



# 项目汇报

汇报人： 周昱辰

Published as a conference paper at ICLR 2021

## GRAPHCODEBERT: PRE-TRAINING CODE REPRESENTATIONS WITH DATA FLOW

Daya Guo<sup>1\*</sup>, Shuo Ren<sup>2\*</sup>, Shuai Lu<sup>3\*</sup>, Zhangyin Feng<sup>4\*</sup>, Duyu Tang<sup>5</sup>, Shujie Liu<sup>5</sup>, Long Zhou<sup>5</sup>, Nan Duan<sup>5</sup>, Alexey Svyatkovskiy<sup>6</sup>, Shengyu Fu<sup>6</sup>, Michele Tufano<sup>6</sup>, Shao Kun Deng<sup>6</sup>, Colin Clement<sup>6</sup>, Dawn Drain<sup>6</sup>, Neel Sundaresan<sup>6</sup>, Jian Yin<sup>1</sup>, Daxin Jiang<sup>7</sup>, and Ming Zhou<sup>5</sup>

<sup>1</sup>School of Computer Science and Engineering, Sun Yat-sen University.

<sup>2</sup>Beihang University, <sup>3</sup>Peking University, <sup>4</sup>Harbin Institute of Technology,

<sup>5</sup>Microsoft Research Asia, <sup>6</sup>Microsoft Devdiv, <sup>7</sup>Microsoft STCA

采用了DFG（数据流向图），DFG在一定程度上能够避免AST中一些不必要的联系从而提升了模型的有效性。

GraphCodeBert也是基于[Transformer](#)开发的

GraphCodeBert是通过三项预测任务来训练模型的：MLM（掩语言模型）、EP（数据流向图的边预测）和NA（数据流向图节点和代码token之间的关系预测）。

用于四项下游任务，包括了：代码搜索、克隆检测、代码翻译和代码改错

# 一

## 本周总结

### 用图模型ResNet18进行图像多分类

```
57 pred_list = test(test_dataloader, model, device)
58 print("pred_list = ", pred_list)
59
60 ...
61 获取文件夹列表
62 ...
63 file_name_list = []
64 data_root = r"enhance_dataset"
65 for a, b, c in os.walk(data_root):
66     if len(b) != 0:
67         print(b)
68         file_name_list = b
```

```
终端: 本地 x region-45....pro:49103 x + v
train time: 52.23601794242859
correct = 0.9191919191919192, Test Error:
Accuracy: 91.9%, Avg loss: 0.031440

Epoch 49
-----
loss: 0.001161 [ 0/ 228]
train time: 50.8171763420105
correct = 0.9191919191919192, Test Error:
Accuracy: 91.9%, Avg loss: 0.031753

Epoch 50
-----
loss: 0.000069 [ 0/ 228]
train time: 65.13531565666199
correct = 0.9191919191919192, Test Error:
Accuracy: 91.9%, Avg loss: 0.031935

(base) root@autodl-container-a160119de8-e3b56d21:~/autodl-tmp#
```

## 单张测试

The screenshot displays the PyCharm IDE interface during a single-image classification test. The left sidebar shows a file explorer with a project structure including folders like 'API21', 'APT30', and 'output', and files such as 'resnet18\_e\_best.pth' and '2.1.5图像均值和方差.py'. The main editor window shows a Python script with the following code:

```
27 return img
28
29 __name__ == '__main__':
30
31     img_path = r'enhance_sample/APT21/2badeadd8bddd305d1a9548ec599032a9d8bd25ed41357eba268aa13cb5'
32
33     val_tf = transforms.Compose([ ##简单把图片压缩了变成Tensor模式
34         transforms.Resize(512),
35         transforms.ToTensor(),
36         transform_BZ # 标准化操作
37     ])
38
39 if __name__ == '__main__': with torch.no_grad()
```

