# 网络流量分析

————————

## 背景

加密流量分析是计算机网络研究的一个前沿方向，该研究有多个应用场景，如Tor匿名网络中进行网站指纹、移动端加密流量分析、视频流量分析和QoE测量等。其中，网页指纹（WF）是一种流量分析攻击，可被恶意攻击者用来识别用户正在访问的网页，侵犯用户隐私。网页指纹攻击可以通过包序列信息来识别出一个用户的网络活动，尽管隐私技术也越来越成熟，比如Tor，是一种匿名网络，目前有着大约50万的日活跃用户，用户数量还在增加，然而，Tor的隐私保护机制依然遭受着网页指纹技术的威胁，除此之外，像其他的隐私技术如SSH隧道技术、VPNs和IPsec也面临着相同的挑战。

当一个客户端应用接入互联网的时候，计算机会将目的地址和其他信息打包成一个数据包，发送给中间路由节点，攻击者可能在路由节点截获用户的数据包，并分析出有价值的信息，为了保护用户的隐私，客户端应用需要对传输的数据进行加密，或者选择使用类似于Tor的匿名性比较好的网络进行传输。网站指纹技术是攻击者使用的针对网络流量的攻击技术，该技术通常利用网络流量包含的可以识别出网页的数据包的长度、顺序和时间信息等，通过观察流经网络的流量获取到用户想要访问的的目的网页，机器学习算法经常被用来对网络流量进行分析。

对于一个数据包序列，网站指纹可以使用分类算法从中识别到要访问的网页。网站指纹指纹的攻击过程可以概括成以下流程：攻击者首先收集感兴趣的已知网页的数据包序列，可以为算法的中的监督训练过程提供训练样本和样本对应的标签，算法最终会从训练样本中学得一个分类模型，攻击者会将这个分类模型应用到新收集的数据包序列上，达到识别目的网页信息的目的。

## 分析过程与方法

网络流量分析的步骤如下：

1. 从网络上抓取50个网页的连接数据。
2. 对数据集进行数据探索分析和预处理。
3. 为处理好的数据设计分类方案，挑选合适的模型。

## 数据预处理



表-1 数据格式

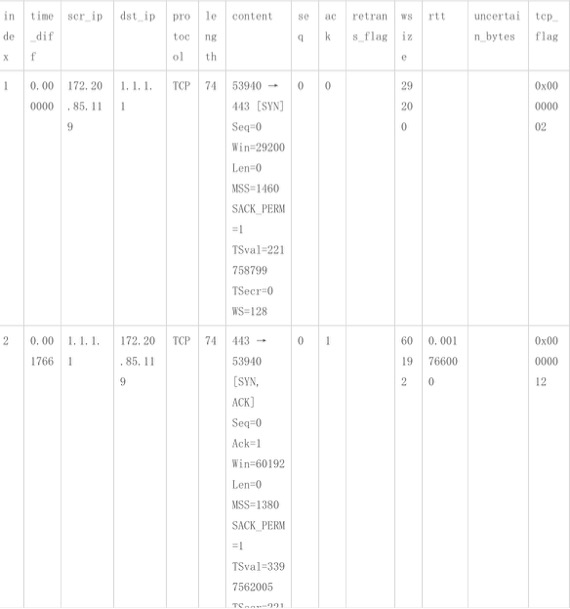


表-2 样本示例

数据预处理是数据挖掘分析中非常重要的一步，由于海量数据中存在着大量不完整（有缺失值）、不一致、有异常的数据，严重影响到了对数据分析的效率，所以在处理数据之前进行数据清理、集成、转换和规约等数据预处理操作显得尤为重要。

1. 数据规约

数据规约可以产生更小但保持原数据完整性的新数据集，在规约后的数据集上进行分析将更有效率。本次实验将对数据集中的src*ip*和*dst*ip进行规约，为每个数据包添加新的属性direction，该属性可以由原数据集中的两个属性推理出来，表示数据包的方向，1代表客户端ip向公网ip发送数据包，0表示公网ip像客户端ip发送数据包。

data[‘direction’] = data[‘dst\_ip’].map(lambda ip: 1 if ip == ‘1.1.1.1’ else 0)

