使用TextCNN模型探究恶意软件检测问题（安全客首发）

笔记本:	日常		
创建时间:	2019/11/17 17:21	**更新时间:**	2019/11/28 21:28
作者:	296645429@qq.com		
URL:	https://www.anquanke.com/contribute/new		

前言

本文以阿里云恶意软件数据集为基础，探究了在工业界背景下使用单模型TextCNN进行恶意软件检测的新方法，获得了很好的结果。

背景

在信息化时代，电子产品上存在很多恶意软件。例如很多用户使用存在后门的破解软件，当他们在操作系统上运行软件，破解软件会默默地获取系统信息并发送给攻击者，这样即泄露了用户的隐私。恶意软件的恶意攻击行为很多，电子产品企业如能使得产品具有良好的恶意软件检测性能，即能更好地保障用户的安全。

近几年，随着深度学习的高速发展，人们逐渐使用它自动学习恶意软件特征，从而识别当前应用是否是恶意软件。本文基于深度学习的知识和工业的单模型需求，使用了TextCNN模型探究恶意软件检测问题。

相关配置信息

数据源

阿里云提供了丰富的恶意软件运行后的数据集。在训练集表格数据中，存在四列数据：file_id、label、api、tid、index，分别表示文件编号、文件恶意行为类别（8类，分别是0-正常/1-勒索病毒/2-挖矿程序/3-DDoS木马/4-蠕虫病毒/5-感染型病毒/6-后门程序/7-木马程序）、文件调用的API、调用API的所属线程号、单个线程调用的API集的顺序号。
参考数据：

https://tianchi.aliyun.com/competition/entrance/231694/information

评估

在阿里云的测试集数据表格中，存在三列数据：file_id、api、tid、index。依据它们预测出label信息的所属类别概率分布，并将这列概率分布保存到一个结果数据表格。
结果数据表格中，存在九列数据：file_id，prob0, prob1, prob2, prob3, prob4, prob5 ,prob6,prob7，分别表示测试集数据中的文件编号，后八列表示预测此文件分别是正常、勒索病毒、挖矿程序、DDoS木马、蠕虫病毒、感染型病毒、后门程序、木马程序的概率，概率和为1。

将结果数据表格提交到阿里云平台即可获得测试结果。

平台

本文使用的是Google Colab平台的GPU和TPU。TPU使用方法会在下面使用时详细说明。

Tensorflow版本：1.13.1，Keras:2.2.5

恶意软件检测

在这一部分，本文首先简述TextCNN的架构等基础内容，然后根据数据的分析情况和检测的结果分层次地对模型进行改进。

TextCNN等基础内容

Tokenizer类的函数和变量

函数fit_on_text(texts)：使用一系列文档来生成token词典，texts为list类，每个元素为一个文档。
函数texts_to_sequences(texts)：将多个文档转换为word下标的向量形式

