UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE COMPUTAÇÃO
CURSO DE CIÊNCIA DA COMPUTAÇÃO

LUCAS JULIANO SCHEK

APLICAÇÃO DE TÉCNICAS DE INTELIGÊNCIA ARTIFICIAL NA OTIMIZAÇÃO DE SISTEMAS DE DETECÇÃO DE INTRUSÃO EM REDES DE COMPUTADORES

TRABALHO DE CONCLUSÃO DE CURSO

MEDIANEIRA

2021

LUCAS JULIANO SCHEK

APLICAÇÃO DE TÉCNICAS DE INTELIGÊNCIA ARTIFICIAL NA OTIMIZAÇÃO DE SISTEMAS DE DETECÇÃO DE INTRUSÃO EM REDES DE COMPUTADORES

Trabalho de Conclusão de Curso apresentado ao Departamento Acadêmico de Computação da Universidade Tecnológica Federal do Paraná como requisito parcial para obtenção do título de "Bacharel em Ciência da Computação".

Orientador: Prof. Dr. Paulo Lopes de Menezes

Co-orientador: Prof. Dr. Arnaldo Candido Junior

MEDIANEIRA

2021

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Diretoria de Graduação e Educação Profissional
Coordenação do Curso de Ciência da Computação

UTFPR
UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

TERMO DE APROVAÇÃO

APLICAÇÃO DE TÉCNICAS DE INTELIGÊNCIA ARTIFICIAL NA OTIMIZAÇÃO DE SISTEMAS DE DETECÇÃO DE INTRUSÃO EM REDES DE COMPUTADORES

Por

LUCAS JULIANO SCHEK

Este Trabalho de Conclusão de Curso foi apresentado às 10:00h do dia 03 de Maio de 2021 como requisito parcial para a obtenção do título de Bacharel no Curso de Ciência da Computação, da Universidade Tecnológica Federal do Paraná, Câmpus Medianeira. O candidato foi arguido pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

Prof. Dr. Paulo Lopes de Menezes
UTFPR - Câmpus Medianeira

Prof. Dr. Arnaldo Candido Junior
UTFPR - Câmpus Medianeira

Prof. Dr. Neylor Michel
UTFPR - Câmpus Medianeira

Prof. Dr. Pedro Luiz de Paula Filho
UTFPR - Câmpus Medianeira

A folha de aprovação assinada encontra-se na Coordenação do Curso.

RESUMO

SCHEK, Lucas Juliano. APLICAÇÃO DE TÉCNICAS DE INTELIGÊNCIA ARTIFICIAL NA OTIMIZAÇÃO DE SISTEMAS DE DETECÇÃO DE INTRUSÃO EM REDES DE COMPUTADORES. 71 f. Trabalho de Conclusão de Curso – Curso de Ciência da Computação, Universidade Tecnológica Federal do Paraná. Medianeira, 2021.

A detecção de intrusões desempenha um papel importante na garantia da segurança das informações, é a principal tecnologia na identificação com precisão de vários ataques na rede. Neste artigo, exploramos como modelar um sistema de detecção de intrusão em redes baseado em aprendizagem profunda e propomos uma abordagem para a detecção de intrusão usando redes neurais recorrentes (RNN-IDS). Os resultados experimentais mostram que o RNN-IDS é muito adequado para modelar um modelo de classificação com alta precisão e que seu desempenho é superior ao dos métodos tradicionais de classificação de aprendizado de máquina.

Neste trabalho é proposto um novo IDS que consiste em uma rede neural recorrente com *Gate Recurrent Unit* (GRU) e *Long Short Term Memory* (LSTM). Para a realização dos experimentos, selecionou-se a base de dados pública NSL-KDD, foi utilizada a técnica de regularização Dropout e K-fold. Os resultados comprovaram que a LSTM e o GRU tiveram melhores acurácias em relação á outros classificadores, como: J48, BayesNet, Randodm Forest dentre outros.

Palavras-chave: Redes Neurais Artificiais, Segurança Computacional

ABSTRACT

SCHEK, Lucas Juliano. APPLICATION OF ARTIFICIAL INTELLIGENCE TECHNIQUES IN THE OPTIMIZATION OF INTRUSION DETECTION SYSTEMS IN COMPUTER NETWORKS. 71 f. Trabalho de Conclusão de Curso – Curso de Ciência da Computação, Universidade Tecnológica Federal do Paraná. Medianeira, 2021.

Intrusion detection plays an important role in ensuring the security of information, it is the main technology in accurately identifying various attacks on the network. In this article, we explore how to model a network intrusion detection system based on deep learning and propose an approach to intrusion detection using recurrent neural networks (RNN-IDS). The experimental results show that the RNN-IDS is very suitable for modeling a classification model with high precision and that its performance is superior to that of traditional machine learning classification methods.

In this article, a new IDS is proposed, which consists of a recurrent neural network with textit Gate Recurrent Unit (GRU) and textit Long Short Term Memory (LSTM). To carry out the experiments, the public database NSL-KDD was selected and the Dropout and K-fold regularization technique was used. The results proved that LSTM and GRU had better accuracy in relation to other classifiers, such as: J48, BayesNet, Randodm Forest, among others.

Keywords: Artificial Neural Networks, Computational Security

Dedico este trabalho à minha amável mãe, Mara Juliana Schek S2, pelo amor, incentivo e apoio para a conclusão desta importante etapa e por me ajudar a buscar meus objetivos com muito mais garra e dedicação.

AGRADECIMENTOS

Agradecimentos aos meus Orientadores e ao Jorge-Sensei pelo carinho e dedicação, muito obrigado!

Queria agradecer também ao meu caro amigo Talyson, passamos por várias matérias, estágios e tudo mais juntos, enfim nos formamos meu jovem! Vamos que vamos =D.

Ademais, Parabéns Lucão!

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

DDoS	Distributed Denial of Service
FN	Falso Negativo
FP	Falso Positivo
IA	Inteligência Artificial
IDS	Intrusion Detection System
IPS	Intrusion Prevention System
GRU	Gated Recurrent Unit
LSTM	Long-Short Term Memory
MLP	Multilayer Perceptron
RNA	Redes Neurais Artificiais
RNC	Redes Neurais Convolucionais
RNR	Redes Neurais Recorrentes
SI	Segurança da Informação
VN	Verdadeiro Negativo
VP	Verdadeiro Positivo

SUMÁRIO

1 INTRODUÇÃO

Atualmente, as redes de computadores atendem às necessidades de empresas e órgãos governamentais, construindo uma rede de informação especializada e complexa, que integra tecnologias desde sistemas distribuídos de armazenamento de dados até acessos remotos e serviços web. De acordo com Shah e Issac (2018) na medida em que o acesso a essas tecnologias se torna essencial, ficam mais vulneráveis do que nunca, possibilitando inúmeros caminhos ao invasor.

De acordo com o Centro de Estudos, Resposta e Tratamento de Incidentes de Segurança[1], ponto central para notificações de incidentes de segurança no Brasil, que reúne informações e estatísticas sobre a guerra cibernética, foram registrados cerca de 875 mil incidentes em 2019, sendo destes, 250 mil de nações estrangeiras, tendo como principais ataques: ataque distribuído de negação de serviço (DDoS), códigos maliciosos, invasões, tentativas de fraude (*phishing*), ataques a servidores Web, *scans* dentre outros.

É importante que uma organização monitore o fluxo da rede e detecte qualquer invasão que esteja violando as políticas da organização. Detectar atividades maliciosas com precisão requer sistemas de detecção de intrusões (IDS), que são capazes de realizar a detecção com base em assinaturas e anomalias. No entanto, os ataques de segurança tendem a ser imprevisíveis, e existem muitos desafios para desenvolver um IDS flexível e eficaz.

Após Hinton et al. (2006) propôr a teoria do *deep learning*, uma nova era na inteligência artificial se abriu e ofereceu uma maneira completamente nova de desenvolver tecnologia inteligente na detecção de invasões. Devido aos crescentes recursos computacionais, as Redes Neurais Recorrentes (RNR) e as Redes Neurais Convolucionais (RNC) recentemente geraram um desenvolvimento significativo no domínio da aprendizagem profunda (KARPATHY, 2015).

Diante deste cenário, é proposto uma abordagem de aprendizado profundo para um Sistema de Detecção de Intrusão utilizando Redes Neurais Artificiais (RNA), para que os novos ataques possam ser detectados sem a necessidade de atualização rotineira da base de dados. O objetivo é apresentar uma solução escalável e confiável, que possa prever e garantir a segurança

[1] https://www.cert.br/stats/incidentes/

e o gerenciamento em redes corporativas e governamentais com maior eficiência.

1.1 OBJETIVOS GERAL E ESPECÍFICOS

Esse trabalho teve como objetivo desenvolver um estudo comparativo entre técnicas de inteligência artificial na otimização de sistemas de detecção de intrusão em redes de computadores, avaliando a efetividade das mesmas, sendo elas: *Long short-term memory* (LSTM) e *Gated Recurrent Unit* (GRU). Esse objetivo principal pode ser dividido nos seguintes objetivos específicos:

- Realizar o levantamento dos campos do dataset relevantes na criação dos modelos;
- Criar os modelos baseados em redes neurais recorrentes;
- Comparar o desempenho dos modelos propostos;
- Avaliar o desempenho do modelo na previsão do tráfego malicioso.

1.2 JUSTIFICATIVA

O IDS é um componente crítico em relação às defesas de segurança das redes, fundamental para a maioria das organizações, tem como objetivo detectar e interromper ataques, na qual as detecções são baseadas em assinaturas ou anomalias, ambos os métodos possuem vulnerabilidades, o IDS baseado em assinatura não tem a capacidade de detectar ataques de dia zero e possuem dificuldades de encontrar variantes de um ataque que já se tem a assinatura, e o IDS baseado em anomalia recebe críticas por sua inerente propriedade de gerar uma alta taxa de alarme falso. Observa-se na literatura que muitas das técnicas de sistemas inteligentes artificiais já estão sendo aplicadas em sistemas de detecção de intrusão, explorando o poder de aprendizagem da máquina.

Lin et al. (2018) aplicaram técnicas de Redes Neurais Convolucionais (RNCs) na estimativa de melhora na precisão da detecção de intrusões para classificação de ameaças. Segundo os autores, a abordagem proposta pode aumentar a precisão da detecção de invasão de rede. No estudo realizado por Yin et al. (2017), foram utilizadas técnicas de Redes Neurais

Recorrentes (RNR) na comparação com métodos tradicionais de classificação, como J48, naive bayesian e random forest. De acordo com os autores, o desempenho da RNR obteve taxas de acurácia e taxas de detecção mais altas com uma baixa taxa de falsos positivos, os autores ainda propõem um estudo sobre a classificação de desempenho do algoritmo LSTM na detecção de intrusão.

Acredita-se que este trabalho possa comprovar que tecnologias de Inteligência Artificial são capazes de prever o tráfego malicioso a partir de um *dataset* que contém um conjunto padrão de dados a serem auditados, tais dados são pertinentes a NSL-KDD[2], que possui diversas melhorias em relação ao dataset original KDD Cup'99[3], que inclui uma ampla variedade de intrusões simuladas em um ambiente de rede militar. Assim, será possível construir um modelo preditivo capaz de distinguir entre tráfego maligno e conexões normais, visando aumentar as taxas de detecções e reduzir a quantidade de falsos positivos na rede por meio de técnicas de Aprendizado Profundo.

1.3 ESTRUTURA DO TRABALHO

Esse documento está organizado da seguinte forma. O Capítulo 2 apresenta inicialmente conceitos referentes a segurança de redes e em seguida uma breve história dos sistemas de detecção de intrusão bem como seus conceitos e aplicações. No Capítulo 3, uma introdução sobre inteligência artificial e um aprofundamento nas técnicas a serem utilizadas durante o estudo. Em seguida, são apresentados os trabalhos correlatos recentes, com o intuito de situar este trabalho no estágio atual do conhecimento. A metodologia utilizada se encontra no Capítulo 4, nele são descritas todas as etapas para o desenvolvimento do projeto. No Capítulo 5 são listados os resultados e discussões e no Capítulo 6 encontra-se a conclusão da proposta do trabalho de conclusão de curso.

[2]https://www.unb.ca/cic/datasets/nsl.html
[3]http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html

2 SEGURANÇA DA INFORMAÇÃO

Devido a extensidade dos conteúdos que podem ser abordados, neste capítulo são descritos as funcionalidades, classificações e padronizações de programas maliciosos e IDSs, destacando aspectos relacionados aos mesmos, bem como as taxonomias à serem utilizadas. Este capítulo também apresenta conceitos gerais da Segurança da Informação (SI), conforme Sêmola (2013) e Beal (2005), SI é o processo que visa proteger a informação dos perigos que podem violar a integridade, disponibilidade e confidencialidade dos dados. Esses três princípios formam a base da segurança da informação.

2.1 MODELO OSI

O modelo OSI[1] (do inglês - *Open System Interconnection*) foi criado pela ISO[2] (do inglês - International Organization for Standardization) e define como a troca de informação irá ocorrer entre os dispositivos na rede (FILHO, 2013). É um modelo de referência que provê um padrão para interoperabilidade de sistemas, ou seja, ele possibilita a comunicação entre *hardware*, *software* e tecnologias de redes distintas.

Como pode ser visto na Figura 1, o modelo OSI é composto por 7 camadas, pelo PDU (do inglês - *Protocol Data Unit*) que se refere à designação dos elementos (dados, segmentos, pacotes, quadros e bits) encontrados em cada camada, o que facilita a compreensão e o gerenciamento de incidentes. Uma descrição mais detalhada das camadas pode ser vista à seguir:

- Aplicação: nesta camada estão as aplicações utilizadas por usuários e seus respectivos protocolos. Então, todos os protocolos com os quais o usuário venha a ter contato direto estarão na camada 7, como exemplo usuários que navegam na Internet e usam, para isso,

[1] https://standards.iso.org/ittf/PubliclyAvailableStandards/
[2] https://www.iso.org

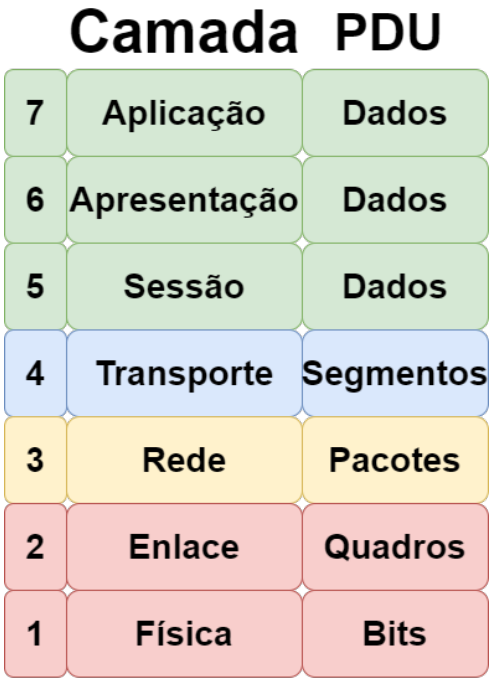

Figura 1 – Modelo OSI e PDU

Fonte: Adaptado de (FILHO, 2013)

o HTTP. Vale citar outros protocolos, como: DHCP, SMTP, DNS e FTP;

- Apresentação: esta camada não possui protocolos de rede, é responsável por três ações sobre os dados: compressão, criptografia e conversão de padrões. Utiliza os protocolos SSL e TLS (que não são protocolos de rede), são responsáveis pela criptografia, transformando HTTP em HTTPS etc;
- Sessão: é onde ocorre o estabelecimento de sessão entre aplicações, não utiliza protocolos de rede, e sim de APIs (do inglês - *Application Program Interfaces*) como NetBIOS e o RPC (do inglês - *Remote Procedures Calls*). As aplicações utilizam esta camada para trocarem informações sobre como deverão trafegar dados entre si, serviços como FTP e SSH se autenticam por meio dela;
- Transporte: essa camada é responsável por rastrear e dividir os dados em segmentos menores, identificar unicamente o processo de cada aplicação com um número de porta, intercalando cada um deles a fim de que todos processos utilizem a rede, além de detectar e corrigir possíveis erros. Os protocolos utilizados são o TCP e o UDP, responsáveis pelo transporte, provendo serviços orientados a conexão (*handshake*) e serviços não orientados a conexão respectivamente.
- Rede: responsável pelo roteamento de pacotes, os protocolos que atuam nessa camada realizam o estabelecimento da rede, realizando ações como: endereçamento, descobrir

e selecionar o melhor caminho, conectar redes diferentes, encaminhar os pacotes até o destino além de enviar mensagens de erro e de controle. Protocolos comuns: IPv4 e IPv6, ICMP e NAT;

- Enlace: nesta camada é montado os frames, onde utiliza endereçamento MAC, é a primeira camada que possui protocolos de rede, gerencia a transferência eficiente de quadros no meio físico controlando e evitando erros na camada física. Protocolos comuns: ARP, CDP e HDLC;
- Física: esta camada é responsável pelos elementos físicos, como: cabos de rede, conectores e placas eletrônicas. Como visto, o PDU dessa camada é o bit, então pode-se ter energia elétrica, sinais luminosos, sinais sonoros e outros métodos de designação de 0 e 1.

