Comparação entre LSTM e CLCNN na detecção de requisições maliciosas em ataques na *web*

Rafael Bosse Brinhosa[1,2], **Marcos A. Michels Schlickmann**[1], **Eduardo da Silva**[1]
Carlos Becker Westphall[2], **Carla Merkle Westphall**[2]

[1] Instituto Federal Catarinense (IFC) – Câmpus Araquari

[2]Programa de Pós-Graduação em Ciência da Computação
Universidade Federal de Santa Catarina (UFSC)

{rafael.brinhosa, eduardo.silva}@ifc.edu.br, marcosamichels@gmail.com
{carlos.westphall, carla.merkle.westphall}@ufsc.br
brinhosa@inf.ufsc.br

Abstract. *Given the use of web applications on dynamic environments of cloud computing integrated with IoT devices, SQL injection and XSS (Cross-Site Scripting) attacks continue to cause security problems. The detection of malicious requests on the application level is a research challenge that's evolving by the use of Machine Learning and neural network. This paper presents a comparison between two architectures of machine learning to detect malicious web requests: LSTM (Long Short-Term Memory) and CLCNN (Character-level Convolutional Neural Network). The results show that CLCNN is more effective on all metrics, with an accuracy of 98.13%, a precision of 99.84%, a detection rate in 95.66% and an F_1-score of 97.70%.*

Resumo. *Com o uso de aplicações web em ambientes dinâmicos de computação em nuvem integrados com dispositivos IoT, os ataques de injeção de SQL e de XSS (Cross-Site Scripting) continuam causando problemas para a segurança. A detecção de requisições maliciosas a nível de aplicação representa um desafio na pesquisa, que está evoluindo usando técnicas de Machine Learning e redes neurais. Este trabalho apresenta a comparação entre duas arquiteturas de aprendizado de máquina usadas para detectar requisições web maliciosas: LSTM (Long Short-Term Memory) e CLCNN (Character-level Convolutional Neural Network). Os resultados demonstram que a CLCNN é a mais eficaz em todas as métricas, com uma acurácia de 98,13%, precisão de 99,84%, taxa de detecção em 95,66% e com um F_1-score de 97,70%.*

1. Introdução

Com o crescimento exponencial do uso de aplicações *web* nos ambientes dinâmicos e inteligentes (computação em nuvem e IoT), cada vez mais os usuários dependem da segurança de suas informações [Rodríguez et al. 2020]. Neste crescente uso, um sistema vulnerável pode ser alvo de ataques capazes de impactar seu funcionamento, e até mesmo expor

informações dos usuários. Na lista das dez maiores vulnerabilidades e riscos de segurança em 2017, publicada pelo *Open Web Application Security Project* [OWASP 2017], os ataques de injeção de SQL estão em primeiro lugar, e o *Cross-Site Scripting* (XSS), outra forma de injeção de instruções maliciosas, em sétimo.

A OWASP afirma que os ataques de SQLIA (*SQL Injection Attack*) são os mais críticos, por serem comuns de ocorrer, de fácil utilização, e podem ocasionar diversos impactos à aplicação, à empresa e aos usuários. O SQLIA ocorre na inserção de consultas SQL maliciosas através de entradas de dados da aplicação para execução no banco de dados, que permite a extração ou manipulação das informações armazenadas [Al-Khurafi e Al-Ahmad 2015]. Já o XSS é um ataque em que um código malicioso é injetado nas páginas *web* acessadas pelas vítimas, e é executado no lado do cliente. Com ele, é possível obter informações pessoais ou os *cookies* de sessão do usuário para roubar sua identidade [Rodríguez et al. 2020]. Assim, o XSS fornece ao atacante a possibilidade de roubar informações sensíveis ou até controlar alguns dispositivos ou aplicações.

Essas vulnerabilidades são comumente evitadas utilizando consultas parametrizadas em instruções preparadas[1], ou uma lista de validação de entrada para aceitar apenas valores pré-definidos [OWASP 2020]. Validações como essas são implementadas pelos WAFs (*Web Application Firewall*) ou pelas próprias aplicações. Entretanto, os sistemas comuns ainda são passíveis de vulnerabilidades por, normalmente, dependerem de filtros por expressão regular criados para padrões de ataques conhecidos [Tian et al. 2019]. Considerando a abrangência dos ataques de injeção, uma solução genérica, aplicada igualmente a qualquer sistema, pode não ser suficiente. Essas vulnerabilidades, quando exploradas, tem o potencial de comprometer a segurança da aplicação, de seus dados e usuários.

Diante disso, várias pesquisas propõem o uso de *Machine Learning* e redes neurais para detectar ataques de forma flexível e customizada para cada sistema [Ito e Iyatomi 2018, Rego e Nunes 2019, Tang et al. 2020]. Utilizando registros de acesso ao servidor *web*, tanto legítimos como anômalos, o objetivo é aproximar a detecção de ataques para o contexto da aplicação. Com os avanços dos estudos e aplicações de DNNs (*Deep Neural Networks*) para o Processamento de Linguagem Natural (NLP, *Natural Language Processing*) [Goldberg 2016], questiona-se o nível de eficácia desses algoritmos para classificar as requisições. As principais classes de DNNs, e utilizadas para tarefas de NLP, são a RNN (*Recurrent Neural Networks*) e a CNN (*Convolutional Neural Network*) [Yin et al. 2017]. No âmbito das RNNs a mais utilizada é a arquitetura LSTM (*Long Short-Term Memory*) que possui ligações entre os neurônios de uma camada permitindo um maior entendimento e processamento da sequência inteira. Já em [Zhang et al. 2015] foi proposta a CLCNN (*Character-level Convolutional Neural Network*) para tarefas de NLP, permitindo o uso de caracteres na CNN.

