计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2021,30(5):99−106 [doi: 10.15888/j.cnki.csa.007908]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

基于深度学习和区块链的 JavaScript 恶意代码检测系统①

陈　鹏[1], 韩　斌[1], 洪华军[2]

[1](江苏科技大学 计算机学院, 镇江 212000)
[2](中国船舶科学研究中心 软件工程技术中心, 无锡 214082)
通讯作者: 陈　鹏, E-mail: 2513943735@qq.com

摘　要: 目前基于深度学习的恶意代码检测技术是恶意代码检测领域的研究热点, 然而大多数研究集中于如何改进算法来提高恶意代码检测的准确率, 忽略了恶意代码数据集样本标签的不足导致无法训练出高质量的模型. 本文利用区块链技术来解决恶意代码检测数据样本孤岛和数据可信任的问题; 同时在代码特征提取上, 使用马尔可夫图算法提取特征; 基于分布式深度学习的训练融合区块链去中心化, 可溯源不可篡改的优势, 将不同算力贡献者采用同步训练更新模型参数. 通过仿真实验和理论分析验证了该方法的可行性和巨大的潜力.

关键词: 恶意代码检测; 深度学习; 区块链; 数据孤岛; 数据可信任

引用格式: 陈鹏,韩斌,洪华军.基于深度学习和区块链的 JavaScript 恶意代码检测系统.计算机系统应用,2021,30(5):99–106. http://www.c-s-a.org.cn/1003-3254/7908.html

JavaScript Malicious Code Detection System Based on Deep Learning and Blockchain

CHEN Peng[1], HAN Bin[1], HONG Hua-Jun[2]

[1](School of Computer Science, Jiangsu University of Science and Technology, Zhenjiang 212000, China)
[2](Software Engineering Technology Center, China Ship Science Research Center, Wuxi 214082, China)

Abstract: At the moment, the detection technology of malicious code based on deep learning is a research hotspot in the field of malicious code detection. However, most researches focus on how to improve the algorithm to enhance the detection accuracy of malicious code, but ignore the lack of sample tags in the data set of malicious code, failling to train high-quality models. In this study, the problem of detecting isolated islands of data samples and data trustworthiness of malicious code is solved by Blockchain technology, and code features are extracted with the Markov graph algorithm. The training fusion block chain based on distributed deep learning has the advantages of decentralization, traceability and non-tampering, and the contributors of different computing power adopt synchronous training to update model parameters. The feasibility and great potential of this method are verified by simulation experiments and theoretical analysis.

Key words: malicious code detection; deep learning; Blockchain; data islands; trusted data

1　引言

如今 Web 已经深刻融入我们的生活中, 为日常生活带来了诸多便利, 我们随时可以通过手机或者电脑来访问各种网站, 但同时这也给网络攻击者提供了大量的攻击机会. 网络攻击者为了达到其目的, 会将一些恶意的代码嵌入到网页中, 造成了正常的互联网用户的个人信息的泄露和个人财产的损失等问题.

针对 Web 安全问题, 研究者们提出了不同的解决方法. 恶意代码检测目前主要分为两大类, 一种是基于代码文本及结构的静态检测方法, 另一种是基于运行

① 收稿时间: 2020-09-14; 修改时间: 2020-10-13; 采用时间: 2020-10-21; csa 在线出版时间: 2021-04-28

结果的分析的动态检测方法[1]. 毛蔚等[2] 在分析进程访问行为异常度和相似度后, 引入了恶意代码检测估计风险, 并提出一种最小化估计风险实现主动学习的恶意代码检测方法; 吴迪[3] 针对 Android 平台的恶意代码检测只关注应用的单方面的特征, 提出了基于多类特征的 Android 恶意代码检测技术; 罗世奇等[4] 提出使用恶意代码纹理指特征纹融合 33 类恶意代码活动向量空间特征, 利用栈式自编码 (SAE) 模型来完成恶意代码的分类; 邓兆琨等[5] 提出了一种动静结合的网络数据检测方法, 在传统静态分析的基础上优化了检测算法, 同时引入了动态模拟运行的检测方式; 黄琨茗等[6] 提出一种最长频繁序列挖掘算法, 该方法提取样本文件的动态 API 序列, 并挖掘最长频繁序列集合构造词袋模型, 将 API 序列转化为向量, 使用随机森林算法来识别恶意代码.

