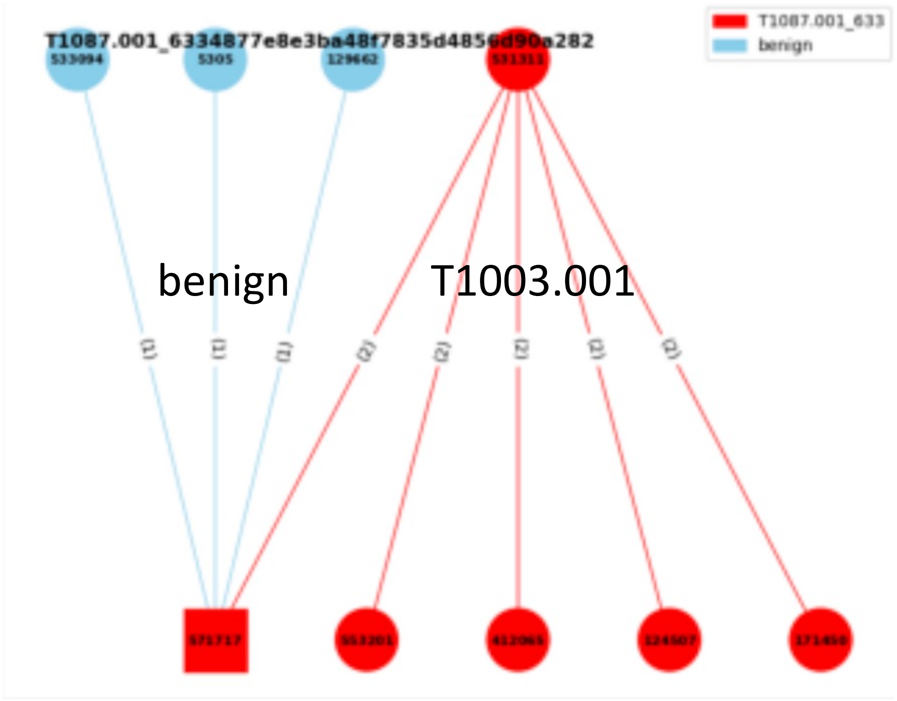
- Total: 25 epochs
  - Optimizer = AdamW(model.parameters(), lr=5e-4)
  - Criterion = nn.CrossEntropyLoss()
  - Batch size = 16
- All about 62% test accuracy → increase 20% compared to GAT
- Experiment 1 and 2 have similar performance
  - since benign only has 1000 graph → kind of balance
  - If we make the benign graph much more than AP(real data) → imbalance

Experiment 3

Experiment 3

- **Experiment 3:**
  - Consider the **neighbor** benign nodes
  - Edge classification

  - Given a graph → label the triplets with the benign or the specific AP



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 edges

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 from the DGL official website:
  1. Let the dgl graph's edge data have the attribute: **edata["label"]**
  2. Use **GraphSAGE** model to get the new **node embedding**
  3. Use **MLP** model to get the **score** of the edge considering the src and dest nodes
  4. Concatenate these two models
  5. Train the final model

```
g.ndata['feat'] = th.tensor(data["node_feat"])
g.edata['feat'] = th.tensor(data["edge_attr"])
g.edata['label'] = th.tensor(data["labels"])
```

```
def model_fn(batched_g, model, criterion, device, count=1, which_type='train'):
    """Forward a batch through the model."""
    batched_g = batched_g.to(device)
    labels = batched_g.edata['label'].to(device)

    logits = model(batched_g, batched_g.ndata['feat'].float())
    loss = criterion(logits, labels)

    output = torch.softmax(logits, dim=1)
    preds = output.argmax(1)

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

Experiment 3

```python
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, 'pool')
        self.layer2 = dglnn.SAGEConv(hidden_dim, out_dim, 'pool')
        self.dropout = nn.Dropout(0.25)

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

```python
class MLPPredictor(nn.Module):
    def __init__(self, out_feats, out_classes):
        super().__init__()
        self.W = nn.Linear(out_feats*2, out_classes)

    def apply_edges(self, edges):
        h_u = edges.src['h']
        h_v = edges.dst['h']
        score = self.W(torch.cat([h_u, h_v], 1))
        return {'score': score}

    def forward(self, graph, h):
        with graph.local_scope():
            graph.ndata['h'] = h
            graph.apply_edges(self.apply_edges)
            return graph.edata['score']
```

```python
class Model(nn.Module):
    def __init__(self, in_features, hidden_features, out_features, num_classes):
        super().__init__()
        self.sage = GraphSAGE(in_features, hidden_features, out_features)
        self.pred = MLPPredictor(out_features, num_classes)

    def forward(self, g, node_feat, return_logits=False):
        h = self.sage(g, node_feat)
        logits = self.pred(g, h)

        return logits
```

Experiment 3

- Total: 20 epochs
  - Optimizer = AdamW(model.parameters(), lr=5e-4)
  - Criterion = nn.CrossEntropyLoss()
  - Batch size = 16
  - Aggregate type of Graphsage: pool (but all 4 types have the similar performance in this task)

- Have pretty high accuracy at trail of using transR_50 (≈ 93%)
- At epoch 3: almost got the final validation accuracy
- Still trying to apply to all the 11 version embeddings