Existem trabalhos de pesquisa recentes utilizando esses métodos para a prevenção de requisições maliciosas, e até propondo variações de modo a obter melhores resultados [Liang et al. 2017, Ito e Iyatomi 2018, Rego e Nunes 2019]. Normalmente, comparam seus resultados com os obtidos sem a característica que propuseram, a outras inovações semelhantes, ou a WAFs e sistemas detectores que não utilizam redes neurais. Entretanto, não foram encontrados estudos que comparassem as implementações clássicas

[1] *Stored Procedures* nos bancos de dados, ou consultas SQL parametrizadas.

da LSTM e da CLCNN, o que pode ser relevante para observar o seu desempenho e os resultados obtidos podendo servir de parâmetro para novas construções.

Este trabalho compara o desempenho na detecção de requisições maliciosas entre a LSTM e a CLCNN, treinadas e avaliadas com o conjunto de dados "HTTP DATASET CSIC 2010" [Giménez et al. 2010] tendo como métricas a acurácia, precisão, taxa de detecção e F_1-*score*. Os resultados demonstram que a CLCNN é a mais eficaz em todas as métricas, com uma acurácia de 98,13%, precisão de 99,84%, taxa de detecção em 95,66% e com um F_1-*score* de 97,70%.

O restante do texto do artigo está organizado da seguinte maneira: a seção 2 descreve os trabalhos relacionados; na seção 3 são apresentados os conceitos de redes neurais; a seção 4 apresenta os experimentos desenvolvidos; na seção 5 os resultados são discutidos e a seção 6 finaliza o artigo com as conclusões.

2. Trabalhos relacionados

Nesta seção são apresentados trabalhos com motivações semelhantes e que contribuíram para o desenvolvimento desta proposta, seja na implementação da LSTM ou da CLCNN, ou para a comparação das duas arquiteturas. Em [Torrano-Gimenez et al. 2009] é, pela primeira vez, utilizado o conjunto de dados "HTTP DATASET CSIC 2010" na proposição de um WAF para detectar anomalias através de uma base contendo o comportamento legítimo do sistema, mas sem o uso de NNs (*neural networks*).

A possibilidade de utilização das redes neurais para detecção de requisições anômalas se dá pelos avanços que elas receberam para tarefas de processamento de linguagem natural. Em [Yin et al. 2017] foram comparadas a RNN e a CNN para tarefas de NLP, em seus experimentos representou os registros em palavras e realizou uma série de tarefas de NLP nas RNNs e na CNN, em seus resultados as duas classes fornecem informações complementares, senda a RNN mais eficaz no entendimento da sequência inteira, enquanto a CNN possui melhores resultados no reconhecimento das frases ou palavras-chave. Em [Zhang et al. 2017] a CNN foi adotada para detectar ataques *web* também utilizando o conjunto de dados CSIC 2010 com a incorporação das requisições por palavras, tendo que pré-processá-las, e obtiveram bons resultados na detecção de requisições anômalas, com uma taxa de detecção de 93,35% e acurácia em 96,49%.

A CLCNN (*Character-level Convolutional Neural Network*) foi proposta em [Zhang et al. 2015] para a classificação de textos ao nível de caractere. Seus resultados demonstram um alto desempenho mesmo sem conhecimento prévio da linguagem. A partir dessa prova de conceito, surgiram alguns estudos em que a CLCNN é utilizada para implementar um detector de vulnerabilidades de segurança. O primeiro, [Saxe e Berlin 2017], propôs CLCNNs paralelas de diferentes tamanhos para detectar URLs, caminhos de arquivos e chaves de registro maliciosas utilizando incorporação por caracteres, conseguiundo automatizar completamente a extração de características além de obter uma melhora de 5% a 10% na taxa de detecção com um índice de 0,1% de falsos positivos. Em [Ito e Iyatomi 2018] foi implementada uma CLCNN para a detecção de requisições *web* maliciosas utilizando o *dataset* CSIC 2010 a partir da implementação feita por [Saxe e Berlin 2017]) com o objetivo de aprimorá-la, obtendo bons resultados na classificação de requisições com uma acurácia de 98,8% e uma média de 2,35 milissegundos de tempo de execução, sendo um pouco melhores que os obtidos utilizando a

arquitetura de [Saxe e Berlin 2017], mesmo com uma implementação mais simples.

A LSTM é utilizada por [Liang et al. 2017] para construir um detector de requisições maliciosas. O estudo utiliza a base CSIC no treinamento da LSTM para aprender inicialmente apenas com registros normais. Após, implementa um classificador utilizando *multilayer perceptron* (MLP), treinado com os registros anômalos, que recebe a saída da LSTM para distinguir entre requisições normais e anômalas. Seus resultados foram competitivos com outras técnicas, com sua implementação obtendo uma acurácia de 98,38% e uma taxa de detecção em 98,58%. Além disso, a incorporação por caracteres também os livrou de realizar seleção de características nos registros.

Existem alguns outros estudos recentes que propõem variações, tanto da CLCNN quanto da LSTM, com o objetivo de obter melhores resultados em detectores de requisições maliciosas. Em [Rego e Nunes 2019] é proposto um filtro de bloom para apoiar os detectores reduzindo o tempo de execução, auxiliando até no aumento da acurácia e precisão. [Tang et al. 2020] apresenta *ZeroWall*, uma abordagem não supervisionada aplicada em conjunto com um WAF para detectar ataques de dia-zero com duas LSTMs em formato codificador-decodificador. Sua estrutura procura obter os padrões sintáticos e semânticos das requisições benignas, e detecta corretamente os ataques de dia-zero, que o WAF falhou em detectar, obtendo um F_1-*score* acima de 98%.