2.2 AMEAÇAS E VULNERABILIDADES

Programa malicioso, código malicioso, *software* malicioso, ou simplesmente *malware*, são expressões que se referem a um *software* que é inserido em um sistema de forma ilícita, com a intenção de comprometer a confidencialidade, integridade ou a disponibilidade dos dados da vítima. Quando um *malware* infecta um computador, ele ativa seu código malicioso (carga útil), procurando meios de replicar-se e realizando tarefas que podem causar algum prejuízo para o usuário.

De acordo com Brown e Stallings (2013), códigos maliciosos possuem características próprias que os definem e os diferenciam dos demais tipos, como forma de obtenção, forma de instalação, meios usados para propagação e possíveis ações maliciosas. Definir e identificar essas características têm se tornado tarefas cada vez mais difíceis, devido às diferentes classificações existentes e ao surgimento de variantes que mesclam características dos demais códigos. Nos capítulos seguintes, faz-se um levantamento dessas várias categorias de *malware*.

2.2.1 Propagação de Vírus e *Worms*

Nesta primeira categoria de *malware*, se encontram os vírus e os *worms*, termos gerais utilizados para descrever programas que são secretamente anexados por eles mesmos por meio de um legítimo "transportador", que, nesse caso, pode ser um documento ou programa (DIOGENES; MAUSER, 2015). O primeiro vírus criado era chamado de Creeper, foi criado por Robert Thomas em 1971 sendo a forma experimental de criar um programa que se auto multiplicava, ao ser executado exibia a seguinte mensagem: "Eu sou o 'Creeper', pegue-me se puder!". Um outro programa chapado Reaper, foi criado para remover o "Creeper", assim, pode-se dizer que foi o primeiro antivírus. Outros exemplos importantes de vírus que podem ser citados são: o *spyware*, que viola a privacidade das informações do usuário e o *keylogger*, projetado para registrar a digitação de um usuário e, comunicá-las a um atacante remoto.

Segundo Brown e Stallings (2013), um vírus de computador é um tipo de software que pode "infectar" outros programas ou até mesmo qualquer tipo de conteúdo executável, modificando-os. Um vírus que se liga a um programa executável pode fazer qualquer coisa que o programa tenha permissão de fazer, ele executa secretamente quando o programa hospedeiro é executado, e, uma vez em execução, o código do vírus pode realizar qualquer função que seja permitida pelos privilégios do usuário corrente, como apagar arquivos e programas. Um exemplo de classificação dos mais diferentes tipos de vírus segundo Aycock (2013) são: vírus de *boot*, macros, cifrado, camuflado, polimórfico, metamórfico e infectante de arquivo.

Um *worm* é um programa que procura ativamente por máquinas para infectar e, então, cada máquina infectada serve como uma plataforma de lançamento automatizada para ataques à outras máquinas (BROWN; STALLINGS, 2013). Diferente dos vírus comuns, os *worms* podem se auto-replicar sem a necessidade de infectar arquivos legítimos, criando cópias funcionais de si mesmos e se espalhando por redes de computadores e por portas USB. Segundo o relatório de Ameaças à Segurança na Internet de 2019 [3], as formas mais comum de propagação são:

- Propagação por e-mail: esse tipo de ataque ainda é bastante comum nos dias de hoje, tal técnica comumente chamada de *Phishing*, consiste na propagação do código via e-mail, esse código malicioso é executado quando o anexo é visualizado;
- Compartilhamento de Arquivos: um *Worm* cria uma cópia de si mesmo e infecta outros arquivos, da mesma forma que um vírus;
- Execução Remota: utiliza de recursos de execução remota, ou explorando uma folha de programação em uma rede para subverter suas operações;

[3] https://www.symantec.com/security-center/threat-report

Worms são escolhidos pela sua habilidade de se espalhar sem ação do usuário, são usados para entregar outros tipos de *malware*, como *bots*, em seu *payload*. Existem também os que tem por características a tentativa de se ocultarem e de manterem o acesso secreto ao sistema, nesta categoria tem-se os Cavalos de Troia, *rootkits* e *backdoor*. *Worms* não são particularmente furtivos, são tipicamente indiscriminados em seus ataques, sondam todos os potenciais alvos na esperança de comprometer muitos alvos rapidamente. Alguns *worms* são fáceis de detectar devido ao alto tráfego de rede que geram.

2.2.2 Carga Útil

Tão logo o *malware* esteja ativo no sistema, a próxima preocupação é quais ações serão executadas, isto é, qual é a carga útil *(payload)* que ele carrega (BROWN; STALLINGS, 2013). Esses *payloads* podem ser programados para: destruir os dados, enviar *spams*, roubar informações, monitorar informações e sequestrar informações.

Um tipo de *malware* que ficou muito popular nos últimos anos, é um software que realiza o bloqueia do acesso ao computador, muitas vezes têm como carga útil um algoritmo de criptografia. Apesar de este tipo de código malicioso ter se popularizado recentemente, o primeiro *ransomware* chamado de PC Cyborg foi registrado em meados de 1989 (DIOGENES; MAUSER, 2015). Outros exemplos bem conhecidos como CryptoLocker ou WannaCry tendem a ser oportunistas e indiscriminados, para invadir seus alvos, eles confiam na automação simples, como anexos enviados para um grande número de possíveis vítimas por e-mail.

A próxima categoria de carga útil é aquela cujo *malware* é capaz de se comunicar com os invasores que o colocaram em sua máquina. Da mesma forma que acontece com o *Worm*, o *bot* pode ser um programa independente, agindo e se propagando através do seu computador, desta forma, ele cria suas redes e espalha conteúdo perigoso através dela.

Este tipo de ameaça leva o nome *bot* por se parecer com um robô, pode ser programado para realizar tarefas específicas dentro do computador do usuário afetado. Esse tipo de *malware* pode ser muito perigoso, pois ele pode captar dados bancários, enviando-os para o invasor, seu computador se torna um veículo de mensagens perigosas, como *spam* e *phishing*, podendo ser a causa de problemas para outros usuários na Internet.

Botnet (ou rede de robôs), também conhecido como exército de zumbis, é uma rede composta por um grande número de computadores que foram infectados por *bots* para atender

aos comandos do *hacker* que o criou (DIOGENES; MAUSER, 2015). Com o controle de centenas ou mesmo milhares de computadores, as *botnets* são geralmente usadas para enviar *spam* ou vírus, roubar dados pessoais ou executar ataques de DoS.

Segundo Basta et al. (2014), um ataque de Negação de Serviço (DoS) é o tipo de ataque em que o invasor faz com que os recursos computacionais dos equipamentos fiquem sobrecarregados, de forma que os equipamentos fiquem indisponíveis, rejeitando as requisições normais oriundas da rede.

Uma ramificação do DoS é a Negação de Serviço Distribuído (DDoS), na qual se têm uma distribuição sincronizada de requisições tendo um determinado endereço IP como alvo (DIOGENES; MAUSER, 2015). Várias requisições serão geradas para afetar de uma forma mais rápida o *host* de destino e fazer com ele pare de responder devido a exaustão de recursos. Os ataques DoS são bem mais fáceis de se evitar com algumas regras no IDS, enquanto o DoS é feito por apenas um invasor que envia vários pacotes, o DDoS é distribuído, o que torna o combate difícil, e as dificuldades são, sobretudo, *IP spoofing*, similaridade entre pacotes legítimos e falsos, prontidão e performance dos sistemas.

2.3 *PENTEST*

Também conhecido como teste de intrusão ou teste de penetração, é o ato de encontrar potenciais vulnerabilidades em um sistema, servidor ou, de forma geral, em uma estrutura de rede. *Pentest* é a abreviação de *Penetration Test* (Teste de Penetração, em tradução literal) e é algo comum nos dias de hoje, essas avaliações são úteis para entender como um ataque é realizado e para validar a eficácia dos mecanismos de defesa e dos servidores por trás deles, muitas organizações contratam empresas de segurança que fornecem esse tipo de serviço (DIOGENES; MAUSER, 2015). Na figura 2 pode-se ver a representação de um processo de *pentest*.

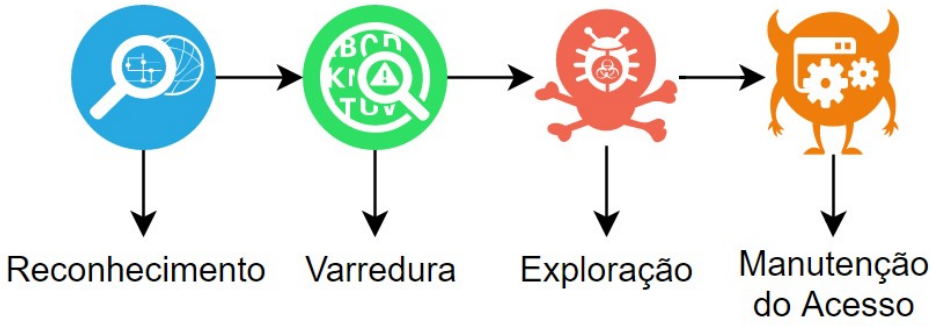

Figura 2 – Etapas de um Pentest

Fonte: Autoria Própria

2.3.1 Reconhecimento e Varredura de Vulnerabilidades (*footprinting*)

O *footprinting* também conhecido como reconhecimento, é a arte de obter informações sobre um sistema alvo usando táticas "seguras", sem perigo de detecção, e que pode dar muitas informações, tais como: sistema operacional utilizado, portas abertas, sistemas de defesa, mapa da rede, informações gerais do alvo e possíveis vulnerabilidades. O *footprinting* é a primeira etapa em um processo de *pentest* e pode ser realizado das mais variadas formas (DIOGENES; MAUSER, 2015):

- Motores de Busca: é a atividade de usar recursos dos motores de busca para recolher informações sobre o alvo, pode-se usar como exemplo, o recurso de "cache", que permite que tenha-se acesso às páginas que já foram tiradas do ar, ou realizar uma busca utilizando um comando *filetype* no Google, que busca por determinado tipo de arquivo como pdf, doc, txt, entre outros. O Google possui diversos recursos que podem ser utilizados durante um teste de invasão, pois permite acesso a todo e qualquer tipo de informação que esteja pública;
- Redes Sociais: esta forma de *footprinting* busca recolher informações pessoais do alvo para possíveis ataques de força bruta em senhas e para possíveis ataques de engenharia social;
- Sistema de Nomes de Domínio (DNS) o *footprinting* de DNS permite obter informações sobre as zonas de DNS, tais como: Endereço de Protocolo da Internet (IP), pesquisas de DNS, extensões de servidores de e-mail e etc. Estes dados permitem determinar os usuários-chave na rede e executar outros ataques;
- Rede: no reconhecimento de vulnerabilidades de rede é preciso reunir informações básicas e importantes sobre o alvo, como o endereço de IP, que dá uma ideia sobre a forma

como a rede é e ajuda a identificar a topologia da rede, além de dispositivos de controle de acesso e sistema operacional usado na rede alvo. Pode-se ainda utilizar a ferramenta Shodan para encontrar tipos específicos de computadores conectados à Internet usando uma variedade de filtros;

- Engenharia Social: A engenharia social usa a influência e a persuasão para enganar as pessoas e convencê-las de que o engenheiro social é alguém que na verdade ele não é (MITNICK; SIMON, 2003). Este tipo de técnica busca obter informações confidenciais como obter acesso não autorizado ao sistema, roubo de identidade, espionagem industrial, com ou sem o uso da tecnologia.

É importante ressaltar que o processo de *footprinting* é a arte de obter informações sobre um sistema alvo usando táticas seguras, sem tentar ultrapassar controles de segurança, e que, quanto mais informações o invasor possuir, maior a probabilidade de acesso ao sistema alvo. Tal processo geralmente é feito através de ferramentas como: Maltego[4], Shodan[5], Wikto[6] e Nmap[7].

2.3.2 Exploração

Nessa fase, o PenTester (profissional que vai realizar o teste de penetração) tendo conhecimento das portas, *softwares* instalados e sistemas ativos, inicia o processo de análise de vulnerabilidades utilizando *scanners*, banco de dados de vulnerabilidades entre outras formas com o objetivo de auxiliar o PenTester a identificar falhas nos ativos do cliente que possam a vir, através da execução de *exploits* (códigos específicos para exploração de falhas) identificar quais serviços estão vulneráveis, e que tipo de informação ou controles podem ser obtidos.

Após as vulnerabilidades serem exploradas, assim que o software é invadido, pode-se executar ataques SQL, interromper o serviço ou mesmo executar outro programa que recebe comandos remotamente. Na exploração do sistema, utiliza-se de *exploits* em sistemas vulneráveis (as vezes utilizando ferramentas como o Metasploit[8]) na tentativa de acessar o sistema do cliente. Nessa fase também se faz o "comprometimento´"´ das informações e a documentação do que foi comprometido, para que o cliente tenha ciência das brechas e do risco

[4]https://www.paterva.com
[5]https://www.shodan.io/
[6]https://sectools.org/tool/wikto/
[7]https://nmap.org/
[8]https://www.metasploit.com/

que corre se deixar tais brechas abertas.

2.3.3 Manutenção de Acesso

Durante a fase de pós-exploração, são reunidas informações sobre o sistema invadido, procura-se por arquivos importantes, utiliza-se de técnicas para elevar os privilégios onde forem necessários, criação de *backdoors* para posteriores acessos ao sistema, e assim por diante. Por exemplo, pode-se utilizar ainda uma máquina explorada para atacar sistemas não disponíveis antes, simplesmente pivotando dentro deles.

Após identificar todas as falhas e vulnerabilidades, a equipe de *pentest* coleta as evidências das falhas de segurança e com base nessas informações, é gerado um relatório completo indicando os pontos vulneráveis de todos os elementos do alvo, como: falhas na rede, *softwares* mal configurados e desatualizados, falta de elementos de segurança e etc. Ao contrário de uma invasão real, o processo de *pentest* não causa nenhum dano à empresa, ela tem total ciência da varredura que será feita em seu alvo e concede autorização para tal. O objetivo é que se tenha acesso ao que representa um risco de segurança e que os problemas sejam corrigidos, e assim, evitando possíveis intrusões e danos para a empresa.