Below the code editor, the terminal window shows the execution of the script:

```
Python 控制台 × 2.2.1生成数据集 × 2.1.5图像均值和方差 × utils × 3.训练模型 × 4.1单张图片模型测试 × 4.1单张图片模型测试 (1) ×
/root/miniconda3/bin/python3 /root/.pycharm_helpers/pydev/pydevconsole.py --mode=client --host=localhost --port=34863
import sys; print('Python %s on %s' % (sys.version, sys.platform))
sys.path.extend(['/root/autodl-tmp/'])
Python 3.8.10 (default, Jun 4 2021, 15:09:15)
In [2]: runfile('/root/autodl-tmp/4.1单张图片模型测试.py', wdir='/root/autodl-tmp/')
Using cuda device
预测结果为: APT21
In [3]:
```

On the right side, the variable inspection window shows the state of variables:

- `a` = {str} 'sample/APT30'
- `b` = {list: 0} []
- `c` = {list: 113} ['00bd90d8']
- `device` = {str} 'cuda'
- `file_list` = {list: 4} ['APT10']
- `finetune_net` = {ResNet}
- `id` = {int} 2
- `img` = {Tensor: (3, 512, 512)}
- `img_path` = {str} 'enhance\_sample/APT21/2badeadd8bddd305d1a9548ec599032a9d8bd25ed41357eba268aa13cb5'
- `img_tensor` = {Tensor: (1, 3, 512, 512)}
- `result` = {Tensor: (1, 4)}
- `state_dict` = {OrderedDict}
- `transform_BZ` = {Normalize}
- `val_tf` = {Compose} Compose
- 特殊变量

## 汇编文件构建控制流图 通过学习控制流图的语义和结构特征对恶意软件分类

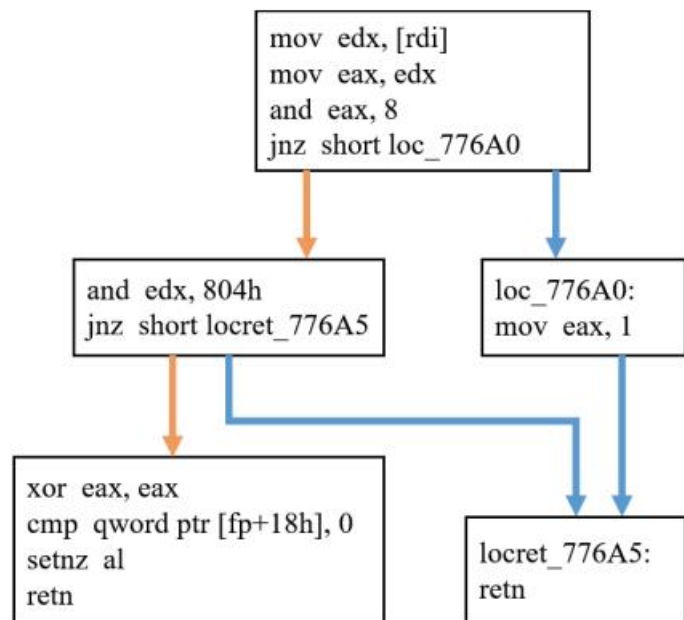
JSON Viewer

gin0jk\_4.pth 037bdc95919b1d3d65af6202e8c9c9ca3caba7a863e4e39162b93fa032881feb.json