Alguns dos estudos apresentados foram considerados na comparação realizada neste trabalho. As implementações propostas nesses estudos avaliam seus resultados em relação aos obtidos sem a sua inovação, a outros estudos semelhantes, ou a WAFs e sistemas de detecção que não utilizam redes neurais. Entretanto, não consideram como a DNN utilizada, seja a LSTM ou a CLCNN, desempenharia em sua forma clássica. Dessa forma, o presente trabalho estuda e compara essas duas arquiteturas de modo a fornecer uma linha de base a trabalhos futuros que possam vir a utilizá-las em uma implementação ou a aprimorá-las para a detecção de requisições maliciosas às aplicações web. A Tabela 1 apresenta a comparação de alguns trabalhos relacionados e este estudo, por meio das características e objetivo de suas contribuições. Nela, pode-se ver a predominância do conjunto de dados CSIC, e a tendência dos estudos de construir novos detectores.

Tabela 1. Sumário de trabalhos relacionados

Trabalho	Conjunto de dados	Entrada	Rede Neural	Objetivo
[Torrano-Gimenez et al. 2009]	CSIC	Palavras		Novo detector
[Zhang et al. 2017]	CSIC	Palavras	CNN	Novo detector
[Ito e Iyatomi 2018]	CISC	Caracteres	CLCNN	Simplificar
[Liang et al. 2017]	CSIC, próprio	Palavras	LSTM, GRU, MLP	Novo detector
[Tang et al. 2020]		Palavras	LSTM	Novo detector
Este estudo	CISC	Caracteres	CLCNN, LSTM	Comparação

3. Rede Neural (NN)

Os modelos computacionais utilizados para o aprendizado de máquina são conhecidos como *Artificial Neural Networks (ANNs)*, uma forma da inteligência artificial. As ANNs consistem de neurônios interconectados que recebem um sinal (valor de entrada), que é computado, e o resultado é enviado, como sinal de entrada, para o próximo neurônio.

Os neurônios e suas interconexões possuem pesos, ajustados conforme o aprendizado, aplicados nos sinais de entrada para variar a sua força, a fim de reduzir o resultado da *loss function*.

É comum dividir os neurônios em camadas, cada uma podendo aplicar diferentes funções de transformação nos sinais. Cada camada possui neurônios do mesmo tipo, e são conectadas umas às outras visando a obter o melhor resultado para o objetivo definido. As aplicações em ANN podem ser avaliadas considerando métricas como a precisão, velocidade de processamento, latência, tolerância a falhas, escalabilidade e convergência [Abiodun et al. 2018]. Entre as ANNs existem as DNNs (*Deep Neural Networks*), que usam múltiplas camadas, e suas principais categorias são a *Recurrent Neural Network* e a *Convolutional Neural Network* [Yin et al. 2017].

Entre os principais usos das ANNs está a de classificação, como reconhecimento de padrões e detecção de anomalias. O estudo realizado em [Abiodun et al. 2018] relata que os modelos contendo propagação em *feedforward* ou *feedback* possuem melhores resultados aplicadas na solução de problemas humanos. Tanto as Redes Neurais Recorrentes quanto as Convolucionais podem utilizar desses mecanismos.

3.1. LSTM

As RNNs são derivadas de ANNs contendo propagação em *feedforward*. Isso permite que usem sua memória interna para processar sequências de entrada com tamanho variável [Dupond 2019]. Entretanto, esse tipo de rede neural encontra um problema ao utilizar métodos de aprendizado em *gradient descent* e *backpropagation*, que é conhecido como *vanishing gradient problem* [Liang et al. 2017].

Nos métodos de aprendizado em *gradient descent* e *backpropagation* cada peso dos neurônios recebe uma alteração proporcional ao resultado da *loss function* e conforme o peso atual a cada iteração do treinamento. O *vanishing gradient problem* ocorre em alguns casos quando o gradiente é extremamente pequeno, impedindo a alteração do peso dos neurônios, podendo afetar e até impedir o aprendizado da rede neural mesmo com mais treinamentos. Esse problema afeta as RNNs em componentes de longo prazo, impossibilitando o modelo de aprender a correlação entre eventos temporalmente distantes.

Para evitar o *vanishing gradient problem*, surgiu o *Long Short-Term Memory* (LSTM). É uma arquitetura de RNN que, diferente das NNs com *feedforward* habituais, também possui conexões de *feedback*, permitindo que processe sequências inteiras de dados, como texto, vídeo e áudio. Isso é dado pela sua capacidade de perceber e armazenar por um longo período a correlação entre informações importantes, mesmo que estejam a uma distância desconhecida.

3.2. Rede Neural Convolucional a Nível de Caractere (CLCNN)

As *Convolutional Neural Networks* recebem esse nome da operação entre matrizes chamada de convolução. Elas possuem um ótimo desempenho, principalmente no processamento de imagens, como classificação, e em tarefas de NLP. Entretanto, a CNN não pode relacionar categorias dependentes espacialmente, ou seja, em que suas localizações influenciam na classificação [Albawi et al. 2017].

A CLCNN foi proposta por [Zhang et al. 2015] para a classificação de textos, porém, em conjuntos de dados de larga escala, não é necessário o conhecimento prévio

das palavras ou da estrutura semântica e sintática da linguagem. Essas descobertas levam a entender que a CLCNN poderia funcionar em diferentes linguagens, visto que utilizam sempre de caracteres, independentemente da possibilidade de separá-los em palavras. A CLCNN recebe uma sequência de caracteres codificados e definidos a um tamanho máximo para serem acumulados em *batches*, permitindo a representação de cada registro em um *tensor* (matriz *one-hot* de caracteres). Após, é realizada a convolução aplicando um *kernel*[2] de uma dimensão no *tensor*.