2.4 ATAQUES DE INTRUSÃO

Como pode ser visto na Figura 3, uma intrusão real se compara muito com um *pentest*, mas, uma vez que um atacante termina seu trabalho, todas as pistas e rastros que possam ser usadas para chegar até ele são eliminadas, tal etapa é chamada de *Covering Tracks* (do inglês - Cobertura de Rastros) e é feita da seguinte forma: limpando *logs*, modificando *logs* e arquivos de registro, danificando os servidores e arquivos criados.

Quando a cobertura de rastros acontece, se torna muito difícil rastrear os invasores, uma vez que sobram poucos dados a serem auditados. De forma a evitar prejuízos e a detectar possíveis invasores já na etapa de reconhecimento, utiliza-se de ferramentas de detecção de intrusão.

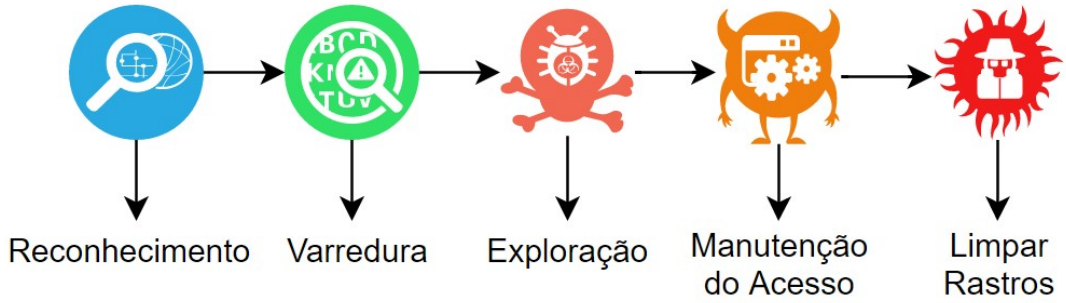

Figura 3 – Etapas de um ataque Hacker

Fonte: Autoria Própria

2.5 DETECÇÃO DE INTRUSÃO

Uma intrusão é quando o indivíduo acessa um serviço na qual não esta autorizado, desde curiosos a indivíduos que tentam ler dados privilegiados, executar modificações não autorizadas e/ou destruir dados do sistema (BROWN; STALLINGS, 2013).

2.5.1 Sistema Especialista em Detecção de Intrusão

Intrusos tipicamente seguem padrões de comportamento reconhecíveis, já que visam obter acesso a um sistema, são utilizados programas maliciosos para realizar o mapeamento da rede, identificar portas e serviços potencialmente vulneráveis dentre outros.

Com o objetivo de inspecionar dados e verificar a existência de fraudes ou erros, as empresas começaram a discutir em meados da década de 80 sobre o que deveria ser detectado e como proteger os ativos das empresas. Entre 1984 e 1986, Dorothy Denning e Peter Neumann pesquisaram e desenvolveram o primeiro modelo de um IDS em tempo real, nomeado de Sistema Especialista em Detecção de Intrusão (BRUNEAU, 2001), um sistema baseado em regras e treinado para detectar atividade maliciosa conhecida.

2.5.2 Sistema de Detecção de Intrusão

Conceitualmente, um Sistema de Detecção de Intrusão (IDS) refere-se a um mecanismo capaz de monitorar, identificar e notificar a presença de atividades maliciosas e não autorizadas. Segundo Brown e Stallings (2013), um IDS compreende três componentes lógicos principais:

- Sensores: responsáveis pela coleta de dados. Tipos de entrada de dados para um sensor incluem pacotes de rede, arquivos de registro e conjuntos de eventos de chamadas do sistema, os sensores coletam e transmitem essas informações ao analisador;
- Analisadores: recebem entradas de um ou mais sensores ou de outros analisadores. São responsáveis por determinar se tais dados são benignos ou malignos e a saída produzida por esse componente é a indicação de que uma intrusão ocorreu;
- Interface de usuário: habilita ao usuário ver a saída do sistema ou controlar o comportamento do sistema. Pode corresponder a um componente de gerenciamento, direção ou console.

A Figura 4 representa a classificação do IDS, baseado na arquitetura, método e ação tomada após detecção.

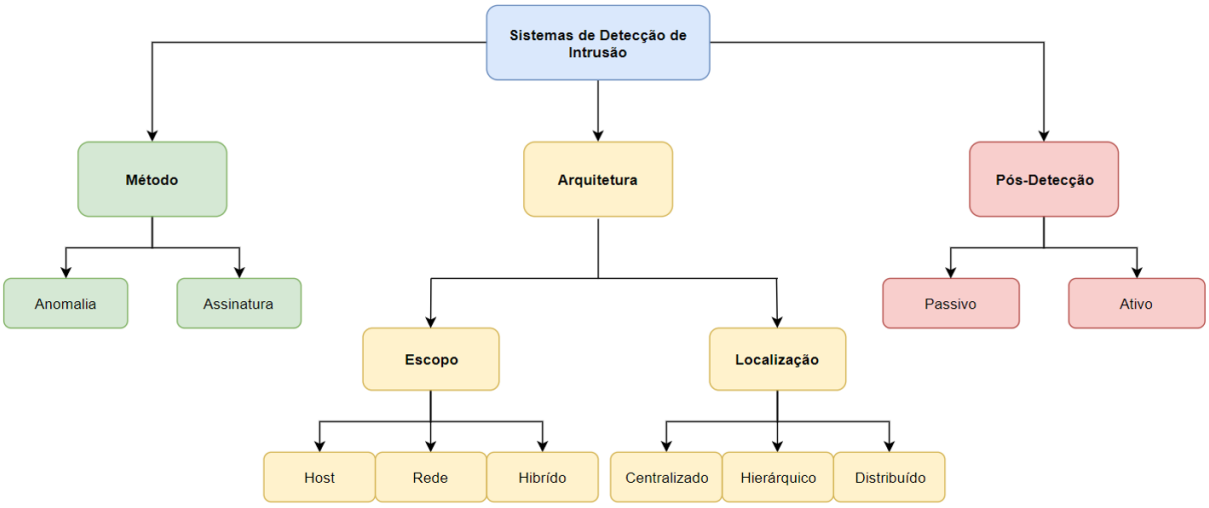

Figura 4 – Estrutura de um IDS

Fonte: (SILVA, 2005)

2.5.3 Tipos de Sistemas de Detecção de Intrusão

Os IDSs podem ser categorizados em três grupos em relação ao escopo e fonte de informações analisadas (BROWN; STALLINGS, 2013), sendo elas:

- IDS Baseado em Rede (NIDS): este tipo de IDS monitora o tráfego de rede em um segmento, e analisa a rede e a atividade dos protocolos para identificar comportamentos suspeitos. São compostos por um módulo de captura de pacotes que trafegam na rede, normalmente denominados *sniffers*;
- IDS Baseado em *host* (HIDS): sistemas que monitoram e procuram por intrusões em um único *host*;
- IDS Híbrido: Sistemas híbridos utilizam como fonte de informações de análise e avaliação tanto características relativas a um computador específico, quanto pacotes transmitidos e recebidos em rede.

2.5.4 Formas de Detecção

O método de detecção a ser empregado é um componente importante na construção de IDSs, esses métodos definem formas de detecção por anomalia aplicando métodos estatísticos sobre a utilização do sistema e detecção por assinatura, onde o monitoramento é baseado em padrões de ataques conhecidos (BROWN; STALLINGS, 2013).

- Baseado em Anomalia: é um sistema mais dinâmico, e irá procurar por possíveis ataques com base no comportamento esperado da rede, vai gerar um alarme sempre que o tráfego for anormal ao padrão, resultando em um maior número de falsos positivos;
- Baseado em Assinatura: segundo Diogenes e Mauser (2015) o sistema tem uma base de dados local que é usada para analisar o tráfego e verificar se os padrões de acesso, protocolos, portas e outras informações pertinentes à camada de aplicação batem com os dados residentes no banco de dados de assinatura, como esse tipo de analise é predefinido, possui um número reduzido de falsos positivos.

Embora o comportamento típico de um intruso seja diferente do comportamento típico de um usuário autorizado, há uma sobreposição nesses comportamentos (BROWN; STALLINGS, 2013). Assim, uma interpretação mais ampla do comportamento de intrusos,

que identificará mais intrusos, também resultará em vários Falsos Positivos (FP) ou usuários autorizados identificados como intrusos. Por outro lado, uma tentativa de limitar FP por meio de uma interpretação mais rígida do comportamento de intrusos resultará em vários Falsos Negativos (FN) ou intrusos não identificados como intrusos.

2.5.5 Modelo de Utilização

Após a detecção de uma tentativa de invasão, o IDS pode ser configurado para exercer medidas pró-ativas ou medidas passivas (SILVA, 2005):

- Ativa: um IDS ao detectar um padrão de ataque, é capaz de impedir uma ameaça detectada imediatamente no sistema ameaçado para protegê-lo de forma automática. Esta forma de utilização tem denominação própria, sistema de prevenção de intrusão (IPS). Embora seja um modelo extremamente interessante, é importante uma padronização adequada nos ambientes protegidos a fim de minimizar falsos positivo;
- Passiva: um IDS passivo é responsável pelo monitoramento do tráfego que passa por suas interfaces, assim identificando potenciais ataques ou anormalidades. Com base nisso, gera *logs* para o administrador, assim podendo ou não tomar alguma decisão.

É importante observar que os IDS ativos são mais precisos e não devem detectar falsos positivos, visto que a implementação de regras equivocadas resultará no descarte do tráfego seguro, enquanto o IDS passivo apenas criará muitos relatórios de falsos positivos.

2.5.6 Estudo de caso: Snort

O Snort é uma ferramenta robusta, capaz de realizar análises em tempo real de todo o tráfego da rede e buscar por padrões que identifiquem ataques, como os de força bruta, *buffer overflow*, *port scans*, ataques de brechas do protocolo SMB, *OS fingerprint*, *worms* entre outros. Segundo a documentação do Snort[9], a ferramenta pode ser executada de 3 modos diferentes:

- *Sniffer*: Captura os pacotes e os exibe no *console*;

[9] https://www.snort.org/documents

- *Packet logger*: Registra os pacotes capturados no disco rígido;
- *Network intrusion detection system*: Monitora todo o tráfego da rede e age com base nas regras definidas pelo usuário.

Desenvolvido inicialmente por Martin Roesch em 1998, comprado pela Cisco em 2013, se tornou um padrão de fato para os IDS na última década, com mais de 4 milhões de *downloads* é extensivamente implementado e investigado em estudos de pesquisa, se encontra em sua versão 3.0 lançada em agosto de 2018 com um grande avanço em relação a 2.9, com um sistema *multithread* por padrão, é altamente flexível e dinâmico, permite implementações de regras em tempo real, possui vasto apoio da comunidade e está em constante atualização.

2.5.7 Estudo de caso: Suricata

Suricata é um IDS de código aberto, capaz de inspecionar o tráfego de rede usando uma linguagem de assinatura e de regras poderosas e abrangentes, possui suporte a *scripts* Lua para a detecção de ameaças complexas. Lançado em 2010, é desenvolvido e mantido pela *Open Information Security Foundation*, teve como sua grande sacada a configuração de fluxos separados para processadores *multithread*, equilibrando a carga de processamento em cada processador, oferece maior velocidade e eficiência na análise de tráfego de rede, o mecanismo é construído em torno de uma base de código moderna, limpa e altamente escalável.

Há suporte nativo para aceleração de *hardware* de vários fornecedores, atualmente se encontra em sua versão 4.1.2 lançada em dezembro de 2018, permitindo a análise dos protocolos SMBv1/2/3, NFSv4, kerberos, FTP, DHCP, IKEv2, dentre outras melhorias. Sua estrutura se divide em:

- Decodificador de pacotes: O pacote é decodificado e então é gerado uma representação interna do pacote, permitindo que diferentes funções sejam chamadas para análise;
- Motor de Detecção: Detecta atividades maliciosas e aplica regras apropriadas;
- *Logs* e Alertas do Sistema: Gera alertas das atividades de intrusão;
- Módulos de Saídas: Salva e organiza as saídas geradas pelos *logs* no formato JSON, facilitando a análise.

2.5.8 Comparação SNORT X SURICATA

Diante das variadas opções de soluções IDS existentes, na execução do presente estudo, foi utilizado o Snort V. 3.0.0 e o Suricata V. 4.1.2, ambos considerados as ferramentas de código aberto mais populares existentes. A análise pode ser vista na Tabela 1.

Tabela 1 – Comparação de Funcionalidades

Aspecto	**Suricata**	**Snort**
Versão	4.1.2	3.0.0
Tipo de Sistema	NIDS	NIDS
Plataformas Suportadas	Multi	Multi
Relatórios	Sim	Sim
Alertas	Sim	Sim
Analise em tempo real	Sim	Sim
Documentação	Razoável	Ótima
Suporte a Interface Gráfica	Sim	Sim
Multithread	Sim	Sim
Aceleração de Hardware	Sim	Sim
Suporte IPV6	Sim	Sim
Pré-Processadores	Não	Sim
Suporte regras VRT	Parcial	Total

Fonte: Autoria Própria

Como pode ser visto na Tabela 1, o Snort e o Suricata apresentam algumas características em comum, dentre elas está o Sistema de Detecção de Intrusão baseados em Rede, ambos multiplataformas, suportam a tecnologia *Multithread* por padrão, IPV6, disparo de alertas, análise em tempo real, relatórios, suporte a interface gráfica e aceleração de *hardware*.

Com o novo pré-processador OpenAppID que a Cisco recentemente tornou *open source*, o Snort é capaz de realizar funções específicas e cruciais para a eficiência do mesmo, como, por exemplo, detectar *portscans*, detectar padrões de ataques mais complexos e mecanismos para remontar sequências de pacotes fragmentados. Ao invés de regras de pré-processador, o Suricata possui vários arquivos de regras para eventos definidos pelos decodificadores, mecanismo de fluxo, analisador de HTTP, etc. Em relação ao Snort, o Suricata possui menos suporte e operações mais complexas quanto ao desempenho.

3 INTELIGÊNCIA ARTIFICIAL

Neste capítulo são apresentadas o que são técnicas de Inteligência Artificial (IA), e dentre elas, é dado maior ênfase para técnicas de Redes Neurais Recorrentes (RNR). Este capítulo é utilizado para mostrar como a inteligência artificial têm se tornado fundamental para otimização de IDS's. Por fim, são apresentados os trabalhos correlatos, que envolvem técnicas de inteligência artificial aplicadas em sistemas de detecção de intrusão.

3.1 INTELIGÊNCIA ARTIFICIAL E APRENDIZADO DE MÁQUINA

O conceito de inteligência artificial veio ajudar os especialistas no desenvolvimento de sistemas de computadores que tenham características de inteligência. Sistemas que exibem características, as quais se relacionam com a inteligência no comportamento do homem, por exemplo: a capacidade de aprender, raciocinar e resolver problemas (FERNANDES, 2003). Tal capacidade está relacionada ao conceito de Aprendizagem de Máquina.

A AM é um processo de indução de hipóteses a partir de uma experiência, capaz de realizar atividades como memorizar, observar e explorar situações para aprender fatos, melhorar habilidades motoras e cognitivas por meio de práticas que organizam o conhecimento, resultando em informações (FACELI et al., 2011).

A tecnologia de AM capacita muitos aspectos da sociedade, desde buscas na web, filtragem de conteúdo em redes sociais, detecção do uso fraudulento de cartões de crédito, recomendações sobre produtos em lojas *online* e está cada vez mais presente em produtos de consumo, como câmeras e celulares (HINTON et al., 2015). Segundo Norvig e Russell (2013) as três principais formas de aprendizagem são: supervisionada, não supervisionada e por reforço.