JSON

- 268439552
- 268439564
- 268439607
- 268439570
- 268439605
  - start\_addr: 268439605
  - end\_addr: 268439605
  - insn\_list
    - in\_edge\_list
      - 0: 268439570
    - out\_edge\_list
      - 0: 268439607
- 268439611
- 268439655
- 268439681
- 268439713
- 268439876
- 268439884
- 268440300
- 268439911
- 268439922
- 268440100
- 268440170
- 268440245
- 268440253
- 268440263
- 268440271
- 268440282
- 268440304
- 268440464
- 268440472
- 268440773
- 268440499
- 268440510
- 268440688
- 268440734
- 268440749

```
7      "address": 268439552,  
8      "opcode": "mov",  
9      "operands": [  
10         "eax",  
11         "[esp+4]"  
12     ],  
13     "next_addr": 268439556  
14 },  
15 {  
16     "address": 268439556,  
17     "opcode": "push",  
18     "operands": [  
19         "esi"  
20     ],  
21     "next_addr": 268439557  
22 },  
23 {  
24     "address": 268439557,  
25     "opcode": "or",  
26     "operands": [  
27         "esi",  
28         "0FFFFFFFh"  
29     ],  
30     "next_addr": 268439560  
31 },  
32 {  
33     "address": 268439560,  
34     "opcode": "test",  
35     "operands": [  
36         "eax",  
37         "eax"  
38     ],  
39     "next_addr": 268439562
```





## 项目的任务

数据集比较小，用的数据集是APT-malware约1600条

```
log_train x best_val_acc.txt x train.py x measure.py x dataset.py x
6354 Step 4526, Epoch 69/69, iter 41/64, Loss 0.6583, Acc 0.7500.
6355 Step 4527, Epoch 69/69, iter 42/64, Loss 0.5524, Acc 0.8500.
6356 Step 4528, Epoch 69/69, iter 43/64, Loss 1.0228, Acc 0.8000.
6357 Step 4529, Epoch 69/69, iter 44/64, Loss 1.0792, Acc 0.7000.
6358 Step 4530, Epoch 69/69, iter 45/64, Loss 0.5293, Acc 0.8500.
6359 Step 4531, Epoch 69/69, iter 46/64, Loss 0.6804, Acc 0.8000.
6360 Step 4532, Epoch 69/69, iter 47/64, Loss 0.8887, Acc 0.7500.
6361 Step 4533, Epoch 69/69, iter 48/64, Loss 0.7871, Acc 0.7500.
6362 Step 4534, Epoch 69/69, iter 49/64, Loss 0.7524, Acc 0.7500.
6363 Step 4535, Epoch 69/69, iter 50/64, Loss 0.6268, Acc 0.8000.
6364 Step 4536, Epoch 69/69, iter 51/64, Loss 0.7015, Acc 0.7500.
6365 Step 4537, Epoch 69/69, iter 52/64, Loss 0.8049, Acc 0.7000.
6366 Step 4538, Epoch 69/69, iter 53/64, Loss 0.9094, Acc 0.7500.
6367 Step 4539, Epoch 69/69, iter 54/64, Loss 0.5217, Acc 0.9000.
6368 Step 4540, Epoch 69/69, iter 55/64, Loss 0.7526, Acc 0.7000.
6369 Step 4541, Epoch 69/69, iter 56/64, Loss 0.5184, Acc 0.8000.
6370 Step 4542, Epoch 69/69, iter 57/64, Loss 0.7555, Acc 0.7500.
6371 Step 4543, Epoch 69/69, iter 58/64, Loss 0.8474, Acc 0.8000.
6372 Step 4544, Epoch 69/69, iter 59/64, Loss 0.6699, Acc 0.8500.
6373 Step 4545, Epoch 69/69, iter 60/64, Loss 0.3584, Acc 0.8000.
6374 Step 4546, Epoch 69/69, iter 61/64, Loss 0.7690, Acc 0.8500.
6375 Step 4547, Epoch 69/69, iter 62/64, Loss 0.8142, Acc 0.8000.
6376 Step 4548, Epoch 69/69, iter 63/64, Loss 1.0478, Acc 0.6500.
6377 Step 4549, Epoch 69/69, iter 64/64, Loss 0.8948, Acc 0.7778.
6378 Epoch: 69
6379 Train Loss: 0.7052, Train Acc: 0.7929
6380 Val Loss: 0.7791, Val Acc: 0.7785
6381 Best val acc: 0.7846
6382
```

```
corpus_vocab.sh x dataset.py x train.py x train.sh x log_train x gnn\best_val_acc.txt x model.py x
7333 Step 7119, Epoch 69/69, iter 81/101, Loss 0.0356, Acc 1.0000.
7334 Step 7120, Epoch 69/69, iter 82/101, Loss 0.1503, Acc 1.0000.
7335 Step 7121, Epoch 69/69, iter 83/101, Loss 0.3331, Acc 0.9000.
7336 Step 7122, Epoch 69/69, iter 84/101, Loss 0.0415, Acc 1.0000.
7337 Step 7123, Epoch 69/69, iter 85/101, Loss 0.0527, Acc 1.0000.
7338 Step 7124, Epoch 69/69, iter 86/101, Loss 0.5059, Acc 0.8500.
7339 Step 7125, Epoch 69/69, iter 87/101, Loss 0.0856, Acc 1.0000.
7340 Step 7126, Epoch 69/69, iter 88/101, Loss 0.1454, Acc 1.0000.
7341 Step 7127, Epoch 69/69, iter 89/101, Loss 0.1050, Acc 1.0000.
7342 Step 7128, Epoch 69/69, iter 90/101, Loss 0.0568, Acc 1.0000.
7343 Step 7129, Epoch 69/69, iter 91/101, Loss 0.0437, Acc 1.0000.
7344 Step 7130, Epoch 69/69, iter 92/101, Loss 0.1693, Acc 0.9500.
7345 Step 7131, Epoch 69/69, iter 93/101, Loss 0.1209, Acc 1.0000.
7346 Step 7132, Epoch 69/69, iter 94/101, Loss 0.1263, Acc 1.0000.
7347 Step 7133, Epoch 69/69, iter 95/101, Loss 0.3209, Acc 0.9000.
7348 Step 7134, Epoch 69/69, iter 96/101, Loss 0.1856, Acc 0.9500.
7349 Step 7135, Epoch 69/69, iter 97/101, Loss 0.2125, Acc 0.9500.
7350 Step 7136, Epoch 69/69, iter 98/101, Loss 0.0823, Acc 1.0000.
7351 Step 7137, Epoch 69/69, iter 99/101, Loss 0.1412, Acc 0.9500.
7352 Step 7138, Epoch 69/69, iter 100/101, Loss 0.1380, Acc 0.9500.
7353 Step 7139, Epoch 69/69, iter 101/101, Loss 2.0129, Acc 0.3333.
7354 Epoch: 69
7355 Train Loss: 0.1521, Train Acc: 0.9644
7356 Val Loss: 0.3883, Val Acc: 0.9289
7357 Best val acc: 0.9308
7358
```