4. Experimentos e resultados

Esta seção apresenta os experimentos realizados, iniciando pela escolha, tratamento e seleção de recursos do conjunto de dados. Após, é apresentada a implementação das duas arquiteturas, em CLCNN e LSTM. Por fim, são descritas as métricas utilizadas para avaliar as duas implementações.

4.1. Planejamento dos experimentos

A avaliação de qual técnica de rede neural é a mais eficaz em prever requisições maliciosas foi realizada em dois experimentos, um LSTM e outro em CLCNN. Em ambos, após preparar o conjunto de dados, foi necessário primeiro treiná-los. Para isso, o conjunto de dados foi dividido com 80% separado para o treinamento e o restante para a sua avaliação.

Os experimentos foram realizados em um *notebook* Samsung Expert x40, com processador Intel Core i5-8265U, com cache de 6MB, 8 GB de memória RAM DDR-4, e GPU NVIDIA GeForce MX110 Graphics de 2 GB GDDR5. O sistema operacional é Ubuntu 20.04 LTS de 64 bits e foi utilizado o Python 3.8.5 com as bibliotecas Tensorflow 2.4.1 e Keras 2.4.3. Apesar da presença de uma GPU, não foi utilizada aceleração de vídeo nos treinos e experimentos.

4.2. Conjunto de dados

Para treinar e avaliar as duas soluções, em LSTM e CLCNN, foi utilizado o conjunto de dados "HTTP DATASET CSIC 2010". Ele foi construído por [Giménez et al. 2010] com o objetivo de testar a eficiência de sistemas de proteção, como os WAFs, a ataques *web* mais atuais como SQLIA, XSS, *buffer overflow*, injeção de CRLF, entre outros. Esse conjunto de dados é amplamente usado para esse propósito [Torrano-Gimenez et al. 2009, Liang et al. 2017, Ito e Iyatomi 2018, Tekerek 2021].

O conjunto de dados CSIC possui 36.000 registros de tráfego legítimo e 25.065 requisições anômalas. Existe um desbalanceamento razoável por conter 30,38% requisições legítimas a mais que anômalas. Com os dados devidamente classificados e embaralhados os caracteres de suas requisições foram convertidos em *tokens*, sendo representados por números. Isso feito, o conjunto de dados foi dividido com 80% dos registros alocados para realizar o treinamento do modelo da rede neural e 20% para, por fim, validar seu desempenho. Essa distribuição dos dados é demonstrada na Tabela 2.

O conjunto de dados pode ser representado como $U = u_1, u_2, ..., u_n$, em que u_i representa a $i$ésima URL. Para cada u_i existem um $y_i \in \{0, 1\}$, em que $y_i = 1$ indica que a URL u_i é uma requisição maliciosa. Na "tokenização" (*tokenization*) é construído um dicionário contendo todos os caracteres presentes no conjunto de URLs, mas

[2]Matriz cujo valores são ajustados no treinamento para reduzir os resultados da *loss function*.

em números. Isso é feito para permitir que esses dados possam ser inseridos na camada de incorporação (*embedding*) da NN, que aceita apenas inteiros positivos.

Tabela 2. Distribuição dos dados para treinamento e validação

	Treino	Validação
Legítimo	28855	7145
Anômalo	19997	5068
Total	48852	12213

4.3. Seleção dos recursos

Para treinar as NNs foram utilizadas apenas as URLs, com seu método http, parâmetros e corpo (para requisições *post*). Para manter as requisições mais concisas, foi removido o espaço entre o método http e a requisição, e os campos do corpo das requisições *post* foram representados da mesma maneira que parâmetros *get*, ou seja, após o `?` e separados por `&`. Abaixo são apresentados alguns exemplos dos registros utilizados.

Requisições normais:

```
gethttp://localhost:8080/tienda1/publico/entrar.jsp?errormsg=credenciales+incorrectas

posthttp://localhost:8080/tienda1/publico/pagar.jsp?modo=insertar&precio=2672&b1=pasar+por+caja
```

Requisições anômalas:

```
gethttp://localhost:8080/tienda1/publico/anadir.jsp?id=2&nombre=jamn+ibrico&precio=85&
cantidad=';+drop+table+usuarios;+select+*+from+datos+where+nombre+like+'%&b1=aadir+al+carrito

posthttp://localhost:8080/tienda1/publico/autenticar.jsp?modo=entrar&login=bob@<script>alert(paros)
</script>.parosproxy.org&pwd=84m3ri156&remember=on&b1=entrar
```

Todos os caracteres das requisições foram representados por números inteiros utilizando o *Tokenizer* do Keras. A construção do dicionário contendo 63 caracteres, com o espaço e um para símbolos não reconhecidos, foi feita utilizando apenas os caracteres encontrados no conjunto de dados:

```
! " # % & ' ( ) * + , - . / : ; < = > ? @ _ ~ $ | 0 1 2 3 4 5 6 7 8 9 a b c d e f g h i j k l m n o p
q r s t u v w x y z
```

4.4. Implementação das arquiteturas

Nesta seção é apresentada a implementação das arquiteturas em CLCNN e LSTM. A Figura 1 esclarece, na visão de um sistema, onde seria aplicado o detector de requisições com uma das implementações de rede neural no WAF, filtrando as requisições anômalas detectadas. Um detector eficaz impediria o maior número possível de anomalias a acessar

o servidor *web*, mas permitindo o acesso às legítimas, assim fornecendo uma camada adicional de proteção à aplicação e evitando que ela tenha de tratar essas requisições. Nas subseções a seguir, são descritas, brevemente, as implementações das duas redes neurais observando as camadas e suas definições. Tanto a LSTM quanto a CLCNN são treinadas usando a abordagem de treinamento em *batches* de 63 caracteres em 3 *epochs*, representando todos os caracteres possíveis.