变量word_index：一个dict，保存所有word对应的编号id，从1开始

Embedding

将词的十进制表示做向量化

SpatialDropout1D

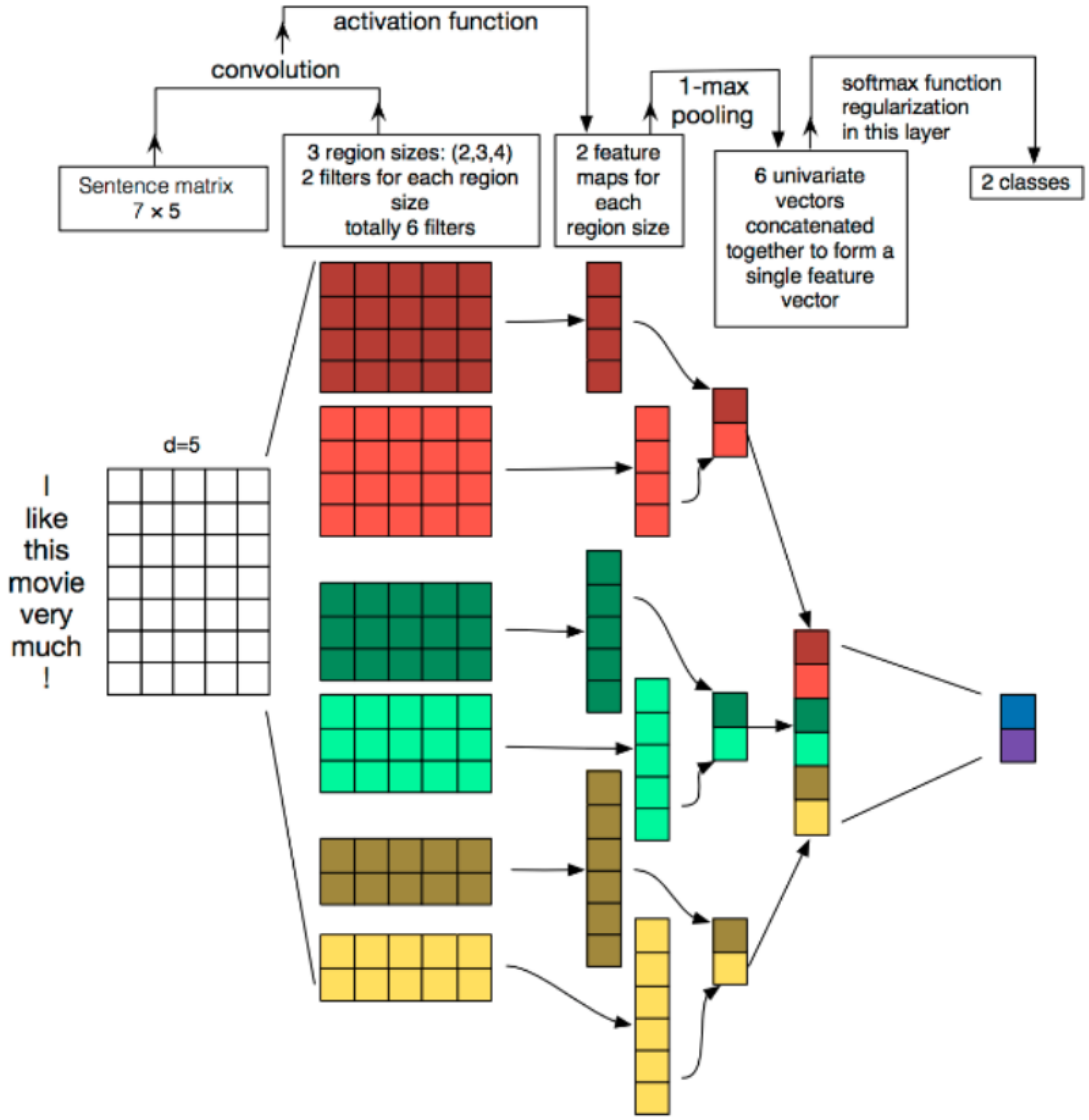
在模型开始应用，会按一定比例丢弃一个特征图中的多个通道信息，增强特征的独立性，这和dropout是不同的。
如下图。

`Dropout()`：让我们定义2D输入：[[1,1,1]，[2,2,2]]。Dropout将独立考虑每个元素，并可能导致类似[[1,0,1]，[0,2,2]]的内容

`SpatialDropout1D()`：在这种情况下，结果将类似于[[1,0,1]，[2,0,2]]。请注意，第二个元素沿**所有**通道归零。

TextCNN

可参考论文《Convolutional Neural Networks for Sentence Classification》。TextCNN通过卷积核能充分获取到句子内部的逻辑相关性，具有很大的优势。



由上图可知，以此句子为例，首先按词数分成七部分，再将每部分的词通过Embedding的方法扩展为五维向量，即最终生成7x5的句子矩阵。再使用不同的卷积核对其进行卷积运算求取特征图。然后通过池化层池化所有特征图，最后通过全连接层汇聚特征为特征向量，从而用于分类，这里的分类数量为2。