1. 数据变换

数据变换主要是对数据进行规范化处理，是为了算法的需要或者获取更有价值的数据，来将数据中的某些特征值转换为我们可以处理的形式，本实验中主要采用的数据变换方式为简单的文本到数值的变换， 我们将协议这个属性对应的文本值用一个数值来代替，以满足算法需求。

#用one-hot编码protocol，tcp\_flag

data[‘protocol’] = data[‘protocol’].map({

‘TCP’: 0,

‘TLSv1’: 1,

‘TLSv1.3’: 2,

‘SSLv2’: 3

})

data[‘tcp\_flag’] = data[‘tcp\_flag’].map({

‘0x00000002’: 0,

‘0x00000012’: 1,

‘0x00000010’: 2,

‘0x00000018’: 3,

‘0x00000011’: 4,

‘0x00000004’: 5,

})

1. 数据清洗

数据清洗主要是删除原始数据集中的无关数据、重复数据，处理缺失值、异常值等。本次实验所使用的数据集中，rtt和uncertain*bytes*两个属性存在缺失值的情况，我们将对这两个属性进行缺失值处理，对有缺失值的样本填充*0*，最后删除掉对训练模型没有价值的*scr*ip、dst*ip*、*content*和大部分都是空值的*returns*flag属性。

#使用pandas库进行处理

#填充0

data.fillna(0, inplace=True)

#消除属性scr\_ip, dst\_ip, content,returns\_flag

removed\_features = [‘scr\_ip’, ‘dst\_ip’, ‘content’, ‘returns\_flag’]

data.drop(removed\_features, axis=1, inplace=True)

## 方案设计

在分类模型的方案设计上，我们尝试了机器学习和深度学习两个方案，总的来说，这两个方案的差异在于对待数据的视角不同，处理数据的单位不同，我们知道，在对一个网站的访问中，会产生许多客户端和网站之间的连接，也就是数据包数据，单一的数据包对网站的识别所做的贡献比较少，如果将连续的数据包综合考虑，那么识别网站的信息量有所增加。机器学习方案中，没有考虑到数据之间的相关性，将所有数据包数据聚合到一个文件中，每一行表示一条数据包数据，并在每行的末尾打上所属网站的标签；深度学习方案中，将与某网站的某次连接所产生的所有数据包看作一个单位，放到一个文件里进行处理，然后为该文件打上相应网站的标签。我们尝试的跑出结果的机器学习算法有KNN、CART、朴素贝叶斯，其中CART结果比较好，另外SVM和AdaBoost因为资源有限，未能得到结果；深度学习算法尝试了CNN和LSTM，其中LSTM效果比较好。

### 机器学习方案

1. 数据处理

该步骤主要为每条数据包数据添加所属网站的标签，处理好的数据如图所示，共1366828条数据。



表-3 预处理后的数据

1. CART算法

决策树简介

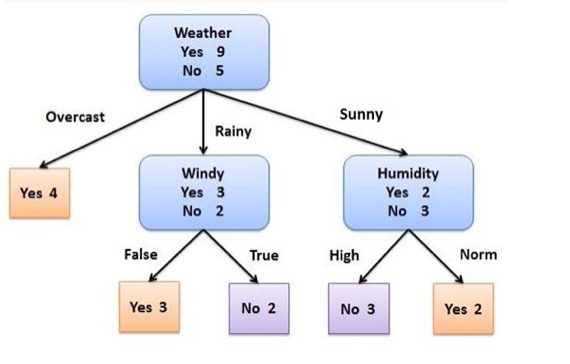


图-1 决策树示意图

决策树是一种由节点和有向边组成的层次结构，如上图所示，决策树包含三种节点：

* + 根节点：没有入边，有零条或者多条出边
  + 内部节点：有一条入边和两条或多条出边
  + 叶节点：只有一条入边，没有出边

在决策树中，每个叶节点都有一个类标号，非叶子节点包含属性测试条件，用于分开具有不同特性的记录。决策树算法的生成过程包括：树构造、树剪枝。

* + 树构造：

决策树采用自顶向下的递归方式从根节点开始在每个节点上按照给定标准选择属性测试条件，然后按照相应属性的所有可能取值向下建立分支，划分训练样本，直到一个节点上的所有样本都被划分到同一个类，或者某一个节点中的样本数量低于给定值时为止。