3.2 REDES NEURAIS ARTIFICIAIS

O trabalho em Redes Neurais Artificiais (RNA), desde sua concepção, foi motivado pelo reconhecimento de que o cérebro humano processa informações complexas de modo diferente e mais efetivo, quando comparado a computadores digitais convencionais (HAYKIN, 2011). Segundo Haykin (2011) uma RNA é um processador paralelamente distribuído constituído de unidades de processamento simples, que pode armazenar o conhecimento experimental, adquire conhecimento por meio de um processo de aprendizagem e armazena o conhecimento adquirido em pesos sinápticos, gerados pela conexão entre os neurônios.

A aplicabilidade de RNAs é muito ampla, pois esta tecnologia pode ser utilizada para: classificação, ajuste de dados, reconhecimento de padrões e processamento de séries temporais. Segundo Hinton et al. (2015), exemplos de aplicações são: simulação de voos, reconhecimento de voz, predição de ações, projeto de próteses, compressão de voz e tradução de idiomas . O poder de processamento das RNAs foi motivação de diversos trabalhos que contribuíram para a evolução dos conceitos e técnicas na área. A seguir, pode-se ver o funcionamento do neurônio biológico e do neurônio artificial, unidade de processamento das RNAs.

3.2.1 Neurônio Biológico

O neurônio é a unidade básica do cérebro humano, sendo uma célula especializada na transmissão de informações, pois nelas estão introduzidas propriedades de excitabilidade e condução de mensagens nervosas. Na Figura 5, tem-se a representação de um neurônio biológico.

O cérebro é formado por bilhões de neurônios, com centenas de bilhões de conexões entre eles, formando uma enorme rede de comunicação, a rede neural. Cada neurônio possui um corpo central, diversos dendritos e um axônio. Segundo Haykin (2011), os dendritos recebem sinais elétricos de outros neurônios através das sinapses, que constituem o processo de comunicação entre neurônios. O corpo celular processa a informação e envia para outro neurônio através do axônio. A ideia de Haykin (2011) era simples: “Se redes neurais formam a inteligência humana, vamos reproduzir isso e criar Inteligência Artificial”.

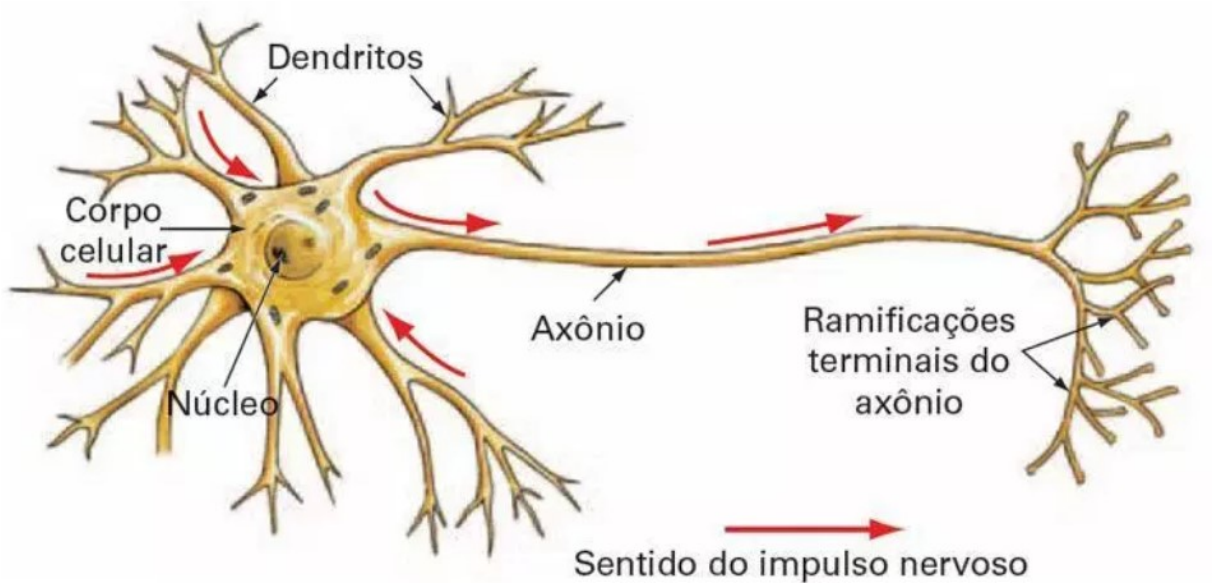

Figura 5 – Representação Simplificada do Neurônio Biológico

Fonte: Adaptado de (NORVIG; RUSSELL, 2013)

3.2.2 Modelo Neurônio Artificial - Perceptron

Um neurônio artificial é um modelo matemático que tenta representar de maneira simplificada o conhecimento sobre redes neurais biológicas. A Figura 6 faz a representação matemática de um modelo de neurônio, no qual os sinais de entrada são $x_1, x_2, x_3, ..., x_n$, os pesos das conexões sinápticas são representados por $w1, w2, ..., wn$, o *bias* é representado por b, $\sum$ o somatório e φ a função de ativação.

A função básica do neurônio artificial, é realizar o somatório, ponderado por fatores denominados pesos sinápticos, dos elementos do vetor de entrada e aplicar este resultado como entrada de uma função não linear denominada função de ativação. A saída y do neurônio a partir do modelo ilustrado na Figura 6 pode ser calculada pela Equação 1:

$$y = \varphi(\sum_{i=1}^{n} x_i w_i + b) \tag{1}$$

A função de ativação é um componente matemático incluído na estrutura de redes neurais artificiais a fim de permitir a solução de problemas complexos, e a escolha dessas funções definem como devem ser os dados de entrada. A função de ativação φ tem, tipicamente, o papel de limitar a amplitude da saída do neurônio a um intervalo limitado como [0,1] ou [-1,1].

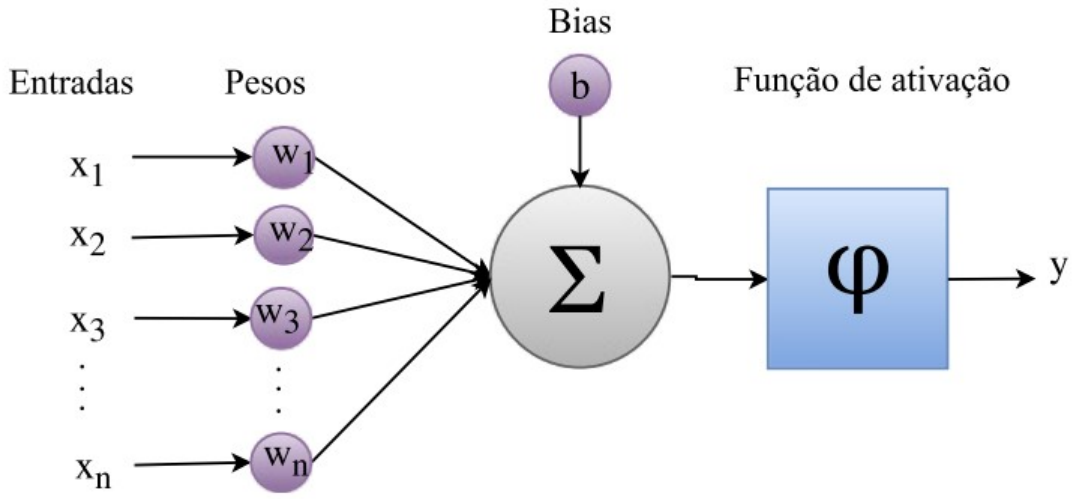

Figura 6 – Modelo Neurônio Artificial

Fonte: Adaptado de (HAYKIN, 2011)

A Figura 7 ilustra alguns exemplos de funções de ativação:

A Equação 2 representa a função segmentalmente linear, a função degrau expressada pela Equação 3, sigmoide pela Equação 4 e por fim, a tangente hiperbólica expressa pela Equação 5.

$$f(x) = \begin{cases} -1, & x < -1 \\ x, & -1 \leq x \leq 1 \\ 1, & x > 1 \end{cases} \tag{2}$$

$$f(x) = \begin{cases} 0, & x < 0 \\ 1, & x \geq 0 \end{cases} \tag{3}$$

$$f(x) = \frac{1}{1+e^{-x}} \tag{4}$$

$$f(x) = tanh(x) \tag{5}$$

Os neurônios artificiais podem se interconectar para compor estruturas de RNAs, que são elencadas a seguir.

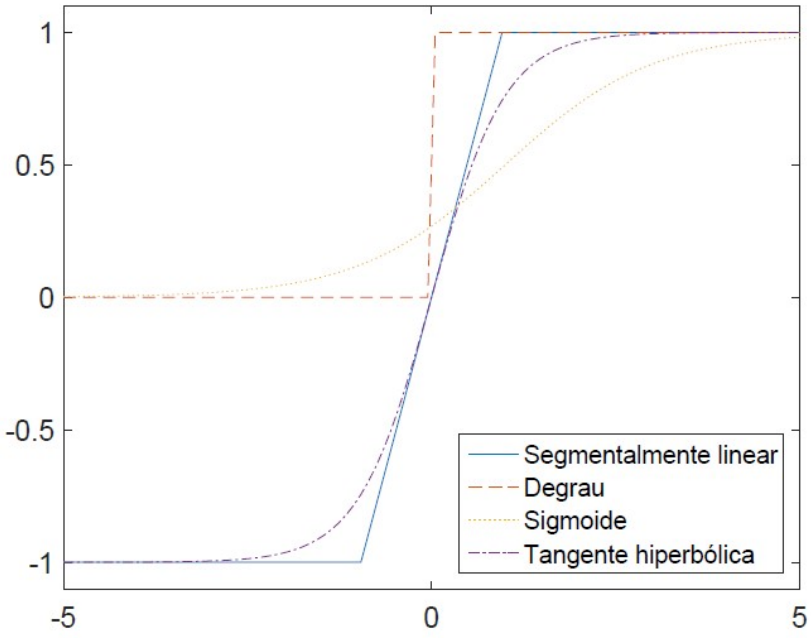

Figura 7 – Exemplo de funções de ativação

Fonte: Adaptado de (HAYKIN, 2011)

3.2.3 Estrutura de Redes Neurais

A maneira pela qual os neurônios de uma rede neural estão estruturados está intimamente ligada com o algoritmo de aprendizagem usado para treinar a rede (HAYKIN, 2011). A estrutura, ou arquitetura de uma RNA, é a forma com a qual os neurônios se conectam uns aos outros em uma rede neural, o que influencia diretamente na sua capacidade de aprendizagem.

Diferentes arquiteturas possuem diversas maneiras para ajustar os pesos e os limiares de seus neurônios. Esse processo de ajuste é conhecido como algoritmo de aprendizagem, sintoniza-se a rede para que as respostas saiam próximas aos valores desejados. Uma rede neural é composta por três camadas:

- Camada de entrada: responsável por receber os dados a serem processados, geralmente normalizados em relação as faixas de variação produzidas pelas funções de ativação;
- Camadas ocultas: chamadas também de intermediárias ou escondidas. São compostas por neurônios que são efetivamente unidades processadoras, que possuem a capacidade de extrair as caracterısticas necessárias para o processo de aprendizagem;
- Camada de saída: camada responsável pela produção e apresentação dos resultados finais da rede, advindos de processos realizados nas camadas anteriores.

Segundo Haykin (2011), em geral é possível distinguir três classes fundamentais de arquiteturas: redes *feedforward* (alimentação a frente) de camada simples, redes *feedforward* de camadas múltiplas e redes recorrentes.

3.2.4 Arquitetura *feedforward* de camada única

Este caso mais simples de rede em camadas consiste em uma camada de entrada (*input layer*) e uma camada de saída (*output layer*), geralmente os neurônios de entrada são lineares, ou seja, eles simplesmente propagam o sinal da camada de entrada para a de saída, propagando o sinal de forma unidirecional. A partir da Figura 8 é possível ver que nas redes pertencentes a essa arquitetura, o número de saídas de rede sempre coincidirá com sua quantidade de neurônios. Essas redes são geralmente empregadas em problemas de classificação de padrões e filtragem linear (SILVA et al., 2016).

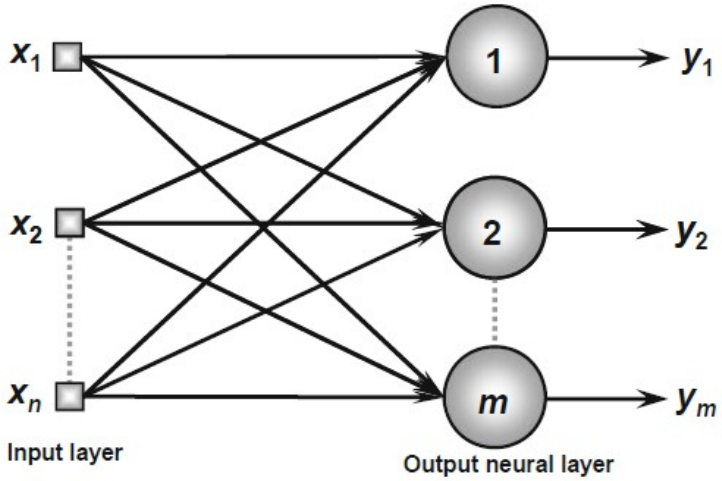

Figura 8 – Exemplo de uma rede de camada única
Fonte: (SILVA et al., 2016)

Como exemplos destas arquiteturas existem Perceptron e Adaline baseadas nas regras de Hebb e Delta, respectivamente (HAYKIN, 2011; SILVA et al., 2016).

3.2.5 Arquitetura *feedforward* de múltiplas camadas

A segunda classe de uma rede neural alimentada adiante (*feedforward*) se distingue pela presença de uma ou mais camadas ocultas ou escondidas (HAYKIN, 2011). Adicionando-se uma ou mais camadas ocultas, aumenta-se o poder computacional de processamento não-linear e armazenagem da rede. O conjunto de saídas dos neurônios de cada camada da rede é utilizada como entrada para a camada seguinte. Em geral o algoritmo de treinamento para este tipo de rede requer a propagação direta *feedforward* do sinal de entrada através da rede, e a retro-propagação (propagação reversa, ou *backpropagation*) do sinal de erro.

Uma estrutura muito utilizada é a Rede Perceptron Multicamada (do Inglês, *Multilayer Perceptron*) (MLP), composta por uma camada de entrada, uma ou mais camadas intermediárias e uma camada de saída. A MLP é totalmente conectada a nível de camadas, o que significa que todos os neurônios de cada camada tem conexão com todos os neurônios das camadas adjacentes. A Figura 9 apresenta uma rede MLP com três neurônios na camada de entrada, quatro neurônios na camada escondida e dois neurônios na camada de saída.

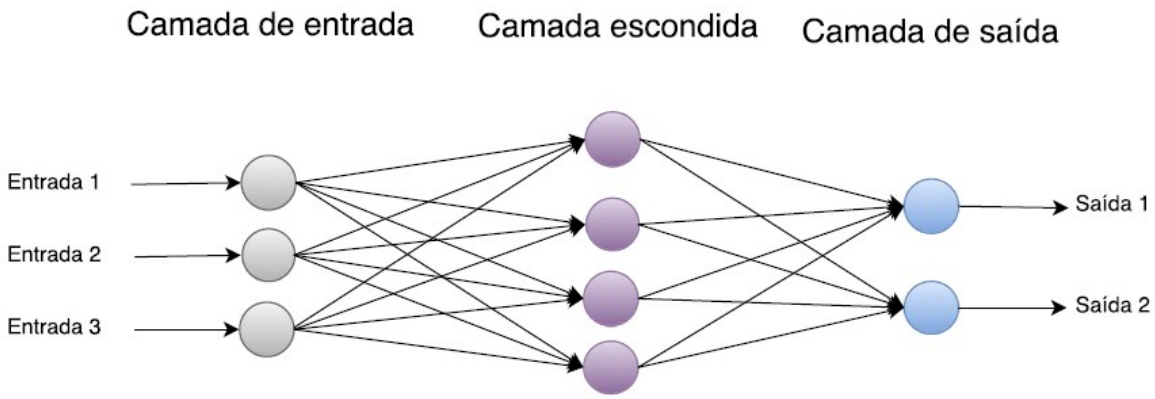

Figura 9 – Estrutura de uma rede MLP 3x4x2
Fonte: (SILVA et al., 2016)

3.2.6 Arquitetura recorrente ou realimentada

O terceiro principal tipo de arquitetura de RNAs são as denominadas de Rede Neural

Recorrente (RNR), pois elas possuem, pelo menos, um laço realimentando a saída de neurônios para outros neurônios da rede. A RNR distingue-se das redes *feedforward* pela existência de pelo menos um laço (*loop*) de recorrência (*feedback*), e podem suportar memória de curto prazo, o que as torna mais interessantes como modelos de cérebro, mas também mais difícil de entender (NORVIG; RUSSELL, 2013).