深度学习技术在恶意代码识别领域已经取得了较好的效果, 但是仍然存在以下方面的问题: (1) 单个机构的恶意代码样本是有限的, 模型泛化能力需要高质量多样性的数据集; (2) 在大数据时代, 有效处理海量的数据是恶意代码检测效率提升的必要条件; (3) 如今数据已成为一项宝贵的资源, 数据的隐私和安全性越来越受到人们的关注. 恶意代码样本这类特殊性质的数据集, 如果被网络攻击者获取, 对于网络安全的威胁是巨大的.

区块链自 2008 年比特币白皮书[7] 提出以来, 因为其去中心化透明可信、防篡改可追溯、隐私安全保障等特点越来越受到人们的关注. 如杨雪梅[8] 探索了将区块链技术中的去中心化的思想与深度学习相结合并尝试运用于语音识别的可行性, 提出了一种适用于处理大规模的声学数据的融合分布式的深度学习算法模型. Gu 等[9] 提出了一种 Java 层的移动终端恶意软件检测方法, 并结合了区块链技术, 达到了多机构的共享的目的. 本文基于区块链技术, 结合卷积神经网络 (Convolutional Neural Network, CNN) 进行了深度融合, 提出了一种基于区块链+深度学习的 JavaScript 恶意代码检测系统设计方案, 利用区块链去中心化特点进行模型的分布式训练和同步更新.

2 相关理论

2.1 区块链

区块链被定义为一种按时间顺序来组织数据区块, 不同区块之间按序形成链条状连接的数据结构, 借助这种数据结构来构建数字账本[10], 是一种可信的, 去中心化防篡改的数据存储技术, 区块链的数据结构如图 1 所示.

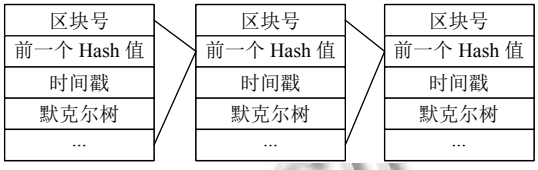

图 1 区块链数据结构示意图

区块链中的基本数据单元是块. 在每个块中存储了所有交易相关的数据信息, 主要包括了区块头和区块体两部分信息. 区块头中主要由父区块的哈希值 (previous hash)、当前的时间戳 (time stamp) 以及默克尔树根 (Merkle tree root) 等构成. 在区块体中一般包括若干个交易列表. 每个块与上一个块都是通过哈希值来进行连接的. 对于区块链网络中的节点, 在收到其余节点发送的数据以后, 并不是立即保存, 而是需要通过“共识机制”对当前的数据进行验证, 如果验证通过则保存到区块链中[11]. 为了加速区块的验证, 区块体中的所有的交易信息都按照默克尔树的形式保存下来, 并用默克尔树的根连接区块头和区块体[12]. 区块链中的每个节点都有自己一对公私钥, 利用非对称加密技术来保证信息的加密传输, 从而保护了交易者的隐私.

2.2 卷积神经网络

卷积神经网络是一种专门用来处理具有类似网格结构的数据的神经网络, 是深度学习的代表算法之一[13,14]. 经典卷积神经网络包括卷积层、池化层和全连接层. 卷积神经网络的卷积核参数共享机制使得能够以较低的运算来提取特征, 给定一张特征矩阵图, 通过不同的组合, 经过全连接达到分类的目的.

(1) 卷积层

卷积层的作用在于提取输入数据的特征. 通过卷积核与对应位置的数据相乘来获取图的特征, 接着使用非线性激活函数完成特征的映射. 定义输入图片 $P_{i,j}$ $(0 \le i \le W, 0 \le j < H)$, 大小为 $W \times H$, 则卷积运算的公式为:

$$y_{i,j} = \sum_{m=0}^{K-1} \sum_{n=0}^{K-1} P_{i+m,j+n} \omega_{m,n} \tag{1}$$

式中, $\omega_{m,n}$ $(0 \le m,n \le K)$表示大小为 $K \times K$ 的卷积过滤器的卷积运算的权值.

更一般地, 在卷积运算的过程中, 通常会有多个通道, 这些通道上的过滤器同时进行特征提取. 定义输入的图片为$P_{i,j,c}(0 \le i \le W, 0 \le j \le H, 0 \le c \le C)$, 表示大小为 $W \times H$, 通道数为 C 的图片, 卷积运算公式为:

$$y_{i,j,l} = \sum_{c=0}^{C-1} \sum_{m=0}^{K-1} \sum_{n=0}^{K-1} P_{i+m,j+n,c} \omega_{m,n,c,l} + b_{i,j,l} \tag{2}$$

式中, $b_{i,j,l}$表示卷积运算的偏置, $\omega_{m,n,c,l}$表示卷积神经网络的权值.