* + 树剪枝：

构造完成的决策树中的分支还会受到数据中的噪声的干扰，该步骤主要检测和去除受到噪声影响的分枝。

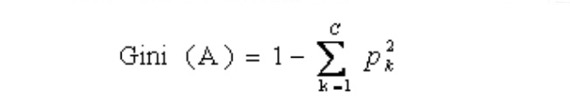
决策树的使用过程是将未分类的样本的属性与决策树节点上的属性比较的过程。

决策树优点

* 非参数方法，不要求任何先验假设，不假定类和其他属性服从一定的概率分布
* 训练需要的时间少
* 对噪声的干扰具有较好的鲁棒性

CART决策树算法

分类回归树CART是一种典型的二叉决策树，可以同时处理连续变量和离散变量。CART使用的分裂准则是Gini系数，如果集合T包含C个类，节点A的Gini系数为：



其中Pk表示样本属于k类的概率。当Gini=0时节点中的所有样本属于同一类，当所有类在节点中以相同的概率出现时，Gini值达到最大，当前属性的最优分裂点是使Gini系数最小的值。

实验及结果

# 导入数据，划分训练集和测试集

data = pd.read\_csv(out\_dir + 'data\_single.csv', skipinitialspace=True)

X = data[data.columns.difference(['label'])]

y = data['label']

# target\_names = data.columns

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

#训练模型

clf = tree.DecisionTreeClassifier(criterion='gini')

clf.fit(X\_train, Y\_train)

y\_pred = clf.predict(X\_test)

print(metrics.classification\_report(Y\_test, y\_pred))

训练集准确率和测试集上的准确率：



图-2 训练集准确率



图-3 测试集准确率

### 深度学习方案

LSTM（长短期记忆网络）

LSTM是RNN的一种变体，RNN处理序列类型的数据，当计算当前时间的数据时会考虑之前时刻的数据，所以RNN对数据具有某种程度的记忆，然而RNN会遇到梯度消失或者梯度爆炸的问题，导致RNN不能处理更长的序列。

LSTM在RNN的基础上引入了门机制，既可以保留RNN的短期记忆，也可以保留长期记忆，有助于处理自然语言或者时间序列任务。

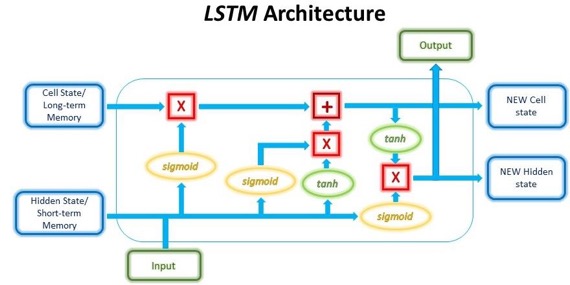


图-4 LSTM架构图

LSTM网络中存在三种类型的门，分别是输入门、遗忘门和输出门。

* + 输入门

输入门的主要功能是决定可以存储在长期记忆中的新信息，该门的输入是来自上一时间步骤形成的短期记忆和当前时间步骤的输入。该门由两层组成，第一层由当前输入和短期记忆经由sigmoid生成表示信息重要性的向量，0表示不重要，1表示信息会被使用；第二层由当前输入和短期记忆经由激活函数（tanh）产生新信息。两层数据相乘产生可以存储在长期记忆中的信息。

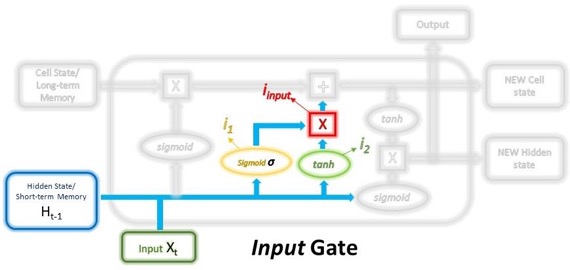


图-5 输入门

* + 遗忘门

遗忘门的主要功能是决定长期记忆中的哪些信息可以被保留下来，该门的输入为当前输入、长期记忆和短期记忆，来剔除长期记忆中的一些信息。具体是当前输入和短期记忆会通过sigmoid函数生成一个0和1组成的遗忘向量，该向量与长期记忆相乘来达到过滤信息的目的，最后经过删除的长期记忆会与输入门产生的新信息相结合输入到输出门中。