A Figura 10 ilustra um exemplo de uma rede Perceptron com realimentação, na qual um dos seus sinais de saída é realimentado para a camada do meio. Assim, usando o processo de *feedback*, as redes com essa arquitetura produzem saídas que levam em consideração os valores de saída anteriores.

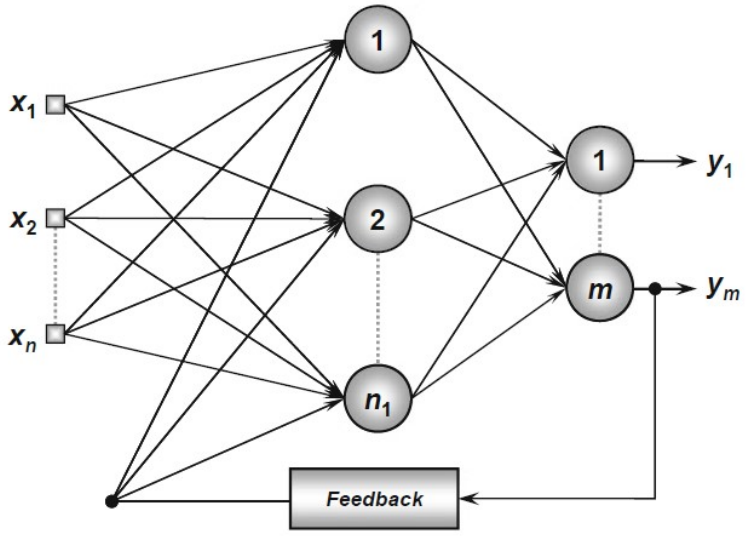

Figura 10 – Exemplo de uma RNR

Fonte: (SILVA et al., 2016)

Segundo Silva et al. (2016), o recurso de *feedback* qualifica essas redes para o processamento dinâmico de informações, o que significa que elas podem ser empregadas em sistemas variantes no tempo, como previsão de séries temporais, identificação e otimização do sistema, controle de processo e assim por diante.

As redes neurais recorrentes são uma maneira muito natural de modelar dados sequenciais, possuem a capacidade de lembrar informações em seu estado oculto por um longo período de tempo, mas é muito difícil treiná-las para usar esse potencial. Atualmente há muito interesse em encontrar formas eficientes de treinar uma RNR.

3.3 PROCESSO DE TREINAMENTO

Uma das características mais relevantes das redes neurais artificiais é sua capacidade de aprender a partir da apresentação de amostras (padrões), que expressa o comportamento do sistema (SILVA et al., 2016). Portanto, depois que ocorre o aprendizado da relação entre entradas e saídas, ela pode generalizar soluções, o que significa que a rede pode produzir uma saída próxima da saída esperada de quaisquer valores de entrada fornecidos.

Segundo Silva et al. (2016), o processo de treinamento de uma rede neural consiste em especificar as etapas ordenadas necessárias para ajustar os pesos e limiares sinápticos de seus neurônios. O conjunto de etapas ordenadas usadas para treinar a rede é chamado de algoritmo de aprendizado. De acordo com Norvig e Russell (2013), as três principais formas de aprendizagem são:

Aprendizagem supervisionada

A estratégia de aprendizagem supervisionada consiste em disponibilizar as saídas desejadas para um dado conjunto de entrada, ou seja, um supervisor monta um conjunto de vetores de exemplos com entradas e saídas desejadas para a RNA, e o objetivo é encontrar e formular uma função que correlacione estes pares entrada-saída.

A cada iteração do algoritmo de treino, a saída obtida pela rede é comparada com a saída desejada, os pesos e limiares sinápticos da rede são continuamente ajustados de forma a minimizar discrepância entre as saídas produzidas com relação às saídas desejadas. A rede é considerada "treinada" quando esta discrepância está dentro de uma faixa de valores aceitável, levando em conta os propósitos de generalizar as soluções (SILVA et al., 2016).

Pode-se ainda classificar a forma de aprendizagem quanto a sua saída. Quando a saída desejada for de um conjunto finito de valores (como ensolarado ou nublado), o problema da aprendizagem será chamado de classificação, quando a saída for um número (como temperatura de amanhã), o problema de aprendizagem é chamado de regressão (NORVIG; RUSSELL, 2013).

Aprendizagem não supervisionada

Diferente do aprendizado supervisionado, a aplicação de um algoritmo baseado em aprendizado não supervisionado não requer nenhum conhecimento dos respectivos resultados desejados. A RNA recebe apenas a entrada, e a partir da redundância existente nos dados reconhece padrões a partir dos quais consegue agrupar/categorizar o conjunto de entrada.

O algoritmo de aprendizado ajusta os pesos e limiares sinápticos da rede para refletir os grupos dentro da própria rede. Alternativamente, o projetista da rede pode especificar

a quantidade máxima desses possíveis grupos, usando seu conhecimento sobre o problema (SILVA et al., 2016).

Aprendizagem por reforço

De acordo com Silva et al. (2016), os métodos baseados no aprendizado por reforço são considerados uma variação das técnicas de aprendizado supervisionado, os algoritmos de aprendizagem ajustam os parâmetros neurais internos baseados em qualquer informação qualitativa ou quantitativa adquirida através da interação com o ambiente que está sendo mapeado, usando esta informação para avaliar o desempenho da aprendizagem.

O processo de aprendizado da rede geralmente é feito por tentativa e erro, pois a única resposta disponível para uma determinada entrada é se ela é satisfatória ou insatisfatória. Se satisfatório, os pesos e limiares sinápticos são gradualmente incrementados para reforçar (recompensar) esta condição comportamental envolvida com o sistema.

3.4 REDES NEURAIS RECORRENTES

As Redes Neurais Recorrentes (RNRs) surgiram como uma das técnicas de aprendizagem profunda mais utilizadas, pois são muito eficientes na predição de dados sequenciais. Segundo Haykin (2011), as redes recorrentes são um poderoso conjunto de algoritmos de redes neurais artificiais especialmente úteis para o processamento de dados sequenciais, como: dados de séries temporais, processamento de linguagem natural, visão computacional, reconhecimento de imagem e fala.

Diferentemente das redes neurais *feed-forward*, redes neurais recorrentes possuem ciclos entre suas unidades. Em outras palavras, unidades podem ter conexões com unidades de camadas anteriores, ou da mesma camada e é visto na Figura 11. Desta forma, a informação não flui em um único sentido, e a saída da rede não depende mais apenas da entrada corrente, mas também das entradas anteriores. O efeito prático disto é a existência de memória de longo prazo na rede.

Atualmente, há muito interesse em encontrar formas eficientes de treinamento para uma RNR devido ao seu potencial em modelar dados sequenciais. Segundo Silva et al. (2016), uma RNR possui as seguintes características: comportamento dinâmico, capacidade de memorizar relacionamentos, possibilidade de armazenar informações e fácil implementação em hardware analógico.

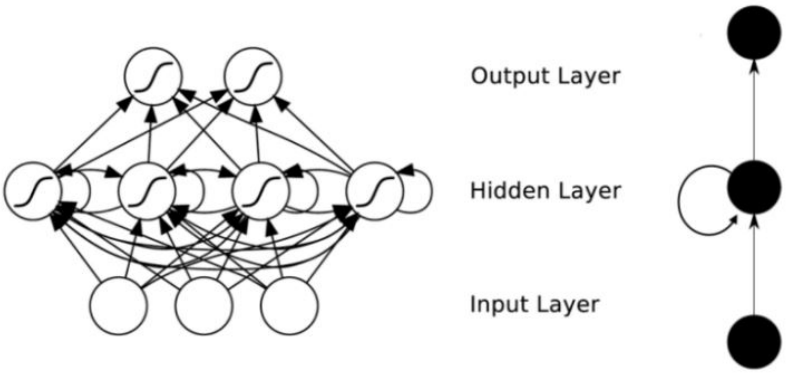

Figura 11 – Exemplo de uma RNR
Fonte: (YIN et al., 2017)

Segundo Olah (2015), todas as redes neurais recorrentes têm a forma de uma cadeia de módulos repetitivos da rede neural. Em RNNs padrão, este módulo de repetição terá uma estrutura muito simples, como uma única camada, que pode ser visto na Figura 12.

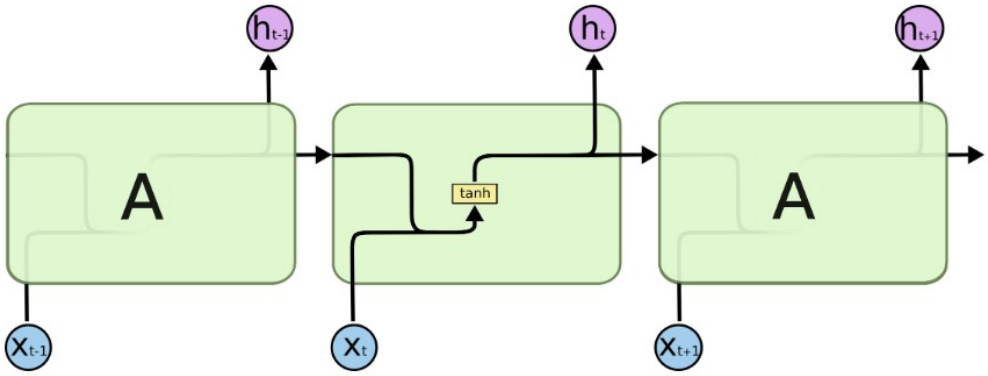

Figura 12 – O módulo de repetição em um RNN padrão contém uma única camada
Fonte: (OLAH, 2015)

.

Ainda que na teoria as RNRs sejam capazes de lidar com dependências de longo termo (longas sequências), Bengio et al. (1994) e Hochreiter (1991) mostram através de experimentos que na prática isso muitas vezes não é possível. A rede recorrente que efetivamente associa memórias e entrada remota no tempo é chamada de Memória de Longo Prazo (LSTM), seu nome relaciona-se com o fato de que são projetadas para evitar o problema de dependência de longo prazo.

3.4.1 *Long short-term memory*

O modelo proposto para classificação do *dataset* é o modelo de *Long short-term memory* (LSTM), proposto por Hochreiter e Schmidhuber (1997), tal modelo foi projetado para evitar o problema da dissipação do gradiente em redes neurais recorrentes. Nos últimos vinte anos, várias modificações de uma célula LSTM original foram propostas. Nesta sessão apresenta-se uma visão geral das estruturas básicas de células LSTM tradicionais.

Os LSTMs também possuem uma estrutura de cadeia, mas o módulo de repetição possui uma estrutura diferente. Em vez de ter uma única camada de rede neural, existem quatro, interagindo de uma maneira muito especial, esquematizados na Figura 13.

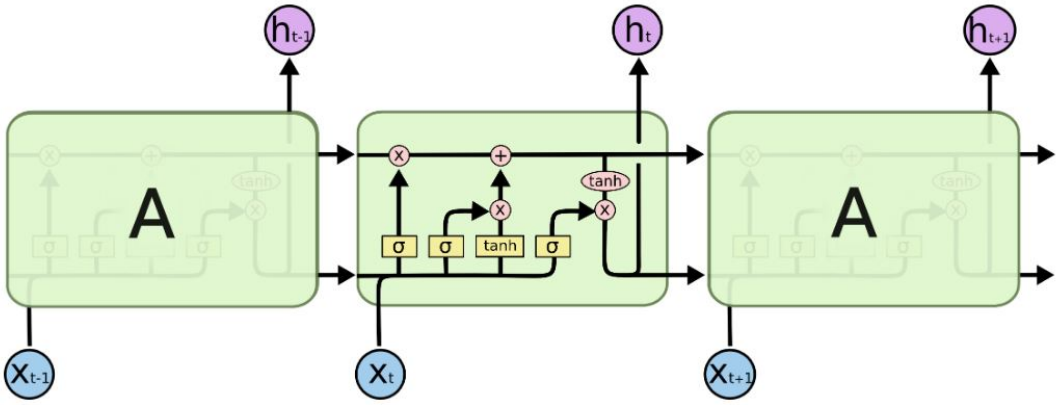

Figura 13 – O módulo de repetição em um LSTM

Fonte: (OLAH, 2015)

.

Cada linha carrega um vetor inteiro, desde a saída de um nó até as entradas de outros. Os círculos cor-de-rosa representam operações pontuais, como a adição de vetores, enquanto as caixas amarelas são camadas de redes neurais aprendidas. As linhas que se fundem denotam a concatenação, enquanto uma linha bifurcada denota que seu conteúdo está sendo copiado e as cópias vão para locais diferentes (OLAH, 2015).

A chave para os LSTMs é o estado da célula, a linha horizontal que passa pela parte superior do diagrama. Seu trabalho é percorrer toda a cadeia, com apenas algumas interações lineares menores. Sua representação pode ser vista na Figura 14.

O LSTM tem a capacidade de remover ou adicionar informações ao estado da célula, e são reguladas por estruturas chamadas de *gates*, ou de portões em português. Os portões são uma forma de, opcionalmente, deixar passar as informações, são compostos de uma camada

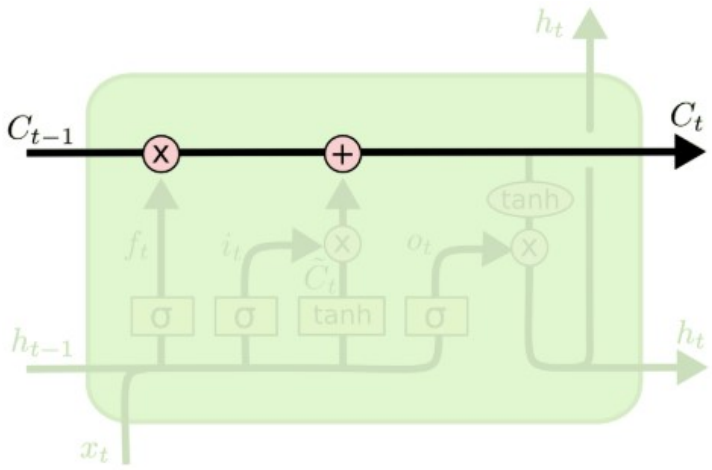

Figura 14 – Estado da célula

Fonte: (OLAH, 2015)

de rede neural sigmoide e uma operação de multiplicação pontual (OLAH, 2015). A camada sigmoide produz números entre zero e um, 1 mantém o componente, e 0 o libera. Um LSTM possui três desses portões, para proteger e controlar o estado da célula, e pode ser visto na Figura 15.