模型实战

数据处理

首先将表格数据做处理存放至pkl文件，方便以后直接使用。
处理过程中，
本人将训练集表格中数据按照file_id分组，并在每组file_id中，依次按照tid、index对数据排序，将排序后的API作为此文件的核心数据，将label作为此文件所属的种类。测试集数据除无label，其他数据处理方式同上。
代码如下：

```
import pandas as pd
import pickle
import numpy as np
```

```python
def LoadTrains():
    # 训练集
    train_path = r'./security_train1.csv'
    labels=[]
    apis=[]
    data = pd.read_csv(train_path)
    # 分组成文件集
    files = data.groupby('file_id')
    for file_id, files_info in files:
        newfile_info = files_info.sort_values(['tid', 'index'])
    #     print(newfile_info)
        api_sequence = ' '.join(newfile_info['api'])
        # print(len(api_sequence))
    #     print(files_info['label'].values[0])
        labels.append(files_info['label'].values[0])
        apis.append(api_sequence)
    # 格式[5,6,...]
    # print(fileId)
    # 格式['getaddr loadexe loaddll','...',...]
#     print(apis)
    with open('mytpu_security_train1.pkl', 'wb') as f:
        pickle.dump(labels, f)
        pickle.dump(apis, f)
#     print(apis)
    return labels, apis
```



```python
def LoadTests():
    # c测试集
    test_path = r'./security_test1.csv'
    apis=[]
    test_fileId = []
    data = pd.read_csv(test_path)
    # 分组成文件集
    files = data.groupby('file_id')
    for file_id, files_info in files:
        newfile_info = files_info.sort_values(['tid', 'index'])
    #     print(newfile_info)
    #     print(file_id)
        api_sequence = ' '.join(newfile_info['api'])
        test_fileId.append(file_id)
        apis.append(api_sequence)
    # 格式['getaddr loadexe loaddll','...',...]
#     print(apis)
    with open('mytpu_security_test1.pkl', 'wb') as f:
        pickle.dump(test_fileId, f)
        pickle.dump(apis, f)
    return test_fileId , apis
```

```
if __name__ == '__main__':
    train_labels = []
    train_apis = []
    test_files = []
    test_apis =[]
    train_labels, train_apis = LoadTrains()
#     print(train_labels)
#     print(train_apis)
    test_files, test_apis = LoadTests()
```


由上图可知生成了训练集和测试集的对应pkl文件。

模型训练及预测

首先使用Tokenizer根据已有API数据生成字典，将API数据转换为数字表示的形式。代码如下：

```
tokenizer = Tokenizer(num_words=None,
                      filters='!"#$%&()*+,-./:;<=>?@[\]^_`{|}~\t\n',
                      lower=True,
                      split=" ",
                      char_level=False)
# print(train_apis)
# 通过训练和测试数据集丰富取词器的字典，方便后续操作
tokenizer.fit_on_texts(train_apis)
...
tokenizer.fit_on_texts(test_apis)
...
train_apis = tokenizer.texts_to_sequences(train_apis)
# 通过字典信息将字符转换为对应的数字
test_apis = tokenizer.texts_to_sequences(test_apis)
```


因为数据量巨大，且每个文件的API数据量都不一致，为了方便TextCNN的运行，使用pad_sequences截取inputLen长度的词。inputLen的长短直接决定了句子的丰富程度，在后续探究工作中可做改进。

```
# 序列化原数组为没有逗号的数组，默认在前面填充,默认截断前面的
train_apis = pad_sequences(train_apis, inputLen, padding='post', truncating='post')
# print(test_apis)
test_apis = pad_sequences(test_apis, inputLen, padding='post', truncating='post')
```


规范好数据后，即可调用TextCNN模型，初次采用的结构如下：

```python
def textcnn():
    kernel_size = [1, 3, 3, 5, 5]
    acti = 'relu'
    # 可看做一个文件的api集为一句话，然后话中的词总量是6000
    my_input = Input(shape=(inputLen,), dtype='int32')
    emb = Embedding(len(tokenizer.word_index) + 1, 20, input_length=inputLen)(my_input)
    emb = SpatialDropout1D(0.2)(emb)
    net = []
    for kernel in kernel_size:
        # 32个卷积核
        con = Conv1D(32, kernel, activation=acti, padding="same")(emb)
        # 滑动窗口大小是2,默认输出最后一维是通道数
        con = MaxPool1D(2)(con)
        net.append(con)
    # print(net)
    # input()
    net = concatenate(net, axis=-1)
    # net = concatenate(net)
    # print(net)
    # input()
    net = Flatten()(net)
    net = Dropout(0.5)(net)
    net = Dense(256, activation='relu')(net)
    net = Dropout(0.5)(net)
    net = Dense(8, activation='softmax')(net)
    model = Model(inputs=my_input, outputs=net)
    return model
```