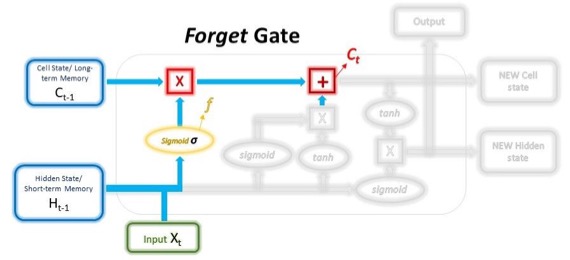


图-6 遗忘门

* + 输出门

输出门的主要功能是产生下一时刻使用的短期记忆/隐藏态。上一时刻的短期记忆和当前输入经由sigmoid处理后的信息，与经由激励函数（tanh）处理后的新长期记忆相乘，得到短期记忆。

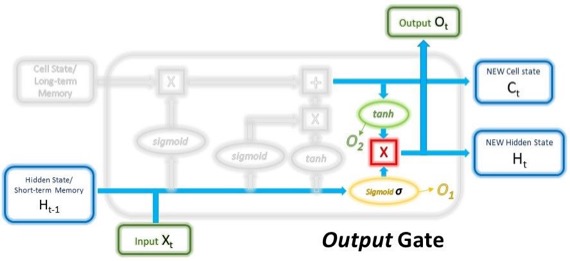


图-7 输出门

数据处理

LSTM使用了pytorch提供的实现版本，pytorch中的tensor（对矩阵的抽象），对数据格式有着严格的要求，在构建tensor的过程中，所传进去的数据必须是整齐的方阵结构。然而我们的网络分析数据中，每个连接形成的数据包被放入到一个文件中，这些文件的每一列对应数据包的特征，比如time\_diff、wsize，文件的每一行对应一个数据包，每个文件的行数不一样，所以每个文件包含的数据包不一样，这就导致将多个维度不一致的tensor放入同一个tensor形成超矩阵是不现实的，自然地，我们想到了对每个tensor进行填充或者截取操作，调整tensor的维度。因为这些tensor只有行数不同， 列数相同，最先想到的方案是求行平均数，小于行平均数的tensor填充0，大于行平均数的tensor截取后几行的数据，这个方法可行，但实验表明对数据的改动很明显会对实验结果产生影响。

实验及结果

1. 架构设计及参数配置

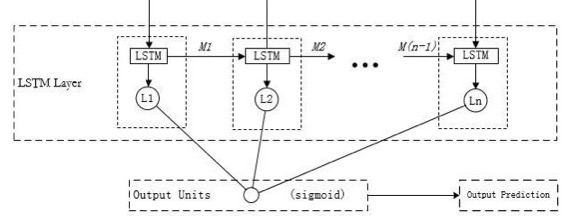


图-8 基本架构

基本原则

* + 采用最简单网络配置
  + 选取较小的数据集，看看能否获得较高的准确率
  + 过拟合、欠拟合的判断方法：
    - train loss不断下降， val loss不断下降 → 仍在学习
    - train 不断下降，val loss不断上升 → 过拟合
    - train loss不断下降， val loss趋于不变 → 欠拟合
    - train loss不变，val loss不变 → 网络饱和
    - train loss不断上升，val loss不断上升 → 网络结构有问题
    - train loss不断上升，val loss不断下降 → 数据集有问题
  + 过拟合的解决方法：
    - 批正则化（添加BN层）
    - 丢弃法（Dropout）
    - 正则化（l2）
  + 欠拟合的解决方法：
    - 去除正则化项
    - 增加网络深度
    - 增加训练集的数据量

初始配置

实验中设置了三层LSTM网络，LSTM内部dropout层使用0.5，LSTM层输出后首先经过了值为0.5的Dropout层，再经过批量正则化层，之后，使用一层全连接层将结果映射到50个类别中，产生分类结果。