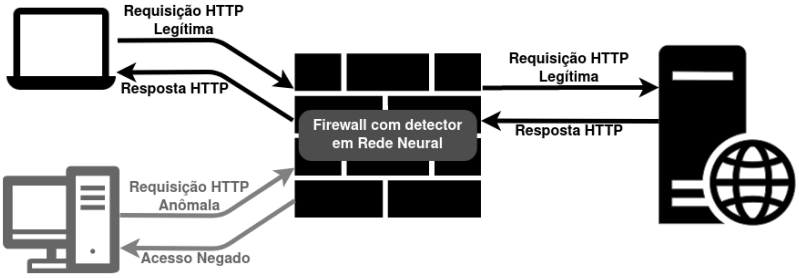

Figura 1. Diagrama de implantação do detector com Rede Neural

4.4.1. Detecção com LSTM

Na Figura 2 são apresentadas as camadas da arquitetura implementada com LSTM. A primeira camada é a de "incorporação", usada para representar partes de um texto que já foram "tokenizadas", como palavras ou caracteres representados por números inteiros, em vetores [Chollet et al. 2015]. O resultado é um vetor de valores reais que codifica o significado das partes extraídas do texto, de tal forma que é esperado que vetores próximos possuam significado similar [Teller 2000]. O tamanho de sua entrada, correspondendo ao tamanho máximo da requisição, é de 1000 caracteres. O espaço vetorial é especificado geralmente com 50, 100, 200 ou 300 dimensões [Bogale Gereme e Zhu 2020]. Nesse modelo o espaço vetorial foi, de maneira arbitrária, definido para 100 dimensões.

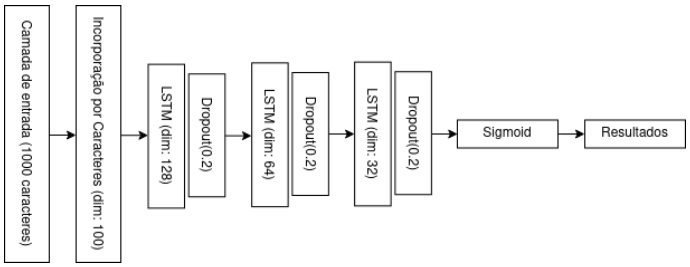

Figura 2. Camadas da arquitetura implementada com LSTM

Após a camada de incorporação são implementadas as camadas LSTM de fato. Como apresentado na seção 3.1 desse trabalho, essa arquitetura é especialmente projetada para manter, por um longo período de tempo, a correlação de informações mesmo que essas estejam espacialmente distantes na sequência recebida. Com isso a LSTM é valiosa em tarefas que tratam sequências de informações, como de NLP ou áudio, por exemplo, podendo assim obter bons resultados na detecção de requisições anômalas. Essa implementação utiliza a arquitetura clássica da LSTM [Zhang et al. 2015] e foi criada de

forma semelhanta a que [Hwang et al. 2019] utilizou para detectar acessos maliciosos a uma rede a nível de pacote.

A camada de *dropout* é utilizada para resolver o problema de *overfitting*, que ocorre quando a rede neural torna-se muito boa em classificar os registros utilizados para treiná-la, mas não os novos registros. Como apresenta [Srivastava et al. 2014], a *dropout* procura resolver isso desativando alguns neurônios, aleatoriamente, durante o treinamento para que não influenciem no resultado. Isto evita que a LSTM aprenda a classificar corretamente apenas os registros utilizados no treinamento, permitindo que generalize bem em registros que levemente desviam-se dos utilizados para treiná-lo. No Keras [Chollet et al. 2015] as entradas das unidades da camada *dropout* são definidas como zero conforme uma frequência (f) parametrizável. Nessa implementação a frequência do *dropout* foi definida como `0.2`, enquanto os outros neurônios, para compensar, aumentam o valor de entrada em escala $1/(1-f)$, a fim de não alterar o resultado da soma de todas as entradas.

Por fim, a camada *dense* é implementada com uma função de ativação[3] que é aplicada na entrada, calculando, assim, o resultado de saída do neurônio. Essa implementação conta com apenas um neurônio *dense* de saída que representa a única categoria do classificador: a confiança de que a requisição é anômala. A função de ativação utilizada nesse neurônio *dense* é a sigmoid, escolhida por ser a mais amplamente utilizada para manter os resultados no intervalo entre 0 e 1 [Sharma e Sharma 2017].

4.4.2. Detecção com CLCNN

A implementação da arquitetura em CLCNN foi inspirada na proposta por [Zhang et al. 2015] para tarefas de classificação de texto. Foram realizados ajustes na estrutura e na configuração das camadas para aproximar da arquitetura em LSTM e para melhor adaptação ao conjunto de dados. A camada de *embedding* foi reduzida de 1014 para 1000 neurônios, e a dimensão de seu espaço vetorial foi aumentada para 100. A estrutura original da CLCNN possui, após a camada *flatten* (representada pela de concatenação das saídas na Figura 3), duas camadas *dense* com função de ativação ReLU intercaladas por duas de *dropout* para evitar *overfitting*, finalizando com uma *dense* com ativação *softmax* com os resultados da predição das classes. Para simplificar, após a camada *flatten* foram removidas as camadas *dense* com ativação em ReLU e mantida apenas uma camada de *dropout* com f em `0.5`. Já a camada *dense* final foi reduzida para apenas um neurônio, representando a categoria de requisição anômala, com função de ativação *sigmoid*.