根据上图可知，设置词向量的维度为20，使用了5个大小分别为1、3、3、5、5的卷积核，使用了基础的最大池化操作。这三部分设置关系了模型的性能，在后续探究工作中可做改进。
在训练及测试过程中，本文使用了5折交叉验证法，有效降低过拟合情况的产生。且结合使用早停机制和选择最优模型机制保存最好的训练结果。最后预测并生成结果数据表。
这部分代码链接：

https://github.com/AI-Winner/Wang-Net-TextCNN/blob/master/my_net.py

模型探究

模型改进1

使用GPU做模型的API信息探究。首先确定较合适的每个文件中API的个数inputLen，然后确定每个文件中每个API的词维度即textcnn网络中Embedding函数的第二个参数output_dim。
查看原始文件信息，可知文件的API个数最大为13264587。考虑GPU内存和速度，依次设置inputLen为100、5000、7000，联合设置output_dim为5、20。
将训练好的模型预测出的结果提交至官网查看结果，如下图。

logloss	valid_acc	validloss	备注
3.225881	0.8303	0.4915	api序列长100，词向量维度是5
1.347444	0.9679	0.105	api序列长5000，词向量维度是5
1.593029	0.9802	0.0689	api序列长7000，词向量维度是5
1.514191	0.9859	0.0516	api序列长7000，词向量维度是20
1.065223	0.78474	0.67778	api序列长5000，词向量维度是20

分析可知，API的个数inputLen为5000、词维度output_dim为20时结果较好，output_dim还有增大的潜力。

模型改进2

做早停方法探究。在早停机制中，依次设置训练过程中损失上升次数即EarlyStopping函数的patience为20、25。将训练好的模型预测出的结果提交至官网查看结果，如下图。

logloss	valid_acc	validloss	备注
0.92848	0.8129	0.7146	api序列长5000，词向量维度是20。早停机制限制20代。
0.837264	0.8252	0.6868	api序列长5000，词向量维度是20。早停机制限制25代。

可知，patience为25时logloss为0.837264，经过改进logloss降低了2个百分点。

模型改进3

使用TPU做数据均衡、权重正则化、网络池化探究。

数据均衡探究。

针对本题，训练集和测试集不一样，应采取措施使得训练集和测试集中的正样本比例相差较小，这样训练出的模型对测试集具备更好的泛化能力。我们使用Keras中的fit函数的class_weight解决这个问题，如下。
fit(self, x=None, y=None, batch_size=None, epochs=1, verbose=1, callbacks=None, validation_split=0.0,validation_data=None,shuffle=True,class_weight=None,sample_weight=None,initial_epoch=0,steps_per_epoch=None,validation_steps=None)
class_weight：参数含义：字典，将不同的类别映射为不同的权值，该参数用来在训练过程中调整损失函数（只能用于训练）。该参数在处理非平衡的训练数据（某些类的训练样本数很少）时，可以使得损失函数对样本数不足的数据更加关注。假设有500个0级样本和1500个1级样本，未解决样本不均衡问题，应该设置class_weight = {0：3，1：1}，这给了0级三倍于1级的权重。
可知官方计算logloss的方法如下。

$$\log loss = -\frac{1}{N}\sum_{i}^{N}\sum_{j}^{M}\left[y_{ij}\log(P_{ij}) + \left(1-y_{ij}\right)\log\left(1-P_{ij}\right)\right]$$

M代表分类数，N代表测试集样本数，yij代表第i个样本是否为类别j(是~1，否~0)，Pij代表选手提交的第i个样本被预测为类别j的概率(prob)，最终公布的logloss保留小数点后6位。
为了得到测试集中各类别的样本数，则模拟一份测试结果提交官网。在测试结果表中，存在8个概率列表示当前文件所属各类恶意文件的概率，联合过渡平滑的思想，统一将一列设置为0.3表示文件的可能类别，其他列设置为0.1。此时，根据logloss计算方式，应用于此8分类问题中，则单个文件预测正确时，logloss计算方式为

$$-(log0.3 + 7log0.9)$$

单个样本预测错误时，logloss计算方式为

$$-(log0.1 + log0.7 + 6log0.9)$$

则当前，在测试结果表中，将8个概率列中一列设为0.3表明所有文件均可能是此类恶意文件，设置真实属于此类恶意文件的文件数为n，测试集样本总数为N，总logloss计算方式应为

$$logloss = \frac{\mathrm{n}[-(log0.3 + 7log0.9)] + (\mathrm{N}-\mathrm{n})[-(log0.1 + log0.7 + 6log0.9)]}{\mathrm{N}}$$

换算可得。

$$n = \frac{\mathrm{N}\ \mathrm{logloss} - \mathrm{N}[-(log0.1 + log0.7 + 6log0.9)]}{-(log0.3 + 7log0.9) + (log0.1 + log0.7 + 6log0.9)}$$

