malware项目代码文档

malware项目是通过二进制文件流，判别一个软件是否为恶意软件，该文档的研究方向是致力于使用convolution neural network(cnn)的模型解决这个问题，而cnn的处理数据需满足structure data，并且关注于局部信息，所以在理论上cnn是适合于这个问题.

数据处理

我们通过将二进制流每 8 位转化为一个 8 bit数据，范围为[0,255],然后将其整合为二维图像数据．

所在目录：

```
malware_2017_10/data_process/bytes
```

抽取二进制数据

给定训练集(train.csv)和测试集(test.csv),代码为copy_file.py.根据两个csv文件中的malware ID去指定的文件夹中搜索文件，然后储存到文件中，文件的目录结构为：

```
.
├── test
│   ├── malware
│   └── normal
├── train
│   ├── malware
│   └── normal
```

代码运行的格式:

```
"""
positional arguments:
  source_root   path of source file
  des_root      path of store destination

optional arguments:
  -h, --help    show this help message and exit
  --train, -t   bool value,represent whether train or test.
  --server, -s  bool value,run in server or local
  """
python copy_file.py /macml-data/ /home/lili/Tao_Zhang/malware_2017_9_7/train --server --train
```

将二进制数据转化为图片

将抽取出来所有的二进制文件都转化为图片

utils.py

```
"""
将一个文件转化为图片
args:
    source_path:源文件的地址
    des_path:图片的储存地址
    size:图片的大小
"""
def transform_to_img(source_path,des_path,size):
```

transfer_img.py

将train和test文件夹中的所有文件都转化为图片，数据的储存目录格式为:

img

```
├── test
│   ├── malware
│   └── normal
├── train
│   ├── malware
│   └── normal
```

```
"""
positional arguments:
  source_dir   path of source file
  des_dir      path of store image file

optional arguments:
  -h, --help   show this help message and exit
  --size SIZE  size of image
"""
python transfer_img.py /home/lili/Tao_Zhang/malware_2017_9_7 /home/lili/Tao_Zhang/malware_2017_9_7/img
```

上述就是所有的数据处理的代码，所有的文件都储存在img文件，img文件夹的格式就如上述所示．

生成验证集

从train set中生成验证集

```
python generate_val_from_train.py
```

注：最终的数据集有两个版本，分别是bytes和opcode.

代码文件说明

数据

数据位于服务器上,没有下载到本地
server:10.206.139.110
passwd:

```
#登录服务器
ssh lili@10.206.139.110
#路径
/home/lili/Tao_Zhang/dataset/img (二进制文件转化为image之后的结果)
/home/lili/Tao_Zhang/dataset/opcode(opcode文件转化为image后的结果)
#该目录下的其他文件为原始数据
#处理数据的文件位于
/home/lili/Tao_Zhang/program/git/malware_2017_10/data_process下的文件.
```

模型构建

测试系统环境：ubuntu14.04LTS
测试代码环境：pytorch0.2.0+GPU(需要设置config文件中的slef.cuda=True,否则默认使用cpu训练)

所在目录为:**malware_2017_10/malware**
目录结构：

```
├── config.py
├── data
│   ├── dataset.py
│   ├── __init__.py
├── main.py
├── logs
├── models
│   ├── BasicModel.py
```

```
│   ├── init.py
│   └── SPL
├── README.md
├── test.ipynb
├── pkls
└── visualize
```

config.py

该文件为模型的配置文件，其中涉及到超参的取值，包括learning-rate等，其中与self-paced learning相关的参数并没有在base-line的模型中使用到.

```
#输出一些调试信息
self.debug = False

#system configuration
##设置数据的根目录
self.root_path = "/home/lili/Tao_Zhang/dataset/opcode"
##设置图片的大小
self.size = 128

#net configuration
##设置learning rate
self.lr = 0.001
##设置batch size
self.batch_size = 512
##设置是train还是test
self.istraining = False
##是否使用GPU,cuda or not
self.cuda = True
##whether read all data nor read batch size data
##(如果内存不够，可以按照batch_size的大小进行读取数据)
self.read_all_data = True
##是否改变数据的比例
self.change_data_proportion = False

##self pace learning
##whether use spl or not
self.spl = False
self.K = 10
self.update_threshold = 0.4

##epoch的数量
self.epoch_num = 400
##是否使用batch normalization
self.bn=True
##保存model为pkl_name
self.pkl_name = "pkls/img_20171130_opcode_init.pkl"
##是否需要可视化
self.visualize = True
##visualize train loss
self.train_loss_env = "opcode_train20171130"
self.train_loss_win = None
##visualize val precison and recall
self.val_env = "opcode_val20171130"
self.val_win = None
self.val_recall_win=None
```

models

pytorch构建的cnn模型

```
/home/lrh/program/git/malware_2017_10/malware/models/BasicModel.py
```

data

处理数据的代码，将img目录下的图像文件，整合读入到一起输入到模型中

```
/home/lrh/program/git/malware_2017_10/malware/data/dataset.py
```

main.py

主文件，

```
python main.py#就可以运行
```

logs

运行代码产生的输出,在运行命令的时候设置

pkls

模型储存在pkls中方便测试,在配置文件中设置

visualize

使用facebook提供的visdom可视化

运行命令格式

```
CUDA_VISIBLE_DEVICES=0 nohup python -u main.py >logs/mnist_capsule_20171127.log 2>&1 &
```

项目github地址：(https://github.com/selous123/malware_2017_10)

福利：因为服务器上的visdom服务一直都开启着,所以在浏览器端可以访问到该项目的运行过程，项目分别可视化了train loss 和validation accuracy 地址为：
http://10.206.139.110:8097/