(2) 池化层

在卷积运算之后通常需要添加一层池化层, 来对卷积运算得到的特征减少映射的表示, 即降维操作. 池化操作具体指的是对局部特征范围内, 提取最大值或者平均值. 通常使用最大值池化和平均池化. 最大池化和平均池化的计算公式为:

$$y_{\max} = \max\{y_1, y_2, \cdots, y_{l-h+1}\} \tag{3}$$

$$y_{\text{average}} = \frac{1}{l-h+1} \sum_{i=0}^{l-h+1} m_i \tag{4}$$

(3) 全连接层

全连接层通常放置在最后几层用作分类器, 根据卷积操作提取得到的特征进行分类. 在全连接层之后, 对应分类上的概率分布通过 *softmax* 或者 *logsoftmax* 来计算. 定义原始的神经网络输出为$y_1, y_2, \cdots, y_n$, 则计算公式为:

$$softmax(y)_i = \frac{e^{y_i}}{\sum_{j=0}^{n} e^{y_j}} \tag{5}$$

$$logsoftmax(y)_i = \log \frac{e^{y_i}}{\sum_{j=0}^{n} e^{y_j}} \tag{6}$$

2.3 分布式深度学习

在深度学习训练过程中, 为了提高模型的训练速度, 可以通过一些参数设置来加速 GPU 的计算, 由于单个的 GPU 的加速的效率没有办法满足某些重量型深度学习模型算力要求, 由此出现了深度学习模型的并行方式. 分布式深度学习系统由多个共享模型和一个中央控制代理组成, 如图 2 所示. 其中贡献者可以贡献自己的模型、数据和算力, 训练完成后发送给中央服务器, 中央处理器负责融合和共享深度学习模型. 整个模型的训练都由一个中央服务器集中代理完成, 所以, 融合模型的过程中可能会遭受单点失效的影响[15,16].

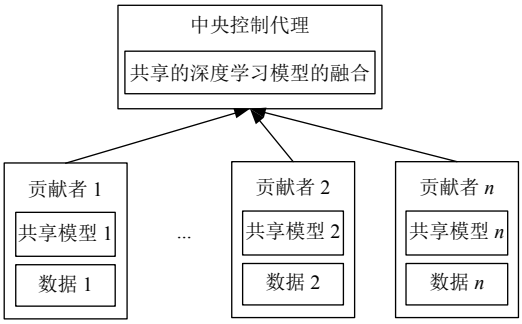

图 2 分布式深度学习示意图

2.4 协作分布式深度学习

与分布式深度学习不同, 系统中划分为了 3 个角色应用程序发起者、若干个算力贡献者和验证贡献者.其系统架构如图 3 所示.

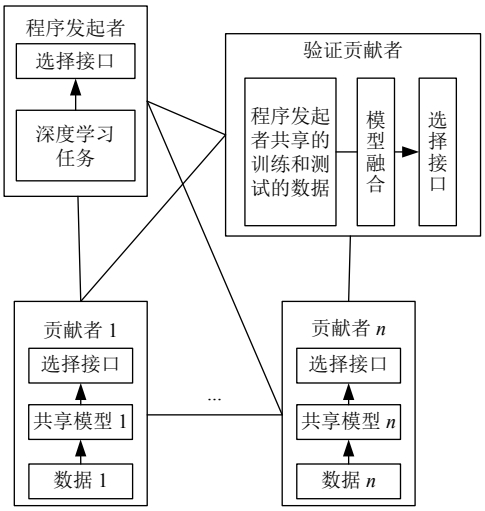

图 3 协作分布式深度学习示意图

在这个设计框架中, 每个单元都能够独立进行决策, 程序发起者负责定义训练模型的任务. 算力贡献者负责调用自己的算力资源进行构建训练任务并训练深度学习模型, 算力贡献者拥有充分的自主权, 可以根据程序发起者的不同任务选择本机的模型或者数据, 在训练完成后将其发送给验证者. 验证者在收到算力贡献者提交的参数后, 负责验证并评估模型的性能指标,

把结果发送给程序发起者. 发起者可以选择保留或者丢弃部分数据[17,18].