训练集和测试集的划分比为8:2



表-4 初始配置

1. 代码

class RNN(Module):

def \_\_init\_\_(self, input\_size, hidden\_size, num\_layers, drop\_prob, num\_classes,bidirectional, device='cpu'):

super(RNN, self).\_\_init\_\_()

self.hidden\_size = hidden\_size

self.num\_layers = num\_layers

self.bidirectional = bidirectional

self.device = device

self.factor = 2 if self.bidirectional else 1

self.lstm = LSTM(input\_size, hidden\_size, num\_layers, dropout=drop\_prob, bidirectional=bidirectional, batch\_first=True)

orthogonal\_(self.lstm.all\_weights[0][0])

orthogonal\_(self.lstm.all\_weights[0][1])

self.fc = Sequential(

#ReLU(),

#Dropout(0.5),

#BatchNorm1d(hidden\_size \* self.factor),

# Linear(hidden\_size, 64),

#ReLU(),

# BatchNorm1d(64),

Linear(hidden\_size \* self.factor, num\_classes),

# Softmax(dim=1)

# Sigmoid()

)

def forward(self, x):

self.to(self.device)

hidden = self.initHidden(x)

out, (h\_n, c\_n) = self.lstm(x, hidden)

out = self.fc(out[:, -1, :])

return out

def initHidden(self, x):

return torch.rand(self.num\_layers \* self.factor, x.size(0), self.hidden\_size).to(self.device), torch.rand(self.num\_layers \* self.factor, x.size(0), self.hidden\_size).to(self.device)

1. 结果

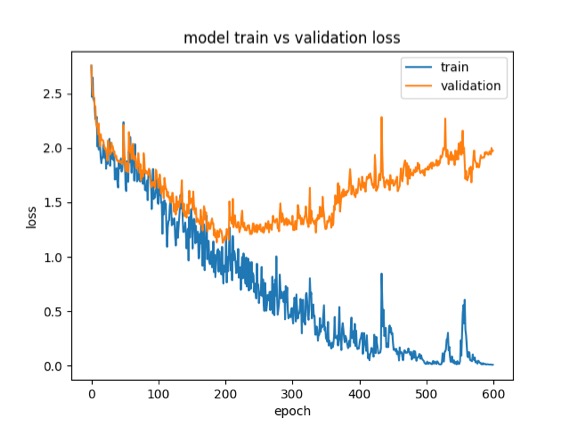


图-9 误差对比图

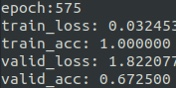


图-10 准确率

1. 优化

过拟合

初始配置训练的模型出现了过拟合的现象：train loss 不断下降，val loss 不断上升。下面将调整模型超参数来避免过拟合问题

Dropout

在训练的每一个批次中，通过忽略指定概率的特征检测器，可以明显地减少过拟合的现象。Dropout可以减少隐藏节点之间的相互作用，简单来说，Dropout在前向传播中，以指定概率让某些神经元的激活值停止工作，可以让模型的泛化能力更强。



表-5 修改dropout新配置

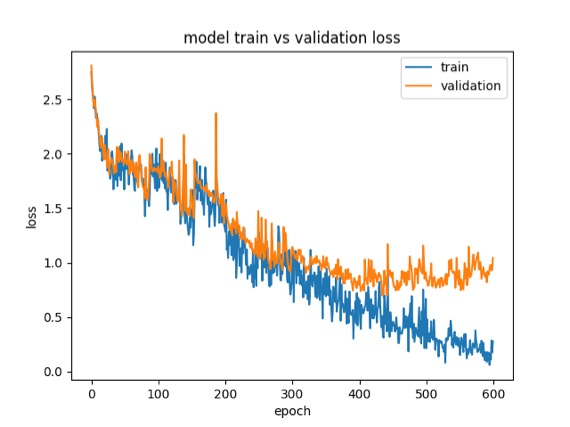


图-11 误差对比图

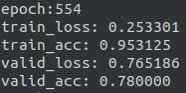


图-12 准确率

Dropout作用

* + 有效减弱过拟合现象
  + 测试集准确率提高了11%

Batch Normalization

**Internal Covariate Shift现象：**

训练深度网络的时候，每一次参数迭代更新后，上一层网络的输出数据经过这一层的计算后，会导致数据的分布发生变化。

**Covariate Shift现象：**

该现象与前者的区别在于，该现象发生在输入的数据上，输入数据中的训练集和测试集存在分布的差异性，给网络的泛化和训练速度带来影响，通常的解决方法是做归一化。

通过训练数据获得的模型可以在测试集上获得较好效果的一个前提是*IID*独立同分布假设，即假设训练数据和测试数据满足独立同分布。Batch Normalization是来确保每一层的神经网络的输入保持相同分布。



