计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2021,30(7):259−264 [doi: 10.15888/j.cnki.csa.007993]

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

基于 Bi-GRU 的 Webshell 检测①

李帅刚, 王全民

(北京工业大学 信息学部, 北京 100124)
通讯作者: 李帅刚, E-mail: ggjustnow@163.com

摘　要: Webshell 是一种隐蔽性较高的 Web 攻击工具, 其作用是获取服务器的操作权限. 在编写 Webshell 时, 攻击者通过一系列免杀技术来绕过防火墙, 这导致现有方法检测 Webshell 的效果不佳. 针对这一现状, 本文从文本分类的角度出发, 提出一种基于 Bi-GRU 的 Webshell 检测方法. 首先将网页脚本文件进行编译, 得到 opcode 指令; 然后, 通过 word2vec 算法将指令转换为特征向量; 最后, 使用多种深度学习模型进行训练, 以准确率、误报率、漏报率作为评估标准. 最终实验结果表明, Bi-GRU 检测效果优于其他算法模型, 证明该算法是可行的.

关键词: Webshell; RNN; Bi-GRU; 恶意代码; 网络安全

引用格式: 李帅刚,王全民.基于 Bi-GRU 的 Webshell 检测.计算机系统应用,2021,30(7):259–264. http://www.c-s-a.org.cn/1003-3254/7993.html

Webshell Detection Based on Bi-GRU

LI Shuai-Gang, WANG Quan-Min

(Faculty of Information Technology, Beijing University of Technology, Beijing 100124, China)

Abstract: Webshell is a highly concealed tool for Web attack, which is used to obtain the operating authority of servers. When writing Webshell, the attacker uses a series of anti-virus techniques to bypass the firewall, which leads to ineffective Webshell detection by existing methods. In response to this situation, we propose a Bi-GRU-based Webshell detection method from the perspective of text classification. Firstly, this method compiles webpage script files to obtain the opcode instructions. Secondly, the instructions are converted to feature vectors by the Word2Vec algorithm. Finally, a variety of deep learning models are used for training with accuracy, false positive rate, and false negative rate as evaluation criteria. The experimental results confirm the feasibility of the Bi-GRU-based Webshell detection since it is better than other algorithm models.

Key words: Webshell; RNN; Bi-GRU; malicious code; network security

随着我国互联网的突飞猛进, Web 应用在不同场景和业务中变得越来越重要, 这也给不法分子提供了数不胜数的 Web 漏洞靶场. 作为一种常用的攻击工具, Webshell 是以 asp、php、jsp 等常见的网页文件形式存在的一种代码执行环境, 也被叫做网页后门. 攻击者通过层出不穷的 Web 漏洞进入到后台管理系统后, 通常会将 php 或者 jsp 后门文件与网站服务器 Web 目录下正常的网页文件混在一起, 然后就可以使用浏览器来访问 Webshell, 从而进行文件上传下载或者进一步的权限提升. 根据国家互联网应急响应中心《2019 年上半年我国互联网网络安全态势报告》, CNCERT 检测发现境内外约 1.4 万个 IP 地址对我国境内约 2.6 万个网站植入后门, 同比增长约 1.2 倍, 其中, 约有 1.3 万个 (占全部 IP 地址总数的 91.2%) 境外 IP 地址对境内约 2.3 万个网站植入后门. 同时, 发生在我国云平台上的网络安全事件或威胁情况相比 2018 年进一步加剧,

① 收稿时间: 2020-10-27; 修改时间: 2020-12-02; 采用时间: 2020-12-18; csa 在线出版时间: 2021-06-30

被植入后门链接数量占境内全部被植入后门链接数量的 6 成以上[1]. 当前检测 Webshell 的方法如静态规则匹配、日志文件分析以及机器学习分类等, 在抵抗灵活多变的逃逸技术时[2], 未能有效保护服务器的安全. 针对现状, 急需一种能够有效检测 Webshell 的方法.

与正常网页文本的区别在于, Webshell 存在大量系统调用和文件操作函数如 eval、system、cmd_shell、readdir 等, 攻击者可通过上述特征函数远程操作服务器. 因此, 可以从文本分类的角度出发识别 Webshell. 随着深度学习的不断发展, 多种算法模型在文本分类中表现优异. 本文使用不同的算法模型进行实验对比, 结果表明, Bi-GRU 的检测效果最佳, 能够有效解决当前检测 Webshell 存在的问题.