由此，N已知，logloss通过官网对此测试表的计算结果可得到，即顺利得到属于此类恶意文件的样本数。依次可得到测试集所有类恶意样本的样本数。
可知测试集各类别样本数：0: 4978 1: 409 2: 643 3: 670 4: 122 5: 4288 6: 629 7: 1216
已知训练集各类别样本数：0: 4978 1: 502 2: 1196 3: 820 4: 100 5: 4289 6: 515 7: 1487
本文可计算class_weight为：
0:1.00,1:0.81,2:0.54,3:0.82,4:1.22,5:1.00,6:1.22,7:0.82。程序设置方法如下。

```
#训练的过程会保存模型并早停
model.fit(train_apis[train_index[:len_train_index]], train_labels[train_index[:len_train_index]], epochs=Epoch, batch_size=batchsize,
                    validation_data=(train_apis[valid_index[:len_valid_index]], train_labels[valid_index[:len_valid_index]]),
                    callbacks=[checkpoint, earlystop], class_weight={0:1.00,1:0.81,2:0.54,3:0.82,4:1.22,5:1.00,6:1.22,7:0.82})
```

权重正则化探究

有三种正则化方法：
L1：绝对值权重之和
L2：平方权重之和
L1L2:两者累加之和
实现方法分别是：
tf.keras.regularizers.l1(l=0.01)
tf.keras.regularizers.l2(l=0.01)
tf.keras.regularizers.l1_l2(l1=0.01,l2=0.01)
卷积层、全连接层使用权重正则化的方法分别是：
tf.keras.layers.Conv2D(32,3,activation=tf.nn.relu,kernel_regularizer=tf.keras.regularizers.l2(l=0.0005),bias_regularizer=tf.keras.regularizers.l2(l=0.0005))
tf.keras.layers.Dense(512,activation=tf.nn.relu,kernel_regularizer=tf.keras.regularizers.l2(l=0.001),bias_regularizer=tf.keras.regularizers.l2(l=0.001))
本文对于textcnn中卷积和全连接层的正则化如下：

```
con = Conv1D(8, kernel, activation=acti, padding="valid", kernel_regularizer=l2(0.0005))(emb)
```

```
net = Dense(256, activation='relu', kernel_regularizer=l2(l=0.001))(net)
```

TPU基础

在TPU中，典型的批量大小是1024。每个TPU 都由 8 个 TPU 核心组成，每个核心都有 8GB 的 RAM（或 HBM，即高带宽内存）。
在使用TPU训练模型时，每次输入的样本数应保持是8的倍数，使得每个TPU核心运行同样数量的样本，这样会不可避免地删除不能被8整除的一小部分样本。则在测试时，一个好的方法是在CPU上进行预测。
在本实验中，设置batchsize为8的倍数，此时训练集和验证集的数据量应符合如下代码：

```
#  设置使得样本量和batchsize（8的倍数）匹配，适应TPU
len_train_index = len(train_index)-(len(train_index)-int(len(train_index)/batchsize)*batchsize)
#  train_index.astype(int)
len_valid_index = len(valid_index)-(len(valid_index)-int(len(valid_index)/batchsize)*batchsize)
```

池化探究

TextCNN中有一种池化层是时序最大池化，它可被认为是一维全局最大池化，可使模型不受人为手工为数据补0的影响。
全局池化就是pooling的 滑窗size 和整张feature map的size一样大。这样，每个 W×H×C 的feature map输入就会被转化为 1×1×C 输出。因此，其实也等同于每个位置权重都为 1/(W×H)1/(W×H) 的FC层操作。

本实验修改TextCNN中一般池化为最大池化和最大平均池化。

```
con = Conv1D(8, kernel, activation=acti, padding="valid", kernel_regularizer=l2(0.0005))(emb)
# 默认输出最后一维是通道数
# con1 = MaxPool1D(2)(con)
con1 = GlobalAveragePooling1D()(con)
con2 = GlobalMaxPooling1D()(con)
net.append(con1)
net.append(con2)
```


实验结果

由于TPU内存更大性能更好的原因，通过更多API信息、更多卷积核提取更多不同的特征如下。具体操作是设置API数量inputLen为20000，API词维度output_dim为128，网络中增加大小分别为1、3、5、7、9、11、13的卷积核。
结果如下：

logloss	valid_acc	validloss	备注
0.555824		loss:0.43	网络结构复杂化，主要添加了全局池化层

经过改进，单模型logloss为0.555824，又降低了3个百分点，效果显著。
代码链接：

https://github.com/HYWZ36/my_aliyun_ML_Malware_detect/tree/master/Code