3 JavaScript 恶意代码检测系统设计

3.1 特征提取

Ni 等[19] 提出了一个恶意代码检测算法 MCSC, 首先将代码文件转化为基于 simhash 的灰度图, 然后使用卷积神经网络完成恶意代码家族的分类. 通过灰度图结合深度学习的方法均需要将图像转化为固定大小, 因此在图像转化的过程中会造成一部分的信息丢失. 本文使用基于马尔可夫图算法[20] 避免了上述的问题. 将 JavaScript 代码转化为 8 位的无符号整数向量, 整个代码就可以看做是一个字节流, 字节流表达了一个随机的过程:

$$Byte_i, i \in \{0,1,\cdots,N-1\} \tag{7}$$

式中, N 代表 JsavaScript 代码的字节长度. 对于每一个变量$Byte_i$, 有 256 种不同的状态值, 可以表示为:

$$Byte_i \in \{0,1,\cdots,255\} \tag{8}$$

由于恶意脚本都衍生自已知的恶意代码, 所以新的恶意脚本只有小部分的改动. 例如文献 [21] 将某恶意代码家族不同样本可视化为灰度图, 如图 4 所示, 可以发现相同恶意代码家族在字节分布上也会存在相似的特征.

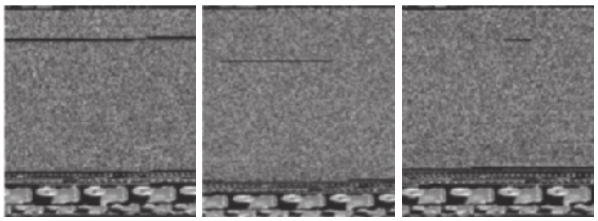

图 4 Fakeraean 木马家族图像实例[21]

基于上述恶意代码家族字节可视化图相似性的特点, 考虑相邻的两项$Byte_{i-1}$和$Byte_i$, 状态空间中状态$Byte_{i-1}$到状态$Byte_i$转换过程是随机的, 具备“无记忆的”性质, 且每个状态相互转换的概率是相同的. 所以在代码字节提取特征上, 为了简化问题, 这里假设$Byte_i$只与$Byte_{i-1}$有关联, 则随机的变量$Byte_i$可视为一个马尔可夫链:

$$P(Byte_{i+1}|Byte_0,\cdots,Byte_i) = P(Byte_{i+1}|Byte_i) \tag{9}$$

马尔可夫转移概率矩阵的构建是基于每一个状态迁移的概率. 如果用$P_{x,y}$记作是 x 的下一个字节为 y 的状态转移概率, 那么对于任意的字节序列计算公式为:

$$P_{x,y} = P(y|x) = \frac{f(x,y)}{\sum_{y=0}^{255} f(x,y)} \tag{10}$$

式中, $f(x,y)$表示 x 后面是 y 的频率, 且$x,y \in \{0,1,\cdots,255\}$. 以此类推考虑整个文件的字节的状态转移概率 M, 计算公式为:

$$M = \begin{bmatrix} P_{0,0} & P_{0,1} & \cdots & P_{0,255} \\ P_{1,0} & P_{1,1} & \cdots & P_{1,255} \\ \vdots & \vdots & \vdots & \vdots \\ P_{255,0} & P_{255,1} & \cdots & P_{255,255} \end{bmatrix} \tag{11}$$

基于式 (11) 生成马尔可夫图像, 图像中的每一个点对应于马尔可夫转移概率矩阵中的不同位置上的转移概率$P_{x,y}$. 将 JavaScript 代码转化为马尔可夫图的示意图如图 5 所示.

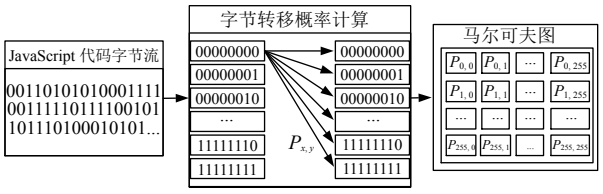

图 5 JavaScript 代码转化为马尔可夫图

3.2 系统架构设计

基于协作式分布深度学习[17] 的角色划分, 并结合机器学习模型训练的 3 要素: 数据、算法、算力. 本文将系统中用户角色划分为 3 种: 任务发起人、若干个算力贡献者和若干个模型验证者. JavaScript 恶意代码检测系统的用户对象关系示意图如图 6 所示. 系统角色划分其数量和对应关系如表 1 所示.

3.3 模型训练算法

某项任务的发起人负责定义任务的内容, 比如输入数据的属性、模型的预期输出、测试数据集、模型的准确度要求; 算力贡献者通过下载任务数据或者自己本地数据, 针对给定的任务进行模型的训练任务; 验证贡献者将算力贡献者训练好的模型进行融合, 然后再加密的测试数据集合上验证. 下文详细描述了流程, 其中所使用的符号定义如表 2 所示.