1 相关研究

近年来, 对 Webshell 检测方法主要分为 3 类, 分别是基于流量的检测、基于日志的检测和基于文本的检测.

1.1 基于流量的检测

基于流量的检测是在网页文本与服务器通信过程中, 通过分析 http 请求与响应数据来识别 Webshell. 王应军[3] 从 Webshell 客户端工具和 Webshell 执行原理的角度出发, 对该过程中通信特征进行分析. Starov[4] 提出不同类型 Webshell 存在某些共性特征, 将这些特征作为分类的依据. 基于流量的检测, 虽然能够达到一定的准确率, 但是受限于人工观察样本和复杂的匹配规则.

1.2 基于日志的检测

基于日志的检测通过分析应用程序执行后生成的日志文件, 找到异常的请求日志, 从而识别出 Webshell. 石刘洋等[5] 通过文本特征匹配, 分析文件关联性进行检测, 研究总结出 Webshell 不常规行为生成的日志与正常日志存在巨大的出入. 潘杰[6] 采取文本分类 SVM 对日志进行检测, 首先将日志进行聚类, 然后对结果进行分析. 但是, 基于日志的检测方法存在滞后性, 即攻击者攻击了系统之后, 才会被发现.

1.3 基于文本的检测

基于文本的检测的优势在于发现及时、特征分析方便. 孟正等[7] 提出了一种基于 SVM 的检测方法, 该方法的准确率为 99%, 存在的不足之处是恶意样本较少. 张涵等[8] 提出一种基于多层神经网络的方式来进行检测, 将代码序列通过嵌入层转化为向量, 然后使用神经网络进行检测. 姜天[9] 采取卷积神经网络模型检测 Webshell, 该方法整体效果不错, 但评估方法中缺少漏报率等硬性评价指标. 周龙等[10] 提出了一种基于循环神经网络的方法来检测 Webshell, 使用 TF-IDF 获取特征, 以 GRU 为样本训练模型, 不过其准确率还有待提高.

目前, 在 Webshell 检测的学术研究中常用的算法有 SVM、MLP、CNN、RNN 等, 这些算法模型有一定的识别 Webshell 能力, 但还存在准确率较低, 误报率和漏报率较高等缺点.

2 本文方法

考虑到 php、asp、jsp 等文本形式的 Webshell 在语句逻辑上相差不大, 且 asp 和 jsp 在当前 Web 开发中逐渐被淘汰, 因此本文选择 php 网页脚本作为研究对象. 在特征选择方面, 使用 Word2Vec 算法[11] 将词语转换为特征向量, 该算法能够表征词语之间的关联度. 在算法模型选择方面, 循环神经网络[12] 在训练序列数据时表现优异. 本文最终选择能够传递正向和反向语句信息的 Bi-GRU 作为分类算法, 从前后两个方向挖掘语句信息, 以提高检测效果. 具体的研究过程如图 1.

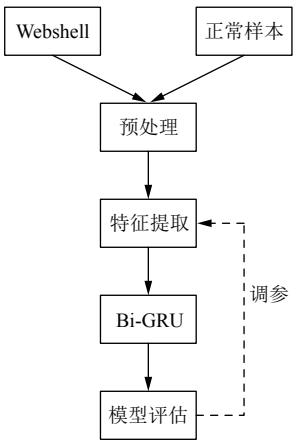

图 1 研究过程

2.1 预处理

数据的预处理工作主要包括文件去重和编译获取 opcode 指令[13]. 文件去重的目的是防止数据中出现大量相同的样本, 主要是通过计算每个样本的 MD5 值, 进行比对, 然后剔除相同的样本, 以防止污染训练数据. 完成文件去重之后, 需要将 php 源码转换为 opcode 指令. opcode 是 php 源码编译之后生成的中间代码, 它

将 php 脚本中可执行语句转换为 Zend 支持的 135 条指令. php 文件在服务器中的执行流程可分为两步: 第 1 步启动 Zend 引擎, 加载开启的扩展模块; 第 2 步 Zend 引擎对 php 文件进行一系列操作后得到 opcode, 然后执行 opcode. 只要源码中实现 Webshell 相关功能, 就会在 opcode 指令中体现出来. 在编译 opcode 的返回信息中, 需要通过正则表达式获取 opcode 指令字符串. 以最常见的一句话木马 webshell 为例.