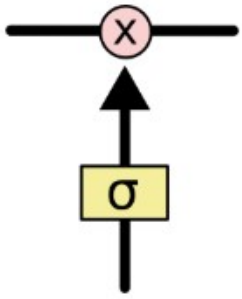

Figura 15 – Exemplo de um *gate*

Fonte: (OLAH, 2015)

O primeiro passo realizado por uma LSTM é decidir quais informações serão descartadas do estado da célula. Esta decisão é tomada por uma camada sigmoide chamada de *forget layer gate*, tal decisão baseada em h_{t-1} e x_t, e gera um número entre 0 e 1 para cada número no estado da célula C_{t-1}, 1 para manter e 0 para descartar. Tal passo é feito pela Equação 6 e na Figura 16 tem-se um exemplo de um *gate*.

$$f_t = \sigma(W_f.[h_{t-1}, x_t] + b_f) \tag{6}$$

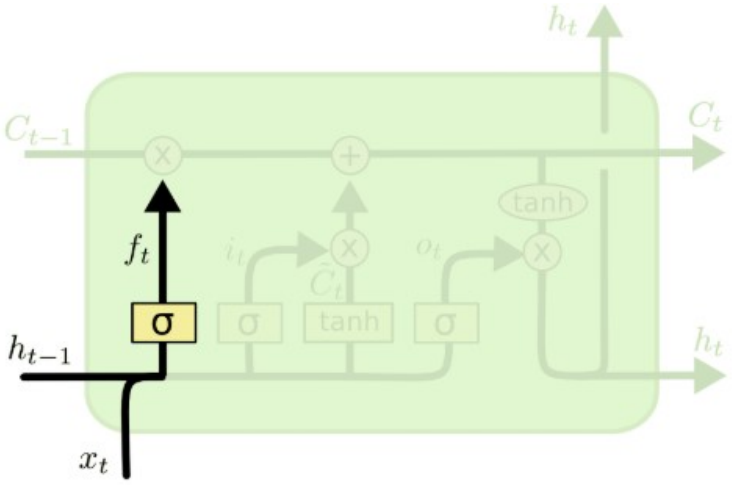

Figura 16 – *Forget layer gate*

Fonte: (OLAH, 2015)

No próximo passo, é decidido quais serão as novas informações a serem armazenadas no estado da célula. Esse processo é dividido em duas etapas, primeiro, uma camada sigmoide chamada de camada de entrada decide quais valores são atualizados, de acordo com a Equação 7. Em seguida, uma camada de *tanh* cria um vetor de novos valores candidatos, $\tilde{C}_t$, isso poderia ser adicionado ao estado pela Equação 8. A combinação dos dois, cria uma atualização para o estado.

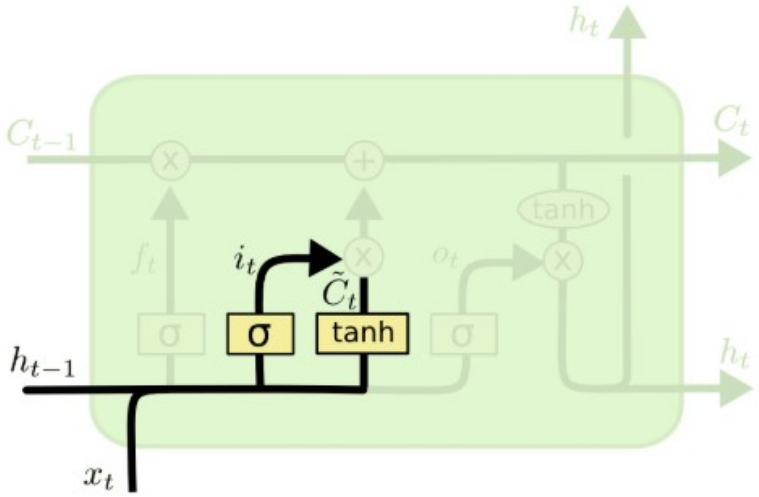

Figura 17 – Entrada de novas informações

Fonte: (OLAH, 2015)

$$i_t = \sigma(W_i.[h_{t-1}, x_t] + b_i) \tag{7}$$

$$\tilde{C}_t = tanh(W_c.[h_{t-1}, x_t] + b_c) \tag{8}$$

De forma a atualizar o estado da célula antiga, C_{t-1}, na nova célula C_t utiliza-se a Equação 9 para tal. A etapa anterior pode ser vista na Figura 18. De forma a multiplicar o antigo estado por f_t, esquecendo as informações anteriores, é adicionado $f_t * \tilde{C}_t$ na equação. Esses são os novos valores candidatos, dimensionados por um valor arbitrário de forma a atualizar cada valor de estado.

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{9}$$

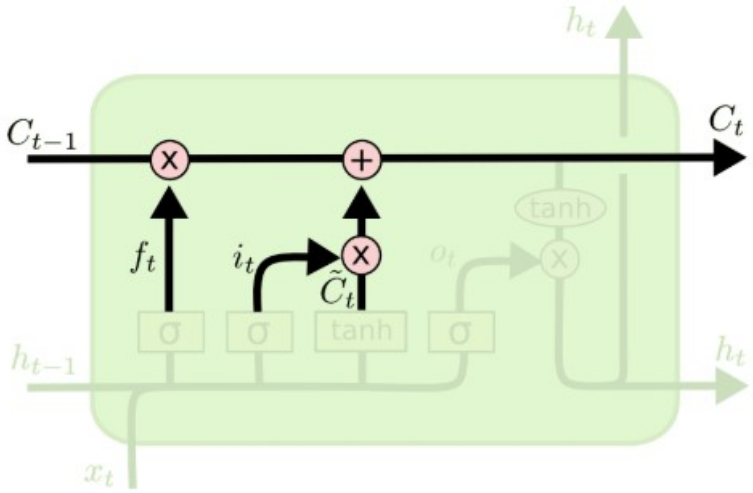

Figura 18 – Atualizar Informações
Fonte: (OLAH, 2015)

A Equação 10 representa o cálculo das informações a serem atualizadas e a saída é baseada no estado de célula, sendo uma versão filtrada da mesma. A camada sigmoide é a responsável por quais partes do estado da célula será produzido. Após isso, é multiplicado o estado da célula *tanh* pela saída da função sigmoide e que pode ser vista na Equação 11. A Figura 19 mostra a última etapa de uma célula LSTM.

$$o_t = \sigma(W_o.[h_{t-1}, x_t] + b_o) \tag{10}$$

$$h_t = o_t * tanh(C_t) \tag{11}$$

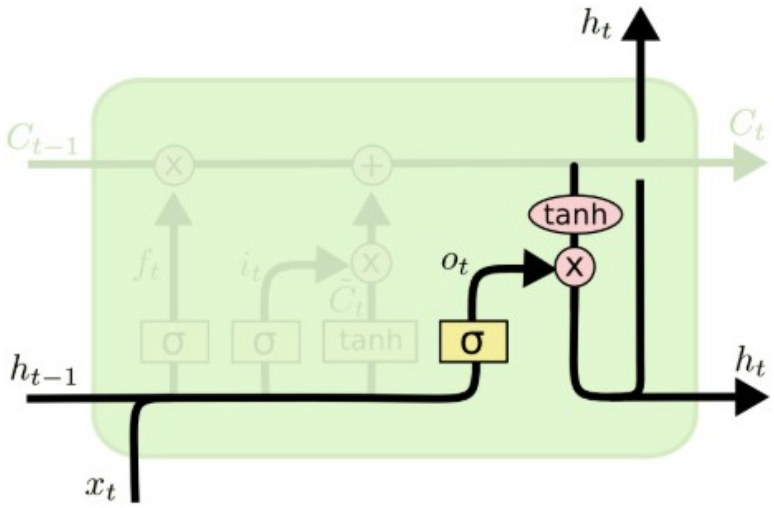

Figura 19 – Atualizar Informações

Fonte: (OLAH, 2015)

3.4.2 Gated GRU

Este novo modelo foi introduzido por Cho et al. (2014) em 2014, o GRU (*Gated Recurrent Unit*) também pode ser considerado uma variação do LSTM, pois ambos são projetados de maneira semelhante e, em alguns casos, produzem resultados igualmente excelentes. A principal diferença com o modelo LSTM é que um único portão passa a controlar simultaneamente a decisão de esquecer e atualizar o estado da célula (GOODFELLOW et al., 2016), assim, a célula passa a ter dois portões, o *reset gate* e o *update gate*. O fluxo de informação dentro de uma célula do tipo GRU é esquematizado na Figura 20.

O *update gate* é responsável por decidir qual parte da saída anterior vai persistir e qual vai ser atualizada. O *update gate* z_t é dado por:

$$z_t = \sigma(W_z.[h_{t-1}, x_t]) \tag{12}$$

Dentro de uma célula do tipo GRU, o *reset gate* é responsável por decidir o quanto da informação do instante anterior deve persistir para a geração dos próximos valores r_t, o *reset gate* é dado pela Equação 13.

$$r_t = \sigma(W_r.[h_{t-1}, x_t]) \tag{13}$$

O conteúdo atual da memória $\tilde{h_t}$ é dado por:

$$\tilde{h}_t = tanh(W.[r_t * h_{t-1}, x_t]) \tag{14}$$

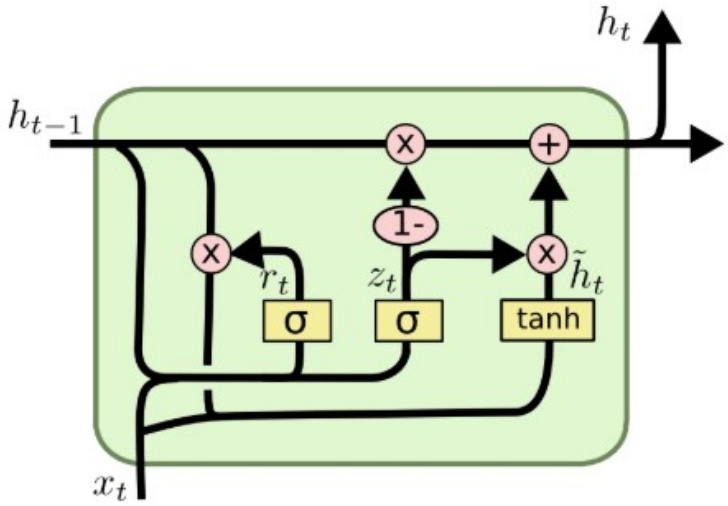

Figura 20 – Fluxo de informações GRU
Fonte: (OLAH, 2015)

E a memória final no momento atual h_t é dada por:

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t \tag{15}$$

As GRUs que usam o recurso de memória interna são valiosas para armazenar e filtrar informações usando suas portas de atualização e redefinição. Comparado ao LSTM, o GRU tem menos portões, isso ocorre porque o GRU não possui um estado de célula e combina as portas de entrada e de esquecimento em um único *gate*, o *gate* de atualização z_t. Portanto, o GRU é muito mais simples que o LSTM em sua estrutura e possui menos parâmetros, o que lhe dá uma grande vantagem em termos de desempenho e convergência.

3.5 TRABALHOS CORRELATOS

Staudemeyer (2015) afirmou que, pela primeira vez, uma rede neural recorrente, em específico o LSTM foi aplicada à detecção de intrusão e confirmou sua eficácia. O LSTM pode aprender a olhar para trás no tempo e descobrir associações a partir de uma perspectiva de tempo. O classificador proposto foi eficaz na detecção de ataques de negação de serviço (DoS) e sondas de rede, ambos com séries temporais distintas de eventos. As experiências mostraram que o LSTM superou as entradas vencedoras do desafio KDD Cup 99, com 93,82% de precisão. Kim et al. (2016) também usaram a estrutura LSTM para construir um modelo de

IDS treinado com o conjunto de dados do KDD 99. Resultados experimentais mostraram que a estrutura LSTM é eficaz para a detecção de intrusão. Em Choudhury e Bhowal (2015), foram testadas diversas técnicas de classificação, como: BayesNet, IBk, JRip, J48, Random Forest dentre outros. Com acurácia máxima de 91,5%.

4 MATERIAIS E MÉTODOS

Para detectar atividades intrusivas na rede, precisa-se da existência de base de dados com padrões normais e padrões de ataques identificados. Com o emprego de técnicas de inteligência computacional baseada em redes neurais artificiais ao problema de detectar atividades intrusivas em redes é o objeto deste trabalho, é utilizado uma base de dados de treinamento e testes da NSL-KDD, um *dataset* que foi aprimorado pelo Instituto Canadense de Cibersegurança[1], sugerido para resolver alguns dos problemas inerentes ao conjunto de dados do KDD Cup'99, mencionados em (TAVALLAEE et al., 2009). Portanto, nesta seção é descrito detalhadamente as técnicas à serem utilizadas neste trabalho.

4.1 CONJUNTO DE DADOS

Para a validação e experimentação de métodos de detecção de intrusão, os pesquisadores geralmente utilizam bases de dados públicas, permitindo mensurar as taxas de detecção de intrusão e acurácia (ASHFAQ et al., 2017). A precisão da detecção de intrusão e o tempo de processamento são fortemente influenciados quando o conjunto de dados não é ajustado para o domínio do problema, ou seja, o conjunto de dados escolhido para os experimentos de detecção de intrusão é tão importante quanto a escolha dos algoritmos.

Para efetuar os teste, optou-se por uma base de dados (*dataset*) que fosse capaz de simular vários tipos de ataques, bem como o tráfego normal de uma rede para que a rede neural possa classificá-los, e não apenas identificar possíveis ataques. O *dataset* escolhido para tal foi o NSL-KDD[2], que é amplamente utilizado pela comunidade de pesquisa de detecção de intrusão para a elaboração de testes consistentes e comparáveis. Nesta seção descreve-se as características do *dataset*, são também discutidos assuntos relacionados à evolução histórica,

[1] https://www.unb.ca/cic/

[2] https://www.unb.ca/cic/datasets/nsl.html

configurações dos subconjuntos de dados, arquivos e formatos.

4.1.1 Evolução dos *datasets* de detecção de intrusão

Para discutir o conjunto de dados NSL-KDD é importante apresentar o seu histórico. Em 1998 o DARPA (*Defense Advanced Research Projects Agency*)[3] em conjunto com o laboratório Lincoln do MIT (*Massachusetts Institute of Technology*)[4], realizaram a captura do tráfego de rede em um período de sete semanas para dados de treinamento e duas semanas de dados de teste, com 38 tipos de ataques e cerca de 1.419.663 conexões de redes.

O conjunto de dados KDD Cup99 foi extraído do DARPA e foi utilizado na 3ª Competição Internacional de Descoberta do Conhecimento e Mineração de Dados. O objetivo dessa competição foi o de construir um modelo capaz de detectar conexões normais ou anômalas em uma rede. O conjunto de dados KDD Cup99 contabiliza cerca de 4.900.000 instâncias de conexões individuais, classificados em conexões normais e anomalias.

Apesar de muito completo, o KDD Cup99 apresenta diversas deficiências que dificultam a sua utilização em IDSs e outras aplicações práticas, pois possui muitas informações redundantes ou mesmo irrelevantes, que geram efeitos indesejáveis no classificador (MCHUGH, 2000). Ao alimentar a rede neural com parâmetros que não agregam informação, a fase de treinamento é prejudicada, diminuindo a capacidade da rede neural de aprender.

Com o intuito de resolver tais problemas, os pesquisadores Tavallaee et al. (2009) propuseram uma nova base denominada NSL-KDD[5], construída com registros selecionados do conjunto completo de dados da KDD Cup99, mantendo os mesmos atributos, e otimizando a questão de redundância de amostras. A Tabela 2 mostra as estatísticas de redundância de instâncias no conjunto de treinamento KDD Cup99, e a Tabela 3 mostra as estatísticas do

Tabela 2 – Estatísticas de redundância de instâncias no conjunto KDD Cup99 de Treinamento

	Instâncias originais	Instâncias distintas	Taxa de Redução
Ataques	3.925.650	262.178	93,32%
Normais	972.781	812.814	16,44%
Total	4.898.431	1.074.992	78,05%

Fonte: (TAVALLAEE et al., 2009)

Tabela 3 – Estatísticas de redundância de registros no conjunto KDDCup99 de Teste

	Instâncias originais	Instâncias distintas	Taxa de Redução
Ataques	250.436	29.378	88,26%
Normais	60.591	47.911	20.92%
Total	311.027	77.289	75.15%

Fonte: (TAVALLAEE et al., 2009)

conjunto de teste.