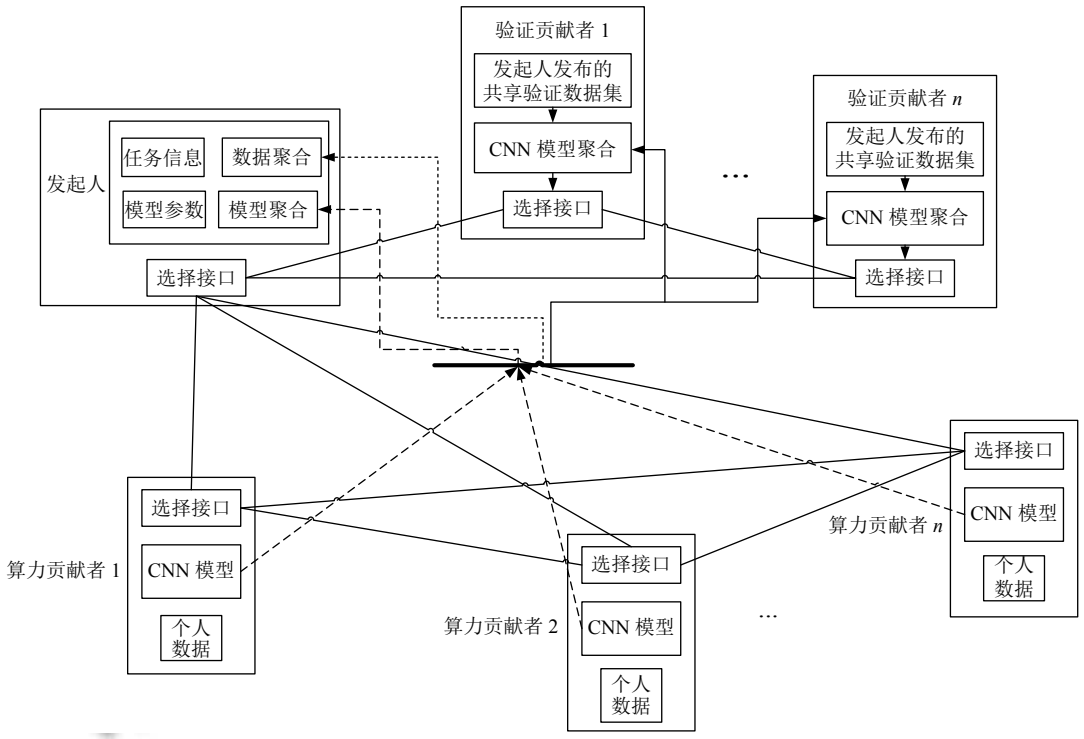

图 6 JavaScript 恶意代码检测系统用户对象关系

表 1 系统角色划分对应关系表

角色	数量	对应要素
任务发起人	1个	算法, 数据
算力贡献者	≥1个	算力, 数据
模型验证者	≥1个	算力

表 2 符号对应意义

符号	意义
N_i	第i个节点
Ori_i	第i个任务发起人
Cp_i	第i个算力贡献者
Ver_i	第i个验证贡献者
$PK_i, SK_i, Cert_i$	第i个公钥, 私钥和证书
$i \to j$	i发送消息给j
$D_{PK_i}(m)$	使用公钥解密m
$E_{SK_i}(m)$	使用私钥加密m
$D_{SK_i}(m)$	使用私钥对m解密
$Hash(m)$	对m求散列
timestamp	时间戳

对于集合 N, Ori, Cp, Ver满足如下条件:

$$\begin{cases} Cp \subseteq N \wedge Ver \subseteq N \wedge Ori \subseteq N \\ N = Cp \cup Ver \cup Ori \\ Cp \cap Ver = \varnothing \end{cases} \tag{12}$$

模型训练算法如下:

(1) 发起交易, 发起人Ori_i发布一笔计算任务交易, 内容包括深度学习模型以及融合模型、深度学习任务、模型参数、融合数据和决策接口记作$Data$. 由于模型训练的数据集比较庞大, 会存放在 IPFS (Inter Planetary File System) 中[22], 将对应的哈希值地址存放在交易中, 其余节点通过访问 IPFS 来获取节点数据信息, 上述过程可以表达如下:

$$Ori_i \to All_{Cp \cup Ver} : Packet_{Ori_i} = (Data \| Sig_i \| timestamp) \tag{13}$$