代码 1. 一句话木马 Webshell

```
<? Php
@eval($_POST[‘cmd’]);
?>
```

其原理是通过 post 方式传入一个可执行的方法, 然后 eval 函数执行该方法, 服务器将运行结果通过 http 返回给用户, 这样就如同拿到管理员权限一样, 可以直接对服务器进行操作. 通常一句话木马需要配合一些工具来使用. 对一句话木马进行编译得到的 opcode 指令如序列 1.

序列 1. opcode 指令

```
BEGIN_SILENCE
FETCH_R
FETCH_DIM_R
INCLUDE_OR_EVAL
END_SILENCE
RETURN
```

2.2 特征提取

通过预处理后得到 opcode 指令字符串, 不能被算法模型识别. 因此, 需要提取样本的特征向量, 以数字的形式输入到算法模型中. 本文使用 Word2Vec 获取文本特征向量.

Word2Vec 在自然语言处理中被大量的使用, 它通过无监督的方式从语料库中学习词语之间的相似性, 然后将每个词用一个向量来表示, 从而用向量表征词的语义信息. 最早的文本向量化技术如 onehot 编码, 通过建立一个语料库大小的一维向量, 只是简单的将单词所在的位置设为 1, 其他位置设为 0. 通过上述方式得到的向量存在维度灾难、无法表征词语之间的语义信息等问题. 而上述问题, 在 Word2Vec 中可以被很好的解决. Word2Vec 模型结构有两种, 一个是 CBOW (Continuous Bag Of Words) 模型, 通过给定上下文来预测输入的值; 另一个是 Skip-Gram, 通过给定输入的词, 来预测上下文. 后者的结构如图 2 所示, 输入层用 one-hot 编码的向量, 输出层通过一个 Softmax 函数输出 0-1 之间的值, 表示当前词语与输入词之间的相似性, 所有输出层的值和为 1. 在隐层中定义词语需要转换的维度, 该层中的权重矩阵是网络学习的最终目标. 本文选择 Skip-Gram.

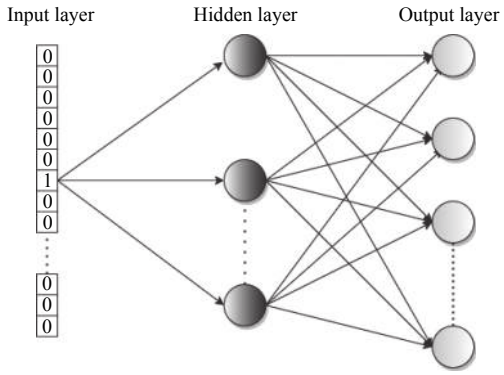

图 2 Skip-Gram 结构[11]

2.3 深度学习算法

本文提出一种基于深度学习的 Webshell 检测算法模型, 该模型使用 GRU 作为神经网络结构. GRU (Gated Recurrent Unit)[14] 是 LSTM (Long Short-Term Memory) 的简化变体, 其结构如图 3 所示. 它将 LSTM 最初的 3 个门, 输入门、输出门和遗忘门合并为重置门和更新门. 这样, 即可以解决 RNN 在训练过程中的梯度消失和梯度爆炸的问题, 同时又能使计算开销减小. GRU 的输入输出结构与普通的 RNN 一致, 存在一个 x^t 和上个节点传递下来的隐状态 (hidden state) h^{t-1}, 该参数保留了前一序列状态的信息, 同时也作为 GRU 的输入的值. 通过当前节点的输入 x^t 与上一节点的保留信息 h^{t-1} 计算两个门控制状态.

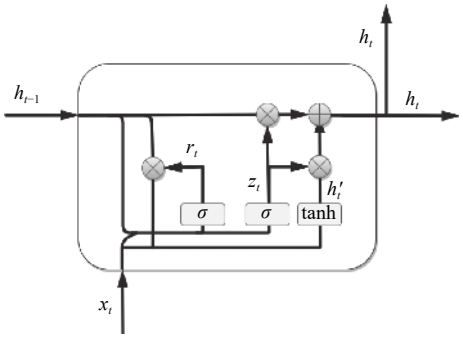

图 3 GRU 结构图[14]

图 3 中, z_t 为更新门 (update gate), 定义了前面记忆保存到当前时间步的多少, 计算过程如式 (1) 所示, 其中 W_z 表示权重, 使用 σ 函数将结果控制在 0~1 之间, 结果值越大, 保留之前的信息就越多. 重置门 (reset gate) 决定了对之前序列信息的遗忘程度, 公式如式 (2) 所示, 整个步骤与更新门一致. h_t' 用来表示当前序列记忆内容, 计算过程如式 (3) 所示, 重置门起到遗忘多少保留信息的作用, 使用 tanh 作为激活函数.h_t 的作用是保留当前单元的信息并传递到下一个单元中, 使用更新门来决定当前记忆内容和上一个序列中需要保留的信息, 其计算公式如式 (4) 所示.