Além das camadas de entrada (*embedding*) e saída (*dropout* e *dense*), a CLCNN é formada por três outras categorias de camada: Conv1D, *Activation* e MaxPooling1D. Esses três formam as camadas da parte convolucional da implementação. A camada Conv1D recebe um *tensor* como entrada, no qual é convolvido um *kernel* de uma dimensão. Após cada camada Conv1D é inserida uma camada de ativação com a função ReLU que em alguns casos se conecta com outra Conv1D, e em outros com uma camada MaxPooling1D. A MaxPooling1D reduz a dimensão da entrada pegando apenas o maior valor de uma região definida pelo parâmetro "pool_size", aqui definido para `3`, que itera em todas as

[3] *Activation Funcion* no contexto das *Artificial Neural Networks*.

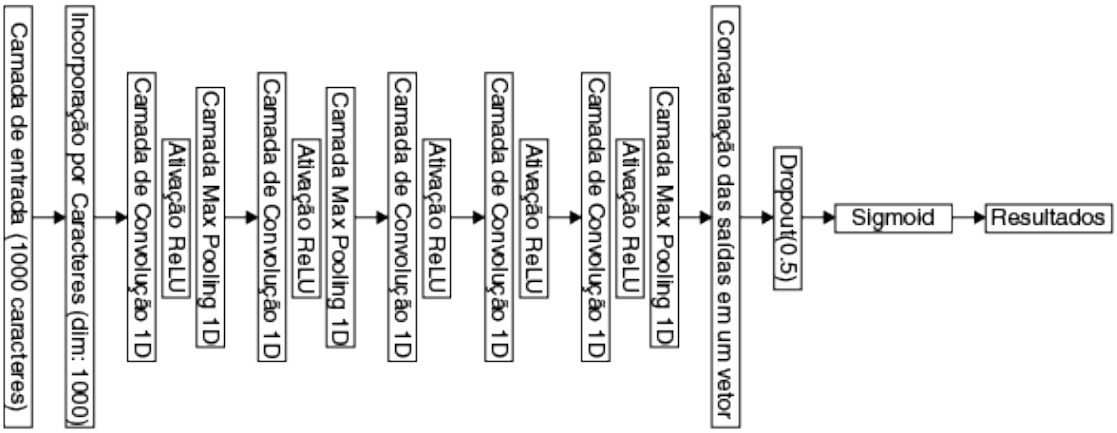

Figura 3. Camadas da arquitetura CLCNN

entradas. Essa redução de dimensão extrai apenas as partes mais notáveis da entrada, para diminuir tanto a carga computacional quando a possibilidade de ocorrer *overfitting*.

4.5. Métricas

Foi extraída a matriz de confusão (Tabela 3) para a avaliação das duas ANNs e para ser utilizada no cálculo das outras métricas: acurácia, precisão, taxa de detecção e F_1-*score*. Também foi observado o tempo de execução para implementações em LSTM e CLCNN classificar todas as requisições de teste e, com esse valor, foi calculado o tempo médio por requisição. Na matriz de confusão são separados os valores de:

- **Verdadeiro Negativo (VN):** taxa de previsões corretas para requisições benignas;
- **Verdadeiro Positivo (VP):** requisições anômalas classificadas corretamente;
- **Falso Negativo (FN):** taxa de requisições anômalas classificadas como legítimas;
- **Falso Positivo (FP):** taxa de classificação incorreta de requisições maliciosas como legítimas.

Tabela 3. Matriz de confusão

		Classificação correta	
		Legítima	Anômala
Classificação Prevista	Legítima	VN	FN
	Anômala	FP	VP

Os valores da matriz de confusão são utilizados para avaliar as ANNs através das métricas descritas abaixo, que são extensamente utilizadas na avaliação de classificadores [Liang et al. 2017, Gong et al. 2019, Rego e Nunes 2019]. Suas fórmulas e abreviações são apresentadas na Tabela 4.

- **Acurácia:** a relação entre os resultados e o real valor, sendo a razão dos acertos pelo todo;
- **Precisão:** também chamada de Valores Positivos Preditivos, é a proporção de resultados positivos que são verdadeiros positivos;

- **Taxa de detecção:** conhecida como sensitividade ou *recall*, em inglês, mede a proporção dos positivos que são corretamente identificados;
- F_1**-*score*:** é a média harmônica da precisão e da taxa de detecção.

Tabela 4. Fórmulas das métricas

Métrica	Acurácia	Precisão	Taxa de detecção	F_1-*score*
Abreviação	AC	P	TD	F_1
Fórmula	$\frac{VP+VN}{VP+VN+FP+FN}$	$\frac{VP}{VP+FP}$	$\frac{VP}{VP+FN}$	$2 * \frac{P*TD}{P+TD}$

5. Resultados e discussões

Essa seção apresenta os resultados da comparação entre os modelos clássicos da CLCNN e da LSTM. Como apresentado na Tabela 5, a CLCNN obteve melhores resultados que a LSTM na matriz de confusão. Nota-se essa vantagem principalmente nos números de Falsos Positivos onde obteve um valor 89,61% menor que a LSTM, significando que classificou, erroneamente, poucas requisições legítimas como maliciosas, não impactando negativamente os usuários do sistema. Também obteve um bom resultado de Falsos Negativos, uma redução de 51,86% em relação ao LSTM, com menos acessos anômalos sendo classificados como legítimos podendo levar a entender que manteria a aplicação mais segura detectando mais ataques caso utilizada num WAF. No número de Verdadeiros Positivos a CLCNN reconheceu 16,96% de requisições maliciosas a mais que a LSTM. Na taxa de Verdadeiros Negativos as duas implementações foram semelhantes, com a CLCNN classificando apenas 0,97% mais requisições legítimas corretamente.