其中:

$$Sig_i = E_{SK_{Ori_i}}(Hash(Data)) \tag{14}$$

(2) 参数初始化, 所有节点完成参数初始化的同步, 每个算力贡献者Cp_i和验证者Ver_i需要对消息进行校验, 确定消息的发送身份以及数据的完整性, 校验函数记作$f_{\text{check}}(Pac, PK_{Ori_i})$, 运算结果中 1 代表验证通过, 0 代表丢弃该消息, 过程如下所示:

$$\text{if}: Pac = Packet_{Ori_i} =>$$

$$f_{\text{check}}(Pac, PK_{Ori_i}) = \begin{cases} 1, D_{Ori_i}(Pac.Sig_i) == Hash(Pac.Data) \\ 0, \text{otherwise} \end{cases} \tag{15}$$

(3) 训练参数, 训练中算力贡献者表示为:$Cp_{Ori_i} = \{Cp_{Ori_i,1}, Cp_{Ori_i,2}, \cdots, Cp_{Ori_i,d}\}, d < |N|$, 在每一轮的训练 k 中, 都会分别完成自己的前向和后向运算, 同时计算

得到每一个参数的更新量$\delta(Cp_{Ori_i},k)$;

(4) 广播参数量, 向周围的节点广播自己得到的参数量;

(5) 同步更新参数量, 此时节点将所有已经接收的进程进行同步 (All-Reduce), 对所有的参数更新量求平均, 得到基于所有进程训练数据的平均参数更新量, 该过程如图 7 所示.

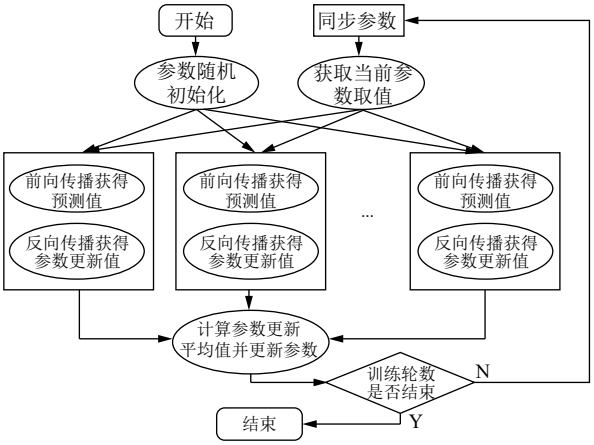

图 7 同步更新参数示意图

参数更新平均值计算如式 (16) 所示, 求得的参数作为下一轮训练的初始参数:

$$\partial_k = \frac{\sum_{j=0}^{d} \delta_j(Cp_{Ori_i},k_j)}{d} \tag{16}$$

(6) 进行下一轮的训练计算;

(7) 重复步骤 (3)~(6) 结束条件为训练集全部训练完毕.节点将参数聚合, 将模型数据写入区块广播给周围节点;

(8) 共识过程. 验证者将已经训练好的模型进行筛选, 将较优的模型汇总成候选集 (candidate set), 并将模型的候选集广播至其他验证节点.

① 验证贡献者将当前最优模型作为提案转发至其他验证节点;

② 验证者将收到的方案与本地候选集和提案模型进行对比, 择优放入候选集;

③ 定义验证贡献节点中模型的投票率χ, 模型投票率的阈值η, 最大阈值 *T_MAX*, 初始情形下投票阈值η_0为 50%. 判断模型是否有效方法为, 如果$\chi \geq \eta$,认为该模型为有效模型计入候选集, 否则该模型被淘汰. 在每一轮的共识过程中阈值η不断累加常量$\Delta\gamma$,表示为$\eta_i = \eta_{i-1} + \Delta\gamma, \Delta\gamma \in (0,1), i \in N^*$, 且对任意的$\eta_i$, 均满足$\eta_i$小于 *T_MAX*, 重复过程①~③, 直到$\eta_i$大于等于 *T_MAX*.

验证贡献者将从大于 *T_MAX* 的 UNL 节点确认的模型中选择最优模型写入本地的区块中. 同时验证贡献节点将本地区块广播给全网, 将收到区块和自己本地区块进行比对, 当正确率超过 *T_MAX* 时, 认为该区块链为有效区块.

4 仿真实验和理论分析

4.1 JavaScript 代码特征提取对比

本文实验使用的数据集为 Github 公开的恶意 JavaScript 恶意代码数据集[23], 对于良性的样本, 从 Alexa's Top 站点进行了爬取脚本的操作. 最终获得了 3526 条良性 JavaScript 脚本代码和 3566 条恶意 JavaScript 脚本代码. 实验将数据集划分为两部分, 70% 留作训练数据集, 30% 作为测试数据集.