4.2 CONJUNTO DE DADOS NSL-KDD

Cada exemplo da base NSL-KDD possui 42 atributos, dos quais 41 são características comportamentais da rede e são extraídos dos pacotes capturados durante a atividade da mesma. O último atributo é o que rotula a instância como algum tipo de ataque ou comportamento normal, os atributos presentes na base NSL-KDD são divididos em 4 categorias: atributos básicos, atributos de conteúdo, atributos de tráfego baseados em tempo e atributos de tráfego baseados em conexão. Em um IDS, os atributos com as características básicas das conexões TCP individuais são essenciais na detecção de intrusão e são apresentados na Tabela 4.

Na Tabela 5 são apresentados os atributos que contém informações capturadas dos pacotes de rede, utilizadas para chegar a conclusões de quais padrões associados às conexões podem representar um determinado tipo de ataque.

Os demais atributos correspondem a informações estatísticas relacionadas ao tráfego anterior e atual, e por essa razão são chamados de atributos de tráfego. Tais dados são divididos em dois grupos: atributos de tráfego baseados em tempo, que levam em consideração as últimas conexões em um intervalo de tempo e pode ser visto na Tabela 6.

[3]https://www.darpa.mil/
[4]http://web.mit.edu/
[5]https://www.unb.ca/cic/datasets/nsl.html

Tabela 4 – Atributos intrínsecos da conexão TCP

ID	Nome	Descrição	Tipo
1	duration	Tempo em segundos de conexão	Contínuo
2	protocol_type	Tipo de conexão (TCP, UDP)	Discreto
3	service	Tipo de serviço no destino (HTTP, Telnet)	Discreto
4	src_byte	Número de bytes da origem ao destino	Contínuo
5	dst_byte	Número de bytes do destino à origem	Contínuo
6	flag	Estado da conexão (normal ou erro)	Discreto
7	land	Portas de origem e destino iguais	Discreto
8	wrong_fragment	Número de fragmentos corrompidos	Contínuo
9	urgent	Número de pacotes urgentes	Contínuo

Fonte: (HETTICH; BAY, 1999)

Tabela 5 – Atributos de conteúdo capturados de pacotes de rede

ID	Atributo	Descrição	Tipo
10	hot	Número de indicadores “importantes”	Contínuo
11	num_failed_logins	Tentativas inválidas de login	Contínuo
12	logged_in	1 se login obteve sucesso e 0 caso contrário	Discreto
13	num_comprissed	Número de condições comprometedoras	Contínuo
14	root_shell	1 se o Shell root é obtido e 0 caso contrário	Discreto
15	su_attempted	Tentativa de login su_root	Discreto
16	num_root	Número de acessos como root	Contínuo
17	num_file_creations	Número de operações de criação de arquivos	Contínuo
18	num_shells	Número de shell prompts abertos	Contínuo
19	num_access_files	Operações em arquivos de controle de acesso	Contínuo
20	num_outpund_cmds	Número de comandos externos (sessão FTP)	Contínuo
21	is_hot_login	Se o login pertence a lista “hot”	Discreto
22	is_guest_login	Se o login é do tipo “guest”	Discreto

Fonte: (HETTICH; BAY, 1999)

Os atributos de tráfego baseados em conexões, que consideram um número específico de últimas conexões pode ser visto na Tabela 7.

Os recursos “ same host ” examinam apenas as conexões nos últimos dois segundos que possuem o mesmo host de destino que a conexão atual e calculam as estatísticas relacionadas ao comportamento do protocolo, serviço, etc.

Os recursos semelhantes do ”same service”examinam apenas as conexões nos últimos dois segundos que possuem o mesmo serviço que a conexão atual.

Os recursos *same host* e *same service* são chamados de recursos de tráfego baseados em tempo dos registros de conexão.

Alguns ataques de rastreamento examinam os *hosts* (ou portas) usando um intervalo de tempo muito maior que dois segundos, por exemplo, uma vez por minuto. Portanto, os registros

Tabela 6 – Características temporais: janela de 2 segundos

ID	Nome	Descrição	Tipo
23	count	Conexões com o mesmo host que a atual	Contínuo
24	srv_count	Conexões com o mesmo serviço que a atual	Contínuo
25	serror_rate	Conexões que possuem erros “SYN”na ID 23	Contínuo
26	srv_serror_rate	Conexões que possuem erros “SYN”na ID 24	Contínuo
27	rerror_rate	Conexões que possuem erros “REJ”na ID 23	Contínuo
28	srv_rerror_rate	Conexões que possuem erros “REJ”na ID 24	Contínuo
29	same_srv_rate	Conexões com o mesmo serviço na ID 23	Contínuo
30	diff_srv_rate	Conexões em serviços diferentes na ID 23	Contínuo
31	srv_diff_host_rate	Conexões com hosts diferentes na ID 24	Contínuo

Fonte: (HETTICH; BAY, 1999)

Tabela 7 – Atributos de tráfego baseados em conexão do *host*

ID	Nome	Descrição	Tipo
32	count	Número de conexões para o mesmo host da conexão atual dentre as últimas 100 conexões	Contínuo
33	srv_count	Número de conexões ao mesmo serviço da conexão atual dentre as últimas 100 conexões	Contínuo
34	same_srv_rate	% de conexões ao mesmo serviço, dentre as conexões consideradas na ID 32	Contínuo
35	diff_srv_rate	% de conexões a diferentes serviços, dentre as conexões consideradas na ID 32	Contínuo
36	same_src_port_rate	% de conexões a mesma porta de origem, dentre as conexões consideradas na ID 33	Contínuo
37	srv_diff_host_rate	% de conexões a diferentes hosts, dentre as conexões consideradas na ID 33	Contínuo
38	serror_rate	% de conexões que tiveram erros do tipo “SYN”, dentre as conexões na ID 32	Contínuo
39	srv_serror_rate	% de conexões que tiveram erros “SYN”, dentre as conexões consideradas na ID 33	Contínuo
40	rerror_rate	% de conexões que tiveram erros do tipo “REJ”, dentre as conexões consideradas na ID 32	Contínuo
41	srv_rerror_rate	% de conexões que tiveram erros do tipo “REJ”, dentre as conexões consideradas na ID 33	Contínuo

Fonte: (HETTICH; BAY, 1999)

de conexão também foram classificados pelo *host* de destino e os recursos foram construídos usando uma janela de 100 conexões para o *samehost* em vez de uma janela de tempo. Isso produz um conjunto de recursos de tráfego baseados em *host*.

Ao contrário da maioria dos ataques de negação de serviços e sondagem, parece não haver padrões sequenciais que sejam frequentes nos registros de ataques de R2L e U2R. Isso

ocorre porque os ataques de negação de serviços e sondagem envolvem muitas conexões com alguns *hosts* em um período de tempo muito curto, mas os ataques R2L e U2R são incorporados nas partes de dados dos pacotes e normalmente envolvem apenas uma única conexão.

4.2.1 Análise dos arquivos do conjunto NSL-KDD

Os arquivos em formato TXT foram criados para serem manipulados por ferramentas como o Microsoft Excel e os arquivos em formato ARFF (*Attribute-Relation File Format*) para serem manipulados no ambiente WEKA.

Tabela 8 – Arquivos do conjunto de dados NSL-KDD

Nome dos arquivos	Descrição
KDDTrain+.ARFF	Arquivo NSL-KDD de treinamento completo com atributos binários em formato ARFF.
KDDTrain+.TXT	Arquivo de treinamento NSL-KDD completo que inclui classe do tipo de ataque e nível de dificuldade 8 em formato CSV (*Comma Separated Values*).
KDDTrain+_20Percent.ARFF	Arquivo contendo 20% do arquivo KDDTrain+.arff.
KDDTrain+_20Percent.TXT	Arquivo com 20% do arquivo KDDTrain+.txt.
KDDTest+.ARFF	Arquivo completo NSL-KDD de teste com atributos binários em formato ARFF.
KDDTest+.TXT	Arquivo completo NSL-KDD que inclui a classe do tipo de ataque e nível de dificuldade, em formato CSV.
KDDTest-21.ARFF	Subconjunto do arquivo KDDTest+.ARFF que não inclui registros com nível de dificuldade 21.
KDDTest-21.TXT	Subconjunto do arquivo KDDTest+.TXT que não inclui registros com nível de dificuldade 21.

Fonte: (TAVALLAEE et al., 2009)

O conjunto de dados NSL-KDD gerado em 2009 é amplamente utilizado em experimentos de detecção de intrusão. Na literatura mais recente visto em Lin et al. (2018), Shah e Issac (2018), Yin et al. (2017), a maioria dos pesquisadores usam o NSL-KDD como o conjunto de dados de referência, que não apenas soluciona efetivamente os problemas inerentes aos registros redundantes do conjunto de dados KDD Cup99, mas também torna razoável o número de registros no treinamento, de tal forma que o classificador não favoreça registros mais frequentes.

4.3 TAXONOMIA DOS ATAQUES UTILIZADOS NO DATASET

Para facilitar a classificação e a conceituação, no presente estudo será utilizada a taxonomia que propõe a classificação das técnicas de ataques e intrusões em quatro categorias distintas, tal classificação utilizada pelo *dataset* NSL-KDD:

- *Denial of Service* (DoS): segundo Basta et al. (2014), um ataque de negação de serviço (DoS) é o tipo de ataque em que o invasor faz com que os recursos computacionais ou de memória dos equipamentos fiquem sobrecarregados de tal forma que as requisições normais oriundas da rede não possam ser atendidas e assim, o equipamento fique indisponível e não possa ser acessado;
- *Remote to Local Attacks* (R2L): um ataque remoto ao usuário é uma classe de ataques em que um invasor envia pacotes para um máquina através de uma rede, em seguida, explora a vulnerabilidade da máquina para obter ilegalmente acesso local do utilizador, acesso não autorizado a partir de uma máquina remota. Os ataques de R2L são um dos mais difíceis de detectar, como eles envolvem a camada de Rede e os recursos à nível do usuário, destaca-se campos importantes como: duração da conexão, serviço solicitado, número de tentativas de login, entre outros;
- *User to Root attacks* (U2R): este tipo de ataque ocorre quando o invasor tenta explorar as vulnerabilidades do sistema a fim de obter privilégios adicionais de administrador/*root*. Os ataques U2R envolvem os detalhes semânticos que são muito difíceis de capturar em um estágio inicial. Tais ataques têm como alvo uma aplicação vulnerável.
- *probing*: são técnicas de reconhecimento, o invasor pode descobrir, por exemplo, quais serviços estão ativos, o fabricante e versão do software que fornece cada um destes serviços dentre outras informações. Do ponto de vista dos IDSs, detectar atividade relacionada a esta categoria é importante, uma vez que permite uma preparação e resposta adequada a próxima etapa que provavelmente será iniciada pelo intruso.

O conjunto de testes possui 24 tipos de ataques específicos no conjunto de treino e 13 tipos no conjunto de treinamento, o que permite fornecer uma base teórica mais realista para a detecção de invasões. O conjunto de dados pode ainda ser mapeado de acordo com o tipo de ataque por classe e é mostrado na Tabela 9.

A Tabela 10 detalha as quantidades de instâncias por tipos de ataque.

Tabela 9 – Mapeamento dos tipos de ataque por classe

Classe	Tipo de Ataque	Total
DoS	Back, Land, Neptune, Pod, Smurf, Teardrop, Apache2, Udpstorm, Processtable, Worm	10
Probe	Satan, Ipsweep, Nmap, Portsweep, Mscan, Saint,	11
R2L	Guess_Password, Ftp_write, Imap, Phf, Multihop, Warezmaster, Warezclient, Spy, Xlock, Xsnoop, Snmpguess, Snmpgetattack, Httptunnel, Sendmail, Named	10
U2R	Buffer_overflow, Loadmodule, Rootkit, Perl, Sqlattack, Xterm, Ps	7

Fonte: Autoria Própria

Tabela 10 – Detalhamento das quantidades de instâncias por tipos de ataques dos subconjuntos de dados que compõem o NSL-KDD

Nome dos arquivos do subconjunto de dados	Total de Instâncias	Classe Normal	Classe DoS	Classe Probing	Classe U2R	Classe R2L
KDDTrain+.ARFF	125.973	67.343	45.927	11.656	52	995
KDDTrain+_20Percent.ARFF	25.192	13.449	9.234	2.289	11	209
KDDTest+.ARFF	22.544	9.711	7.458	2.421	200	2.754
KDDTest-21.ARFF	11.850	2.152	4.342	2.402	200	2.754

Fonte: Autoria Própria

4.4 PRÉ-PROCESSAMENTO DE DADOS

Existem 38 características numéricas e 3 características não numéricas no conjunto de dados NSD-KDD. Como o valor de entrada para os testes deve ser uma matriz numérica, para isso, foi utilizado a função LabelEncoder do sklearn.preprocessing para transformar dados como “protocol_type”, “service”e “flag”, em formato numérico.

A base NSL-KDD é rotulada em 23 classes, sendo que 22 concernem a tipos de ataque e 1 a comportamento normal. O foco do presente trabalho é detectar comportamentos intrusivos e não intrusivos, não havendo uma preocupação em determinar a classe ou o ataque específico. Deste modo, a base será modificada, de modo a rotular os exemplos apenas como intrusivos ou não intrusivos, na qual as 22 classes de ataques passarão a compor uma única classe de amostras intrusivas.

4.4.1 Transformação de Dados

Após os dados estarem todos em forma numérica, foi utilizado a função Normalizer do sklearn.preprocessing para normalizar os dados no intervalo entre 0 e 1, tal intervalo é utilizado pelas RNR para treinamento e teste. Por último, para alimentar a rede com os dados, foi necessário mudar os formatos dos numpy arrays de 2D para 3D, pois é uma exigência da camada de entrada da LSTM e GRU.

4.5 ESPECIFICAÇÕES DO AMBIENTE DE TREINAMENTO

No desenvolvimento de métodos para treinar algoritmos de RNR, a comunidade da área de aprendizado de máquina tem adotado sistemas baseados em GPU, pois, de acordo com o modelo em foco, é capaz de realizar em horas treinamentos que CPU realizam em dias. De tal forma, foi escolhido o Google Colab, que é um serviço gratuito fornecido pela Google executado integralmente em nuvem, tem a capacidade de processar informações em CPU, GPU e TPU e conta com as seguintes especificações:

- Suporte para Python 2.7 e Python 3.6[6];
- Placa gráfica Tesla 12GB;
- Memória RAM CPU de 13.3 GB.

4.5.1 Bibliotecas e Linguagem de Programação

As bibliotecas utilizadas para o treinamento das RNRs contemplam em sua estrutura todas a funcionalidades necessárias para criação, treinamento e utilização das RNRs. Além disso, as bibliotecas empregadas na implementação do presente trabalho possuem vasta documentação e são todas ferramentas de *software* livre:

Python: é uma linguagem de programação de alto nível, que possui suporte a diversos

[6]https://www.python.org/

paradigmas como funcional, imperativo, orientado a objetos e oferece bibliotecas prontas para as necessidades de um projeto de inteligência artificial. É desenvolvida pela comunidade e mantida pela *Python Software Foundation*;

Tensorflow: é uma biblioteca de código aberto mantida e desenvolvida por pesquisadores da Google. Tem como seu objetivo auxiliar na implementação de modelos matemáticos voltados ao aprendizado de máquina, com foco principal na aprendizagem profunda (*Deep Learning*)[7];

NumPy: é outra biblioteca de código aberto. Adiciona o suporte a tensores multidimensionais ao Python. Possui uma grande coleção de funções matemáticas de alto nível para manipular tensores. O NumPy[8] é mantido e desenvolvido pela comunidade;

Keras: é uma biblioteca de rede neural que trabalha em conjunto com o TensorFlow. Projetada para permitir experimentações rápida com redes neurais profundas, é fácil de usar, modular e extensível[9];

Pybrain: a biblioteca oferece alguns algoritmos de treinamento fáceis de usar para redes, conjuntos de dados, treinadores para treinar e testar a rede [10].