Tabela 5. Matriz de confusão comparando CLCNN com LSTM

	CLCNN	LSTM	Ideal
Verdadeiro Negativo (VN)	**7137 (99.89%)**	7068 (98,92%)	7145
Falso Positivo (FP)	**8 (0.11%)**	77 (1,08%)	0
Falso Negativo (FN)	**220 (4,34%)**	457 (9,02%)	0
Verdadeiro Positivo (VP)	**4848 (95,66%)**	4611 (90,98%)	5068

Utilizando os resultados da matriz de confusão foram calculadas as métricas apresentadas na Figura 4 em que é observado, novamente, o melhor desempenho da CLCNN em relação a LSTM. Isso é visto principalmente na taxa de detecção em que a CLCNN possui uma vantagem de 4,68%, detectando um alto número de requisições anômalas (Verdadeiros Positivos) com um baixo número de Falsos Negativos, assim reforça sua superioridade no reconhecimento de requisições maliciosas. Já com a precisão, vemos que ambas são similares na alta ocorrência de Verdadeiros Positivos em relação a tudo o que foi classificado como positivo, contando com os Falsos Positivos, portanto, tanto a LSTM quanto a CLCNN possuem proporções baixas de requisições legítimas sendo erroneamente classificadas como anômalas, e ambas atingiram uma precisão acima de

98%. Observa-se com a acurácia que as duas implementações possuem bons resultados de classificação em relação ao real valor, a CLCNN se manteve a frente nessa métrica por 2,5%. Na F_1-*score*, outra métrica de acurácia em que é calculada a média harmônica entre a precisão e a taxa de detecção, a CLCNN também obteve o melhor resultado com um aumento de 3,17% em relação a LSTM.

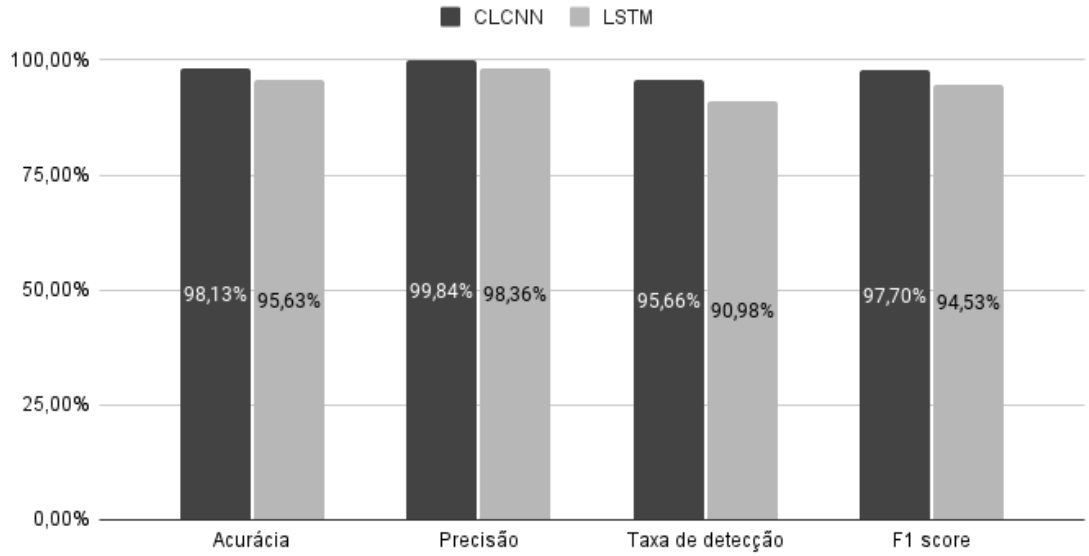

Figura 4. Acurácia, precisão, taxa de detecção e F1-score da CLCNN e LSTM

A CLCNN também possui o melhor resultado em relação ao tempo de execução, com uma média de 4,75 milisegundos por requisição. A LSTM classificou cada requisição em, aproximadamente, 11,05 milisegundos, um tempo 132% maior que a CLCNN.

6. Conclusão

A detecção de requisições maliciosas em uma aplicação *web* é uma tarefa essencial para garantir a segurança do sistema, de seus dados e usuários. As aplicações implementam normalmente medidas de proteção, seja internamente ou em WAFs, para evitar ataques como de SQLIA ou XSS, porém, em muitos casos ainda estão vulneráveis a esses ataques. Diversos estudos propõem o uso de redes neurais para detectar requisições maliciosas utilizando como base tanto a LSTM quanto a CLCNN, e com bons resultados. Entretanto, não existiam comparações entre as implementações clássicas dessas duas categorias.

Os resultados encontrados demonstram que tanto a CLCNN quanto a LSTM implementadas nesse estudo são competitivas em relação aos outros estudos. A CLCNN foi melhor que a CNN com incorporação por palavras [Zhang et al. 2017] (enquanto a CLCNN realiza incorporação ao nível de caractere), superando por 1,64% na acurácia e por 2,31% na taxa de detecção. Em relação à proposta de [Ito e Iyatomi 2018] os resultados foram semelhantes, com sua implementação da CLCNN sendo apenas 0,67% superior na acurácia que a deste estudo.