使用机器学习工具包 Sklearn 所提供的算法库实现了文献 [24] 提出的 7 种不同的机器学习算法, 按照文中描述方法进行了代码特征的提取分类. 采用十折交叉验证的方法, 将数据集分为 10 等份, 其中的 1 份作为测试集, 其余 9 份用作训练集, 重复 10 次, 将平均值作为最终的结果. 与本文方法进行对比 (马尔可夫特征提取+CNN 分类器) , 实验果如表 3 所示.

表 3 本文特征提取方法和传统方法对比

分类器	准确率(%)	精度(%)	TPR	FPR
Naïve Bayes	78.94	88.95	0.761	0.163
J48 Decision Tree	96.81	96.50	0.973	0.037
Random Forest	98.27	98.86	0.976	0.011
SVM	97.22	97.30	0.971	0.027
AdaBoost	97.59	98.12	0.972	0.020
REP Tree	97.03	97.63	0.965	0.025
AD Tree	91.99	92.80	0.910	0.071
本文方法	**98.98**	**99.63**	**0.983**	**0.003**

对比实验结果发现, 马尔可夫图的特征提取准确率达到了 98.98%, 而准确率最低的为 Naïve Bayes 方法仅为 78.4%, 传统提取方法表现最优的为随机森林, 达到了 98.27%, 但是仍然低于本文方法达到的准确率, 说明本文基于马尔可夫图算法的特征提取能够有效提升恶意代码的检测率.

4.2 不同阈值 *T_MAX* 对共识算法性能的影响

为了分析不同阈值 *T_MAX* 对共识算法性能的影

响, 设计了一组仿真实验. 实验中固定$\Delta\gamma$为 10%, 算力贡献者和验证贡献者节点数均为 100 个. 本实验的区块链仿真平台采用 Python3.7 结合 Flask 框架搭建, 本地部署区块链, 对不同阈值 *T_MAX* 下的共识算法进行了模拟, 并记录吞吐量、模型收敛的时间, 多次试验求平均值, 实验结果如图 8 所示.

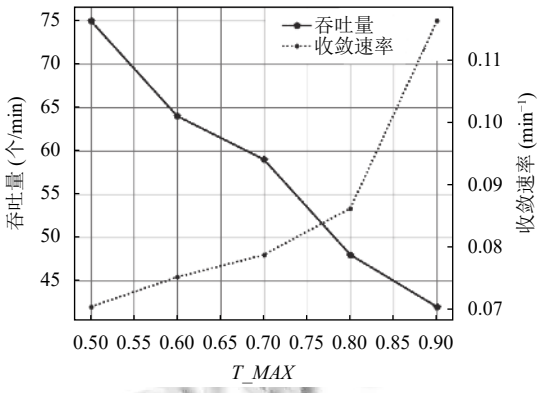

图 8 不同阈值 *T_MAX* 对共识算法性能影响示意图

实验结果表明, 随着阈值 *T_MAX* 的不断增加, 共识算法的吞吐量逐渐减少. 如果该阈值设置过小, 尽管共识算法吞吐量较大, 但是系统收敛到最优模型的速率会减慢; 反之, 如果阈值设置过大, 共识算法的吞吐量会减少, 但是算法能够较快收敛到最优模型.

4.3 检测系统设计理论分析

(1) 数据样本去中心化存储. 区块链上的节点出于某种原因无法共享数据, 此时可以在训练中除了可以拥有发起人共享的数据, 还可以调用本地的数据集样本参与模型的训练, 有效突破了数据孤岛的障碍, 另外数据训练得到的参数也将被验证者校验, 解决了数据可信任的问题.采用该平台设计, 极大扩充提升了训练样本的数量, 提升模型的泛化能力, 从而有效检测未知的 JavaScript 恶意代码;

(2) 模型训练结果参数防篡改. 模型一旦训练完成之后, 便会通过共识算法进行校验, 择优模型然后被存入区块链中. 假设存在恶意节点, 但是通过共识机制该模型也不会被其余节点认可, 所以修改是无效的, 保证了模型参数存储的安全性;

(3) 分散算力有效利用资源. 该平台设计通过协调多方的算力, 将传统的工作量证明机制的算法替换为有意义的深度学习模型训练, 从而有效合理地利用了算力资源, 推进了整个区块链上恶意代码检测准确度的提高.

5 结论与展望

深度学习作为一种新兴技术代表着先进生产力,而区块链则代表了新的生产关系, 两者有非常大的合作空间. 本文尝试将区块链技术和深度学习分布式系统训练相结合, 解决了数据孤岛和数据可信任的问题,另外在特征提取上使用马尔可夫图减少了人工的干预,提升了检测流程的效率, 该检测系统的设计可以为恶意代码检测研究提供一定的参考.