尝试了不同图模型，调整参数和数据集

	corpus_vocab.sh	dataset.py	train.py	train.sh
1				
2				
3				
4				

对恶意软件分类贡献最大的子图

CFGExplainer result: [malware = Ldpinch | graph = Trojan-PSW.Win32.LdPinch.bu] #nodes = 1582]

node :1  
node sub\_13149F50(320118643) info:  
start: 320118643  
prev: []  
instructions:  
320118643, ['test', 'esi,3']  
320118649, ['jz', 'shortloc\_13149F8E']  
next: [320118651, 320118670]

node :2  
node sub\_13149F50(320118651) info:  
start: 320118651  
prev: []  
instructions:  
320118651, ['xor', 'al,[esi]']  
320118653, ['inc', 'esi']  
320118654, ['mov', 'ebx,0FFh']  
320118659, ['and', 'ebx,eax']  
320118661, ['shr', 'eax,8']  
320118664, ['xor', 'eax,[edi+ebx\*4]']  
320118667, ['dec', 'ecx']  
320118668, ['jnz', 'shortloc\_13149F73']  
next: [320118643, 320118670]

node :3  
node sub\_13149F50(320118684) info:  
start: 320118684  
prev: []  
instructions:  
320118684, ['xor', 'eax,[esi]']  
320118686, ['add', 'esi,4']  
320118689, ['mov', 'ebx,0FFh']