Experiment 3

- **Format of the true and predicted labels:**
  - 65 is benign

```
labels of Test: tensor([155,  65, 155, 155, 155], device='cuda:0') torch.Size([5])
predicted of Test: tensor([155,  65, 155, 155, 155], device='cuda:0') torch.Size([5])
labels of Test: tensor([61, 61, 61], device='cuda:0') torch.Size([3])
predicted of Test: tensor([61, 61, 61], device='cuda:0') torch.Size([3])
```

- **Classification report**:

T1564_dedfa0a54c9c13ce5714a0dc2e1f5d1a	0.00	0.00	0.00	100
T1566.001_1afaec09315ab71fdfb167175e8a019a	1.00	1.00	1.00	800
T1574.001_63bbedafba2f541552ac3579e9e3737b	1.00	1.00	1.00	6200
T1574.011_72249c1e9ffe7d8f30243d838e0791ca	1.00	1.00	1.00	600
benign	1.00	1.00	1.00	134563
accuracy			0.97	310263
macro avg	0.60	0.61	0.60	310263
weighted avg	0.97	0.97	0.97	310263

  - **Macro average** is similar to previous experiments → won't be affected by benign
  - **Weighted average** is very high since the # of the benign is high(unbalanced) and predictable
    - Except for the benign, all the AP's support is multiples of 100

Future Work

Future Work

- **GNN**
  - Finish all 11 version of embedding
  - Try to break the current limit of accuracy (60%)
  - TBD…

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

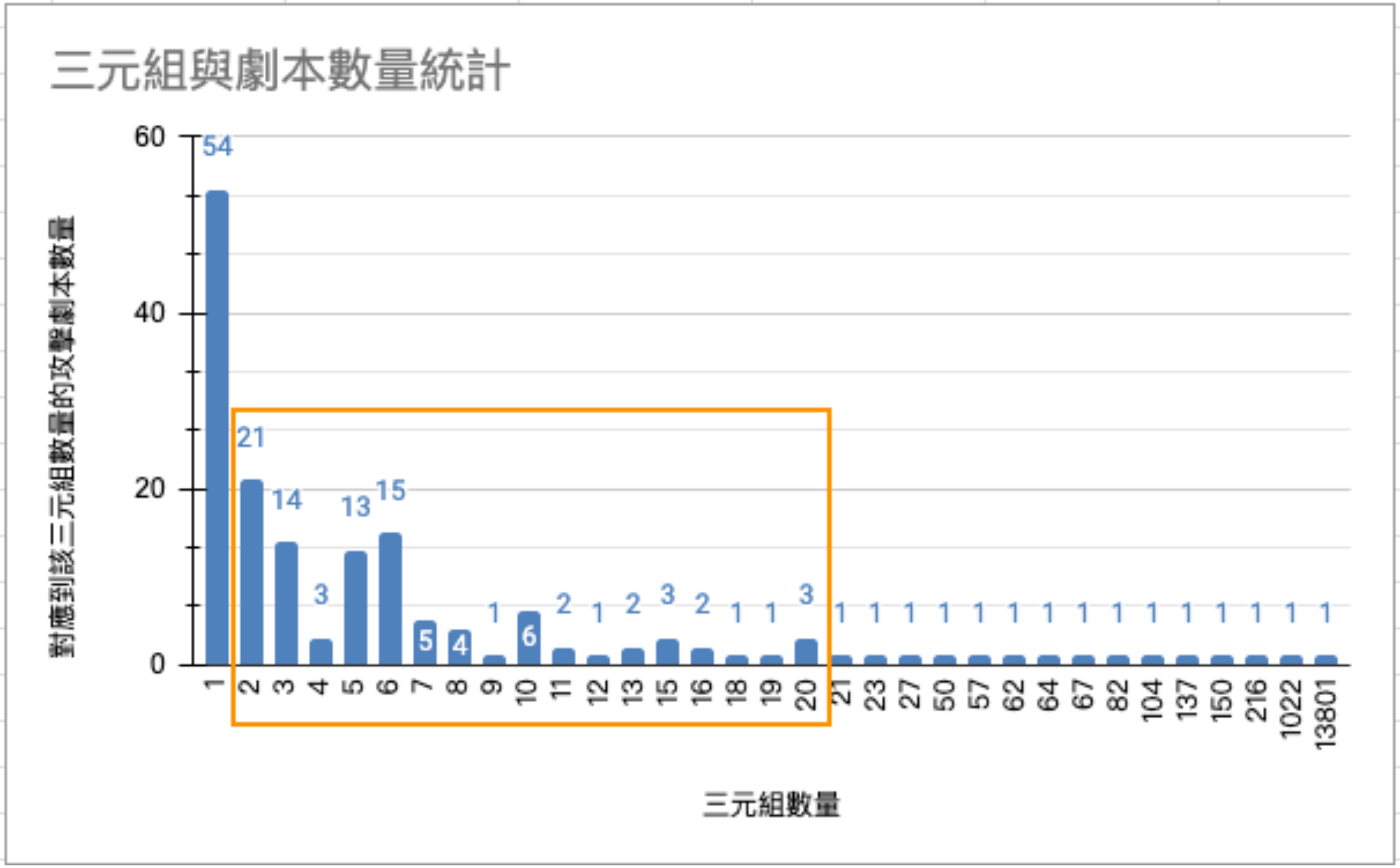
https://docs.dgl.ai/en/1.1.x/guide_cn/minibatch-edge.html#guide-cn-minibatch-edge-classification-sampler

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

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
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
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
三元組數量