参考文献

1 Stokes JW, Agrawal R, McDonald G, *et al*. ScriptNet: Neural static analysis for malicious JavaScript detection. Proceedings of 2019 IEEE Military Communications Conference. Norfolk, VA, USA. 2019. 1–8.

2 毛蔚轩, 蔡忠闽, 童力. 一种基于主动学习的恶意代码检测方法. 软件学报, 2017, 28(2): 384–397. [doi: 10.13328/j.cnki.jos.005061]

3 吴迪. Android 恶意代码检测技术研究 [硕士学位论文]. 郑州: 解放军信息工程大学, 2017.

4 罗世奇, 田生伟, 禹龙, 等. 基于纹理指纹与活动向量空间的 Android 恶意代码检测. 计算机应用, 2018, 38(4): 1058–1063.

5 邓兆琨, 陆余良, 黄钊. 动静结合的网络恶意代码检测技术研究. 计算机应用研究, 2019, 36(7): 2159–2163.

6 黄琨茗, 张磊, 赵奎, 等. 基于最长频繁序列挖掘的恶意代码检测. 四川大学学报 (自然科学版), 2020, 57(4): 681–688.

7 Nakamoto S. Bitcoin: A peer-to-peer electronic cash system. https: //bitcoin. org/en/bitcoin-paper. (2018-12-08).

8 杨雪梅. 基于区块链技术的语音识别. 价值工程, 2019, 38(36): 281–283.

9 Gu JJ, Sun BL, Du XJ, *et al*. Consortium blockchain-based malware detection in mobile devices. IEEE Access, 2018, 6: 12118–12128. [doi: 10.1109/ACCESS.2018.2805783]

10 Underwood S. Blockchain beyond bitcoin. Communications of the ACM, 2016, 59(11): 15–17. [doi: 10.1145/2994581]

11 王群, 李馥娟, 王振力, 等. 区块链原理及关键技术. 计算机科学与探索, 2020, 14(10): 1621–1643. [doi: 10.3778/j.issn.1673-9418.2004029]

12 王健, 周念成, 王强钢, 等. 基于区块链和连续双向拍卖机制的微电网直接交易模式及策略. 中国电机工程学报, 2018, 38(17): 5072–5084.

13 LeCun Y, Bengio Y, Hinton G. Deep learning. Nature, 2015, 521(7553): 436–444. [doi: 10.1038/nature14539]
14 Gu JX, Wang ZH, Kuen J, *et al*. Recent advances in convolutional neural networks. Pattern Recognition, 2018, 77: 354–377. [doi: 10.1016/j.patcog.2017.10.013]
15 Zhang Y, Pezeshki M, Brakel P, *et al*. Towards end-to-end speech recognition with deep convolutional neural networks. arXiv preprint arXiv: 1701.02720, 2017.
16 Young T, Hazarika D, Poria S, *et al*. Recent trends in deep learning based natural language processing. arXiv preprint arXiv: 1708.02709, 2017.
17 Xu XW, Pautasso C, Zhu LM, *et al*. The blockchain as a software connector. Proceedings of the 13th Working IEEE/IFIP Conference on Software Architecture. Venice, Italy. 2016. 182–191.
18 Dennis R, Owen G. Rep on the block: A next generation reputation system based on the blockchain. Proceedings of 2015 10th International Conference for Internet Technology and Secured Transactions. London, UK. 2015. 131–138.
19 Ni S, Qian Q, Zhang R. Malware identification using visualization images and deep learning. Computers & Security, 2018, 77: 871–885.
20 Yuan BG, Wang JF, Liu D, *et al*. Byte-level malware classification based on markov images and deep learning. Computers & Security, 2020, 92: 101740.
21 龙廷艳, 万良, 邓烜堃. 基于卷积神经网络的 JavaScript 恶意代码检测方法. 计算机工程与应用, 2019, 55(18): 89–94. [doi: 10.3778/j.issn.1002-8331.1808-0005]
22 Politou E, Alepis E, Patsakis C, *et al*. Delegated content erasure in IPFS. Future Generation Computer Systems, 2020, 112: 956–964. [doi: 10.1016/j.future.2020.06.037]
23 https://github.com/HynekPetrak/javascript-malware-collection. [2020-08-23].
24 Patil DR, Patil JB. Detection of malicious JavaScript code in web pages. Indian Journal of Science and Technology, 2017, 10(19): 1–12.