Foi realizada a comparação entre a LSTM e a CLCNN em suas implementações clássicas. Para isso, as duas arquiteturas foram implementadas e comparadas considerando métricas comumente utilizadas. Os resultados mostram que a CLCNN é superior

à LSTM em todas as métricas, e possui ótimos resultados. A principal vantagem da CLCNN se deu na taxa de detecção, em que detectou um alto número de requisições anômalas com uma taxa baixa de falsos negativos. Na precisão as duas implementações obtiveram resultados semelhantes e acima de 98%, significando que detectaram uma taxa alta de requisições anômalas com uma baixa taxa de falsos positivos. As diferenças entre a CLCNN e a LSTM na acurácia e na F_1-*score* são semelhantes, com a CLCNN sendo superior por 2,5% e 3,17%, respectivamente, e as duas possuem bons resultados reforçando que possuem uma boa taxa de acerto em relação aos falsos positivos e negativos.

Referências

Abiodun, O. I., Jantan, A., Omolara, A. E., Dada, K. V., Mohamed, N. A., e Arshad, H. (2018). State-of-the-art in artificial neural network applications: A survey. *Heliyon*, 4(11):e00938.

Al-Khurafi, O. B. e Al-Ahmad, M. A. (2015). Survey of web application vulnerability attacks. In *2015 4th International Conference on Advanced Computer Science Applications and Technologies (ACSAT)*, pages 154–158. IEEE.

Albawi, S., Mohammed, T. A., e Al-Zawi, S. (2017). Understanding of a convolutional neural network. In *2017 International Conference on Engineering and Technology (ICET)*.

Bogale Gereme, F. e Zhu, W. (2020). Fighting fake news using deep learning: Pre-trained word embeddings and the embedding layer investigated. In *2020 The 3rd International Conference on Computational Intelligence and Intelligent Systems*, pages 24–29.

Chollet, F. et al. (2015). Keras documentation. *keras.io*, 33.

Dupond, S. (2019). A thorough review on the current advance of neural network structures. *Annual Reviews in Control*, 14:200–230.

Giménez, C. T., Villegas, A. P., e Marañón, G. Á. (2010). Http data set csic 2010. *Information Security Institute of CSIC (Spanish Research National Council)*.

Goldberg, Y. (2016). A primer on neural network models for natural language processing. *Journal of Artificial Intelligence Research*, 57:345–420.

Gong, X., Lu, J., Wang, Y., Qiu, H., He, R., e Qiu, M. (2019). Cecor-net: A character-level neural network model for web attack detection. In *2019 IEEE International Conference on Smart Cloud (SmartCloud)*, pages 98–103. IEEE.

Hwang, R.-H., Peng, M.-C., Nguyen, V.-L., e Chang, Y.-L. (2019). An lstm-based deep learning approach for classifying malicious traffic at the packet level. *Applied Sciences*, 9(16):3414.

Ito, M. e Iyatomi, H. (2018). Web application firewall using character-level convolutional neural network. In *2018 IEEE 14th International Colloquium on Signal Processing & Its Applications (CSPA)*, pages 103–106. IEEE.

Liang, J., Zhao, W., e Ye, W. (2017). Anomaly-based web attack detection: a deep learning approach. In *Proceedings of the 2017 VI International Conference on Network, Communication and Computing*, pages 80–85.

OWASP (2017). Top 10-2017. *The Ten Most Critical Web Application Security Risks. OWASP™ Foundation. The free and open software security community.*

OWASP (2020). Sql injection prevention cheat sheet.

Rego, R. C. e Nunes, R. (2019). Filtro de bloom como ferramenta de apoio a detectores de ataques web baseados em aprendizagem de máquina. In *Anais do XIX Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais*, pages 85–98. SBC.

Rodríguez, G. E., Torres, J. G., Flores, P., e Benavides, D. E. (2020). Cross-site scripting (xss) attacks and mitigation: A survey. *Computer Networks*, 166:106960.

Saxe, J. e Berlin, K. (2017). expose: A character-level convolutional neural network with embeddings for detecting malicious urls, file paths and registry keys. *arXiv preprint arXiv:1702.08568*.

Sharma, S. e Sharma, S. (2017). Activation functions in neural networks. *Towards Data Science*, 6(12):310–316.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., e Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958.

Tang, R., Yang, Z., Li, Z., Meng, W., Wang, H., Li, Q., Sun, Y., Pei, D., Wei, T., Xu, Y., et al. (2020). Zerowall: Detecting zero-day web attacks through encoder-decoder recurrent neural networks. In *IEEE INFOCOM 2020*, pages 2479–2488. IEEE.

Tekerek, A. (2021). A novel architecture for web-based attack detection using convolutional neural network. *Computers & Security*, 100:102096.

Teller, V. (2000). Speech and language processing: an introduction to natural language processing, computational linguistics, and speech recognition.

Tian, Z., Luo, C., Qiu, J., Du, X., e Guizani, M. (2019). A distributed deep learning system for web attack detection on edge devices. *IEEE Transactions on Industrial Informatics*.

Torrano-Gimenez, C., Perez-Villegas, A., e Alvarez, G. (2009). A self-learning anomaly-based web application firewall. In *Computational Intelligence in Security for Information Systems*, pages 85–92. Springer.

Yin, W., Kann, K., Yu, M., e Schütze, H. (2017). Comparative study of cnn and rnn for natural language processing. *arXiv preprint arXiv:1702.01923*.

Zhang, M., Xu, B., Bai, S., Lu, S., e Lin, Z. (2017). A deep learning method to detect web attacks using a specially designed cnn. In *International Conference on Neural Information Processing*, pages 828–836. Springer.

Zhang, X., Zhao, J., e LeCun, Y. (2015). Character-level convolutional networks for text classification. *arXiv preprint arXiv:1509.01626*.