$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t]) \tag{1}$$

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t]) \tag{2}$$

$$h_t' = \tanh(W \cdot [r_t \cdot h_{t-1}, x_t]) \tag{3}$$

$$h_t = (1 - z_t) \cdot h_{t-1} + z_t \cdot h_t' \tag{4}$$

2.4 实验框架

本文实验框架如图 4 所示. 样本通过嵌入层将每个 opcode 指令转换为特征向量, 接着输入到 Bi-GRU 中, 将正向通过 GRU 输出的隐状态 $\overrightarrow{h}_t$ 与反向通过 GRU 输出的隐状态 $\overleftarrow{h}_t$ 拼接, 从正向和反向获取样本中上下文信息. 值得一提的是, 本文选择将双向 GRU 的输出值作为全连接层的输入, 该层输出一维向量, 然后使用 *Softmax* 函数计算最终的概率, 如式 (5) 所示. 将模型输出值与 0.5 进行比较, 如果大于 0.5, 说明为正常样本, 否则为 Webshell[15–17].

$$Softmax(z_i) = \frac{e^{z_i}}{\sum_{c=1}^{C} e^{z_c}} \tag{5}$$

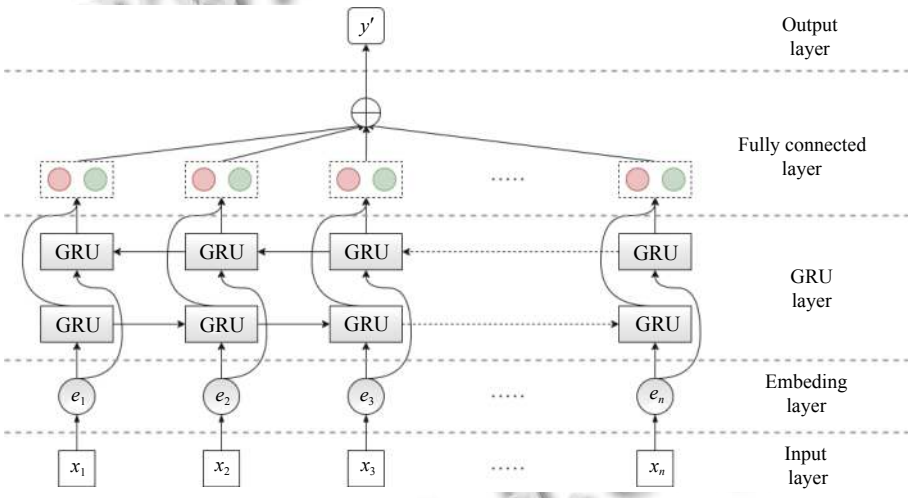

图 4 实验框架

3 实验过程

3.1 实验数据来源

实验数据分为 Webshell 样本和正常网页文本. 其中 Webshell 样本来源于 Github 排名靠前的仓库如 php-webshells、webshellSample 等. 正常样本来源于, 当前用户量较大的 php 框架如 Thinkphp、WordPress 等. 其中部分数据来源如表 1 所示. 由于不同 Webshell 所使用的 php 版本存在不一致的情况, 这造成部分样本编译 opcode 失败, 因此在编译 opcode 指令过程中直接忽略报错的样本. 为了方便进行后续分类, 本文将正常样本的标签设置为 1, 而 Webshell 样本设置 0.

对原始样本进行编译后, 最终得到 5830 样本, 其中正常脚本个数为 4664, Webshell 脚本个数为 1166, 两者比例约为 4:1.

表 1 数据来源

数据来源	标签
phpcms_v9.5.10_UTF8	1
Thinkphp_3.2.3_full	1
WordPress	1
Yii	1
https://github.com/JohnTroony/php-webshells	0
https://github.com/tanjiti/webshellSample	0
https://github.com/tennc/webshell	0
https://github.com/ysrc/webshell-sample	0

3.2 实验参数

系统结构如图 4 所示, 每个样本中 opcode 指令的

个数为 400, 超出的部分直接划掉, 不足的样本用 0 填补. 通过 Word2Vec 训练后, opcode 指令转换为特征向量, 维度为 128, 将其输入到 Bi-GRU 中进行训练. 在 GRU 层中, 正向和反向中 GRU 单元的个数都为 128, 选择 dropout 为 0.2 防止过拟合的产生. 输出层中的激活函数为 Softmax.