表-6 添加BN配置

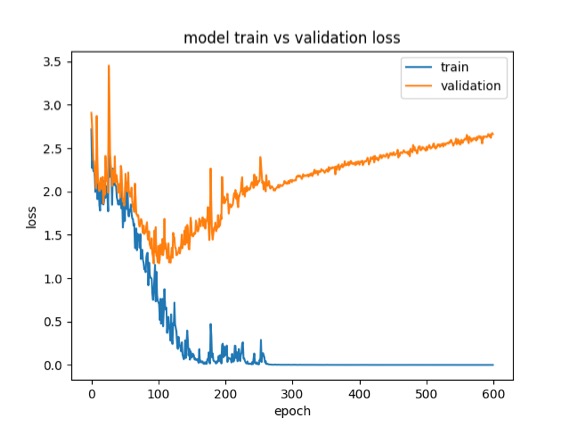


图-13 误差对比图

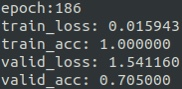


图-14 准确率

BN作用

* + 过拟合现象仍然存在，对过拟合帮助不大
  + 可以明显提升模型训练速度
  + 测试集准确率提高3%

正交（orthogonal）初始化

循环神经网络（RNN）中经常遇到梯度消失（vanishing）和梯度爆炸（exploding）的问题，在深度学习中，经常用到矩阵连乘的操作，所以值的增大或者减小会通过链式法则变化的非常快，影响网络的稳定性，正交初始化是一种解决办法。在这里，我们使用到了正交矩阵的特征值的模为1，无论我们执行多少次矩阵连乘，最后的矩阵都不会爆炸或消失。



表-7 添加正交初始化配置

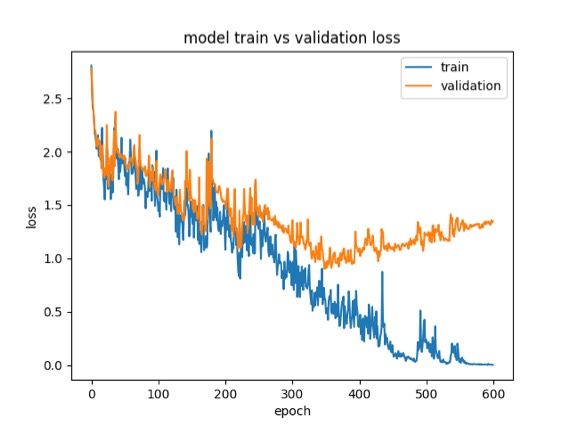


图-15 误差对比图

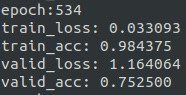


图-16 准确率

正交初始化作用

* + 减弱了过拟合现象
  + 测试集准确率提升8%

l2正则化

正则化可以用来防止神经网络训练过程中的过拟合问题，降低模型复杂度，减少噪音数据的影响。



表-8 添加l2正则化配置

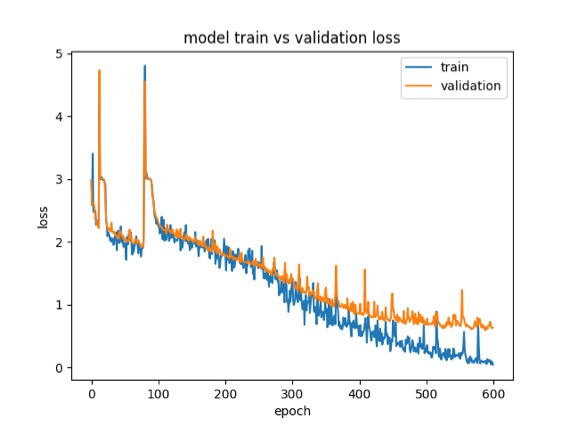


图-17 误差对比图

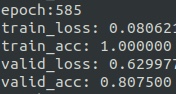


图-18 准确率

l2正则化作用

* + 有效减弱了过拟合现象
  + 测试集准确率提升了13%

最优参数



表-9 最优配置

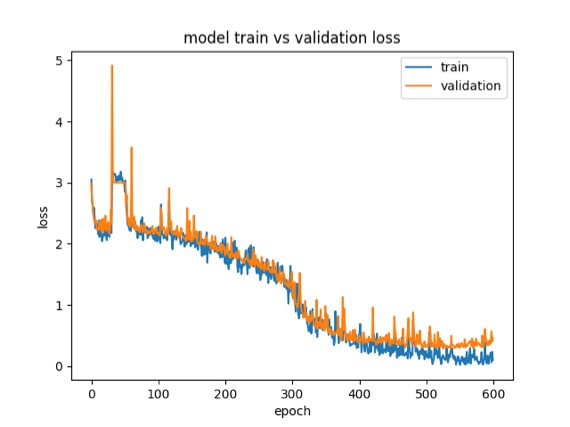


图-19 误差对比图

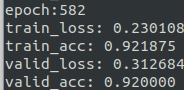


图-20 准确率

测试全数据集

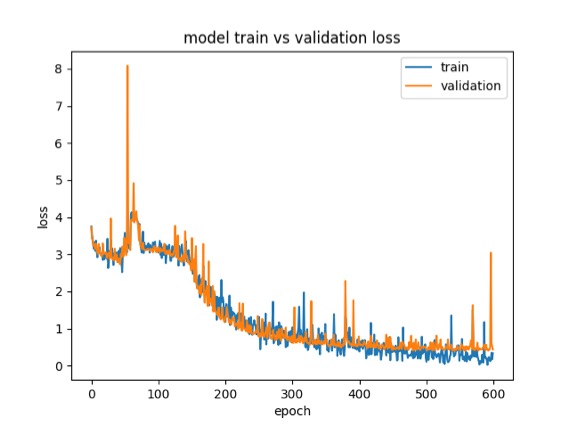


图-21 全数据集误差对比图

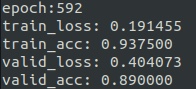


图-22 全数据集准确率

## 结论

本次加密流量分类实验中，我们尝试了两种方案，使用了机器学习中的CART决策树算法和深度学习中处理序列任务的LSTM网络，两种方案都在训练集上有着优异的表现，但都出现了过拟合的现象，决策树过拟合解决方法有后剪枝操作，由于将时间过多的放到了LSTM调参中，这一部分我们还未探索；LSTM中解决过拟合的方法较多，可以采用Dropout、Batch Normalization、正则化等方法。另外CART相较于LSTM方法的优点是训练模型的时间比较短，1366828条数据的训练时间在2～3分钟之内，LSTM的训练时间依数据集大小、参数设置情况而有所不同，需要花费较多的时间和计算资源。LSTM非常容易过拟合，降低过拟合的最有效手段是正则化和Dropout，batch normalization手段可提高模型训练速度，另外LSTM可输入变长序列，在pytorch中，该方法由packed*pad*sequence等方法实现，可避免数据的截断或者填充带来的误差，由于时间原因，这部分工作我们未能尝试成功。选择这两个模型进行重点实验原因是，最简单的配置可以获得较好效果，另外，我们尝试了简单的knn模型，准确率在40%，还有CNN最高40%左右，朴素贝叶斯在2%，其中CNN没有进行过多的参数探索，使用CNN进行流量分类的工作在参考11中可以了解到。关于网络的详细配置参考代码文件。

## 参考

1. <https://blog.floydhub.com/long-short-term-memory-from-zero-to-hero-with-pytorch/>
2. <https://www.zhihu.com/question/57828011>
3. <https://www.cnblogs.com/kamekin/p/10163743.html>
4. <https://zhuanlan.zhihu.com/p/38114469>
5. <https://blog.csdn.net/qq_23262411/article/details/100175943>
6. <https://mc.ai/how-to-reduce-overfitting-of-a-deep-learning-model-with-weight-regularization/>
7. <https://smerity.com/articles/2016/orthogonal_init.html>
8. <https://zhuanlan.zhihu.com/p/72503153>
9. Nitish Srivastava, Nitish Srivastava. Improving Neural Networks with Dropout[J]. 2013.
10. Effective Attacks and Provable Defenses for Website Fingerprinting.In USENIX Security Symposium (2014).
11. Sirinam P , Imani M , Juarez M , et al. Deep Fingerprinting: Undermining Website Fingerprinting Defenses with Deep Learning[J]. 2018.