实验采用 Adam 优化算法, 学习率为 0.01, 损失函数使用交叉熵. 其中训练集和验证集的比列为 8:2, 共计训练 5 个 epoch.

3.3 评估标准

本实验是一个二分类问题, 选择准确率、假正率和假负率作为模型评估的标准. Webshell 样本可看作负样本, 正常样本为正样本. 根据表 2 定义混淆矩阵.

表 2 混淆矩阵

	预测为Webshell	预测为正常
实际为Webshell	*TN*	*FP*
实际为正常	*FN*	*TP*

准确率 (Accuracy, *Acc*), 表示预测结果正确的比率, 计算过程如式 (6) 所示.

$$Acc = \frac{TP+TN}{TP+FN+FP+TN} \tag{6}$$

假正率 (False Positive Rate, *FPR*), 表示 Webshell 被预测为正常样本的比率, 可作为漏报率, 计算过程如式 (7) 所示.

$$FPR = \frac{FP}{FP+TN} \tag{7}$$

假负率 (False Negative Rate, *FNR*), 表示正常样本被预测为 Webshell 的比率, 可作为误报率, 计算过程如式 (8) 所示.

$$FNR = \frac{FN}{TP+FN} \tag{8}$$

3.4 实验与分析

为了验证 Bi-GRU 的有效性, 本文共使用了 7 种算法进行对比分析. 在 SVM 和 MLP 的训练过程中, 特征向量皆为原始样本的 opcode 指令. 卷积神经网络算法使用 TextCNN[18] 的模型结构, 将单个文本转换为二维向量, 建立卷积层, 从图片角度对文本进行分类. 在循环神经网络的算法模型中, 分别比较了 LSTM、Bi-LSTM 和 GRU 模型的分类效果. 最终实验数据如表 3 所示.

表 3 实验数据

算法名称	准确率	漏报率	误报率	AUC
SVM	0.8747	0.5198	0.0022	0.8816
MLP	0.8597	0.3327	0.0760	0.9012
CNN	0.9699	0.0764	0.0129	0.9911
LSTM	0.9356	0.1433	0.0352	0.8705
BI-LSTM	0.8979	0.3216	0.0211	0.8670
GRU	0.9682	0.0605	0.0021	0.9855
Bi-GRU	0.9802	0.0350	0.0140	0.9926

从表 3 中可以看出, Bi-GRU 算法明显优于其他算法.Bi-GRU 在文本分类中的准确率最高, 达到了 98%, 高于 GRU 约 1.2%, 是因为 Bi-GRU 获取的是文本中双向记忆信息, 结构更复杂. 同时, LSTM 和 Bi-LSTM 在识别 Webshell 的准确率上表现差劲, 分别为 93% 和 89%, 其他 3 类算法中 CNN 的准确率表现较好, 但准确率低于 Bi-GRU 约 1.1%, 说明 CNN 并不适用于 Webshell 的检测. 在 7 种算法模型中, Bi-GRU 的漏报率最低, 为 3.5% 左右, 但是误报率稍逊于 SVM 和 CNN, 但在真实的检测环境下, 更应该关注其漏报的情况.

图 5 是所有分类算法训练结果计算出的 ROC 曲线, 其中 Bi-GRU、GRU、CNN 更加靠近左上角, 说明分类效果远超其他 4 类算法. Bi-GRU 将样本的中的每个 opcode 当做序列, 前面时间步的信息会被保留下来, 作为本时间步喂养的数据. 在模型训练的过程中, 学习 Webshell 上下文中特殊的函数组合与语句逻辑, 将学习到的内容作为识别 Webshell 和正常样本的依据. 同时, 从正反两个方向中发现语句的关联信息, 对 Webshell 的识别也起到了很大的帮助.

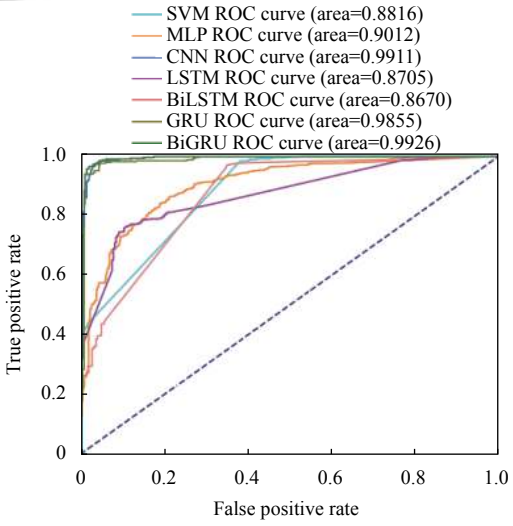

图 5 ROC 曲线

4 总结

本文提出了一种基于 Bi-GRU 的 Webshell 检测方法, 以 php 的 opcode 作为原始数据, 设定样本中 opcode 指令的个数为 400, 使用 Word2Vec 获取特征向量, 采用七种不同的算法模型进行对比试验, 得出 Bi-GRU 的模型在检测 Webshell 的准确率、漏报率上明显优于其他算法模型, 但是在误报率上还存在不足.

在接下来的研究中, 考虑使用词袋模型对样本中的 opcode 指令进行切分组合, 结合 TF-IDF 获取词频特征, 尝试其他文本分类模型如图卷积神经网络, 解决误报率高的问题, 进一步提高模型检测 Webshell 的效果.

参考文献

1 国家计算机网络应急技术处理协调中心. 2019 年上半年我国互联网网络安全态势. 国家计算机网络应急技术处理协调中心, 2019. 37–38.

2 龙啸, 方勇, 黄诚, 等. Webshell 研究综述: 检测与逃逸之间的博弈. 网络空间安全, 2018, 9(1): 62–68. [doi: 10.3969/j.issn.1674-9456.2018.01.014]

3 王应军. 基于流量的 Webshell 通信识别 [硕士学位论文]. 武汉: 武汉大学, 2018.

4 Starov O, Dahse J, Ahmad SS, *et al*. No honor among thieves: A large-scale analysis of malicious Web shells. Proceedings of the 25th International Conference on World Wide Web. Montreal, QC, Canada. 2016. 1021–1032. [doi: 10.1145/2872427.2882992]

5 石刘洋, 方勇. 基于 Web 日志的 Webshell 检测方法研究. 信息安全研究, 2016, 2(1): 66–73.

6 潘杰. 基于机器学习的 Webshell 检测关键技术研究 [硕士学位论文]. 天津: 中国民航大学, 2015.

7 孟正, 梅瑞, 张涛, 等. Linux 下基于 SVM 分类器的 Webshell 检测方法研究. 信息网络安全, 2014, (5): 5–9. [doi: 10.3969/j.issn.1671-1122.2014.05.002]

8 张涵, 薛质, 施勇. 基于多层神经网络的 Webshell 改进检测方法研究. 通信技术, 2019, 52(1): 179–183. [doi: 10.3969/j.issn.1002-0802.2019.01.032]

9 姜天. 基于卷积神经网络的 Webshell 检测方法研究. 信息技术与网络安全, 2019, 38(7): 27–31. [doi: 10.19358/j.issn.2096-5133.2019.07.005]

10 周龙, 王晨, 史崟. 基于 RNN 的 Webshell 检测研究. 计算机工程与应用, 2020, 56(14): 88–92. [doi: 10.3778/j.issn.1002-8331.1904-0420]

11 Mikolov T, Chen K, Corrado G, *et al*. Efficient estimation of word representations in vector space. arXiv: 1301.3781, 2013.

12 Lukoševičius M, Jaeger H. Reservoir computing approaches to recurrent neural network training. Computer Science Review, 2009, 3(3): 127–149. [doi: 10.1016/j.cosrev.2009.03.005]

13 PHP Official Website. Zend engine 2 opcodes in PHP at the core: A hacker's guide. 2014.

14 Cho K, van Merriënboer B, Gulcehre C, *et al*. Learning phrase representations using RNN encoder-decoder for statistical machine translation. Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing. Doha, Qatar. 2014. 1724–1734.

15 Liu PF, Qiu XP, Huang XJ. Recurrent neural network for text classification with multi-task learning. arXiv: 1605.05101, 2016.

16 Li F, Zhang MS, Fu GH, *et al*. A Bi-LSTM-RNN model for relation classification using low-cost sequence features. arXiv: 1608.07720, 2016.

17 Zhou P, Shi W, Tian J, *et al*. Attention-based bidirectional long short-term memory networks for relation classification. Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics. Berlin, Germany. 2016. 207–212. [doi: 10.18653/v1/P16-2034]

18 Kim Y. Convolutional neural networks for sentence classification. Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing. Doha, Qatar. 2014. 1746–1751. [doi: 10.3115/v1/D14-1181]