4.6 ABORDAGEM EXPERIMENTAL

Em relação a todo processo de treinamento dos testes propostos nas duas RNR, tem-se o uso de parâmetros, funções de perda e otimizadores que auxiliam a rede neural na obtenção da melhor performance de classificação e segmentação.

A melhor acurácia do modelo foi encontrada utilizando 100 épocas de treinamento para as duas redes, a quantidade de amostras que alimentam a rede em cada iteração, conhecido como *batchsize* foi de 32, e foram utilizadas duas abordagens para trabalhar com os *dataset*:

Validação Cruzada (k-fold): é uma técnica para avaliar a capacidade de generalização de um modelo, amplamente utilizada em problemas na qual o objetivo da modelagem é a predição. Essa técnica realiza o particionamento do *dataset* em subconjuntos mutuamente exclusivos, e separa esses subconjuntos em parâmetros de treinamento e de validação e ao final, calcula-se a acurácia sobre os erros encontrados. Percebe-se que apesar de alguns *fold* apresentarem resultados suavemente melhores, é constatado que a rede se comportou bem por

[7] https://www.tensorflow.org/
[8] http://www.numpy.org/
[9] http://www.keras.io/
[10] http://www.pybrain.org/

os resultados não terem alta discrepância.

Holdout: Este método divide o *dataset* em dois subconjuntos mutuamente exclusivos, um para o treinamento e outro para teste. Foi utilizado 60 porcento dos dados para treinamento e 40 porcento para testes.

4.6.1 Função de Ativação

Para modelar também as relações não-lineares da rede, precisa-se utilizar uma função de ativação, para este trabalho foi escolhido a função Sigmoid, é a função utilizada nos 3 portões da LSTM e nos 2 Portões da GRU, ela aproxima os resultados em 0 e 1, facilitando o processo de classificação. Para a função de ativação utilizada dentro das camadas da rede, foi utilizado a Tanh, pois é a configuração padrão da rede.

4.6.2 Função de Custo

Como é uma classificação binária, a função de custo escolhida foi a entropia cruzada (*binary crossentropy*), de forma a calcular o erro das previsões da rede.

4.6.3 Otimização e Performance da Rede

Por fim, os modelos neurais utilizados para realização dos testes de detecção de intrusões passaram por um processo de ajuste fino (*fine tuning*), método que tem como objetivo aumentar a performance de generalização da rede durante a fase de treinamento. Como técnica desse ajuste, foi implementada uma camada de *Dropout*, proposta por Srivastava et al. (2014), antes da camada de saída completamente conectada.

Em todos os testes foi determinado o Adam como otimizador, proposto por Kingma e Ba (2014). Considerado como uma versão melhorada do gradiente descendente, ele incorpora momento e taxa de aprendizado adaptativa, mitigando a rápida queda da taxa de aprendizagem.

4.7 ANÁLISE ESTATÍSTICA

Nesta etapa os resultados dos experimentos foram avaliados, a fim de comparar o método LSTM com o GRU, propostos neste trabalho. Para problemas de classificação, o resultado de uma classificação pode estar correto ou incorreto e todos os resultados possíveis podem ser divididos nas quatro condições a seguir de uma matriz de confusão:

- Verdadeiro Positivo (VP): ataques reais que são corretamente classificados;
- Verdadeiro Negativo (VN): essa categoria abrange instâncias não intrusivas, e corretamente classificados pelo método de detecção;
- Falso Positivo (FP): registros normais que são classificados como ataques. Essa condição também é conhecida como falsos alarmes;
- Falso Negativo (FN): instâncias classificadas como normais pelo método de detecção, e que são intrusões.

A partir desta definição e dos resultados dos experimentos, serão calculadas alguns indicadores, a taxa de acurácia na qual é mostrado a porcentagem de instâncias classificadas corretamente e é dada pela Equação 16.

$$Acurácia = \frac{VP+VN}{VP+VN+FP+FN} \tag{16}$$

A taxa de erro na qual é mostrado a porcentagem de instâncias que foram classificadas incorretamente é dada pela Equação 17.

$$Erro = \frac{FP+FN}{VP+VN+FP+FN} \tag{17}$$

Precisão: dentre todas as classificações de classe Positivo que o modelo fez, quantas estão corretas, é dado pela Equação 18.

$$Precisão = \frac{VP}{VP+FP} \tag{18}$$

Sensibilidade (*recall*): dentre todas as situações de classe Positivo como valor esperado, quantas estão corretas, é dada pela Equação 19.

$$Sensibilidade = \frac{VP}{VP+FN} \tag{19}$$

F1-Score: média harmônica entre precisão e recall, é calculada pela Equação 20.

$$F1 - Score = \frac{2 \cdot precisão \cdot recall}{precisão + recall} \tag{20}$$

5 RESULTADOS E DISCUSSÃO

Neste capitulo é discutido os resultados obtidos no treinamento das redes e ao final, todos os resultados estão resumidos em uma tabela comparativa entre o LSTM e o GRU, e uma tabela com o comparativo com outros trabalhos.

5.1 EXPERIMENTO COM A LSTM

Todo o treinamento da rede LSTM seguiu a metodologia descrita no capítulo 4, o resultado máximo foi alcançado com 100 épocas de treinamento, depois disso a rede não melhorava mais, podendo entrar em *overfitting*. Percebe-se por meio da tabela 11 que a rede é bem generalizada, independe do *fold* analisado, vemos também que atingiu altas taxas de acurácia na Detecção de Intrusão em Redes. Pela abordagem de regularização *Holdout*, os resultados foram sutilmente melhores do que o *k-fold*, e pode ser visto na Tabela 12 com uma acurácia de 98,63% na classificação.

Tabela 11 – Resultado Rede LSTM K-Fold(5)

K-Fold	Acurácia(%)	Erro
Fold 1	98.4177	0.0395
Fold 2	98.7712	0.364
Fold 3	98.3201	0.0364
Fold 4	98.5153	0.0422
Fold 5	98.7173	0.0373
Média	98.5483	0.0394

Fonte: Autoria Própria

A matriz de confusão que se encontra na Tabela 13 trás métricas interessantes: a rede classificou corretamente 14030 dos ataques (VP) de um total de 14285, e 255 instâncias classificadas como normais pela rede, mas que na verdade são intrusões (FN). Percebe-se

Tabela 12 – Resultado da Rede LSTM - Holdout

Rede	Acurácia(%)	Erro	Precisão(%)	Recall(%)	F1-Score(%)
LSTM	98.63	0.0452	98.63	98.63	98,63

Fonte: Autoria Própria

também, que foram classificadas 152 instâncias como ataques, mas que na verdade são tráfego benigno (FP), e por fim tem-se 15267 de instâncias não intrusivas que foram corretamente classificadas.

Tabela 13 – Matriz de Confusão

	Positivo	Negativo	Total
Positivo	14030	255	14285
Negativo	152	15267	15319
Total	14182	15572	29704

Fonte: Autoria Própria

Na Figura 21 é mostrado o gráfico da acurácia por época de treinamento.

Na Figura 22 é mostrado o gráfico do erro por época.

5.2 EXPERIMENTO COM A GRU

Como explicado no Capítulo 3, a rede GRU é considerada em algumas literaturas como uma versão melhorada da LSTM. De forma semelhante ao treinamento da LSTM, o treinamento da rede seguiu a metodologia descrita no capítulo 4, alcançando seu resultado máximo em 100 épocas, após isso a rede começou a dar overfiting. Percebe-se por meio da tabela 14 que a rede se saiu ligeiramente melhor nas taxas de detecção de intrusão, alcançando uma acurácia de 98,73%, os resultados de ambas as redes serão comparadas na próxima Seção.

Tabela 14 – Resultado da Rede GRU - Holdout

Rede	Acurácia(%)	Erro	Precisão(%)	Recall(%)	F1-Score(%)
GRU	98.73	0.0446	98.73	98.73	98.73

Fonte: Autoria Própria

Na matriz de confusão da GRU, se comparada a matriz da LSTM, percebe-se que mesmo uma diferença de 0,1% nos resultados já trás uma diferença significativa na detecção

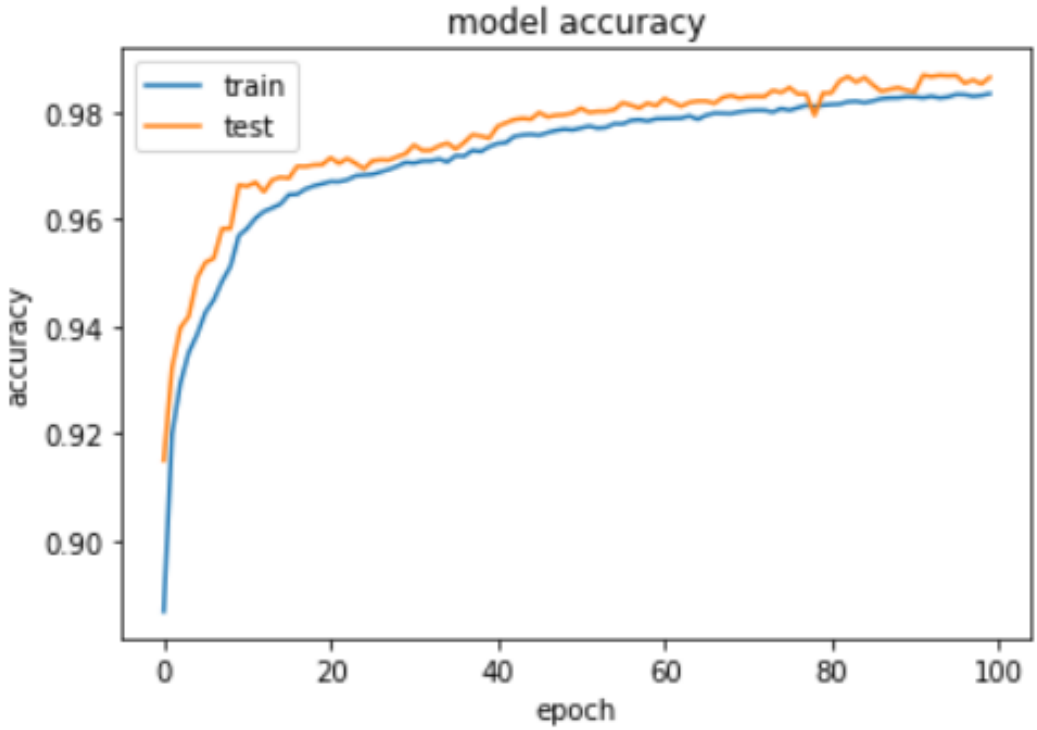

Figura 21 – Gráfico da Acurácia da Rede LSTM

Fonte: Autoria Própria

de intrusão. A rede classificou corretamente 14064 dos ataques (VP) de um total de 14285, e 221 instâncias classificadas como normais pela rede, mas que na verdade são intrusões (FN). Percebe-se também, que foram classificadas 156 instâncias como ataques, mas que na verdade são tráfego benigno (FP), e por fim 15263 de instâncias não intrusivas que foram corretamente classificadas.

Tabela 15 – Matriz de Confusão

	Positivo	**Negativo**	**Total**
Positivo	14064	221	14285
Negativo	156	15263	15419
Total	14285	15419	29704

Fonte: Autoria Própria

O gráfico da acurácia por época de treinamento pode ser visto na Figura 23, após as 40 épocas o desempenho a taxa de acurácia da rede se normaliza. Na Figura 24 vemos o gráfico do erro por época.

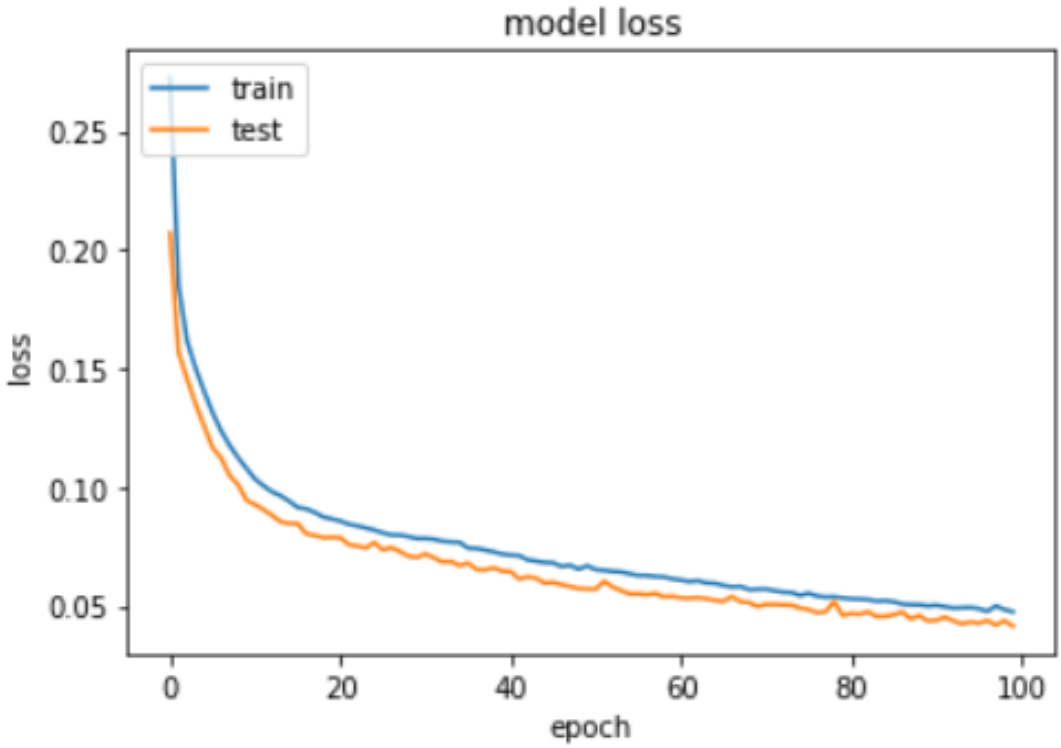

Figura 22 – Gráfico do Erro da Rede LSTM

Fonte: Autoria Própria

5.3 TABELA COMPARATIVA DE RESULTADOS E DISCUSSÃO

Esta seção apresenta a Tabela 16, composta pelos resultados obtidos pelas métricas utilizadas para validar o projeto nos experimentos de detecção de intrusão. Analisando de maneira geral, nota-se uma característica interessante para a igualdade entre a acurácia e f1-score para os experimentos, mostrando que as classes estão balanceadas. A acurácia da LSTM chegou a 98.63%, já o GRU foi sutilmente melhor, com uma acurácia de 98.73%..

Quando se trata de Segurança da Informação, dependendo do ataque que estiver sendo realizado, basta que o invasor tenha sucesso apenas 1 vez para conseguir controle do sistema invadido ou causar algum dano ao ativo.

Vale ressaltar, que para obter um bom resultado, a rede precisa ter uma baixa taxa de falsos positivos e falsos negativos. O objetivo do IPS é barrar o máximo de intrusões possíveis sem comprometer o tráfego normal, então mesmo a diferença mais sútil entre a acurácia das redes representa muito.

Essa diferença é vista na comparação entre as Matrizes de Confusão das Redes. A

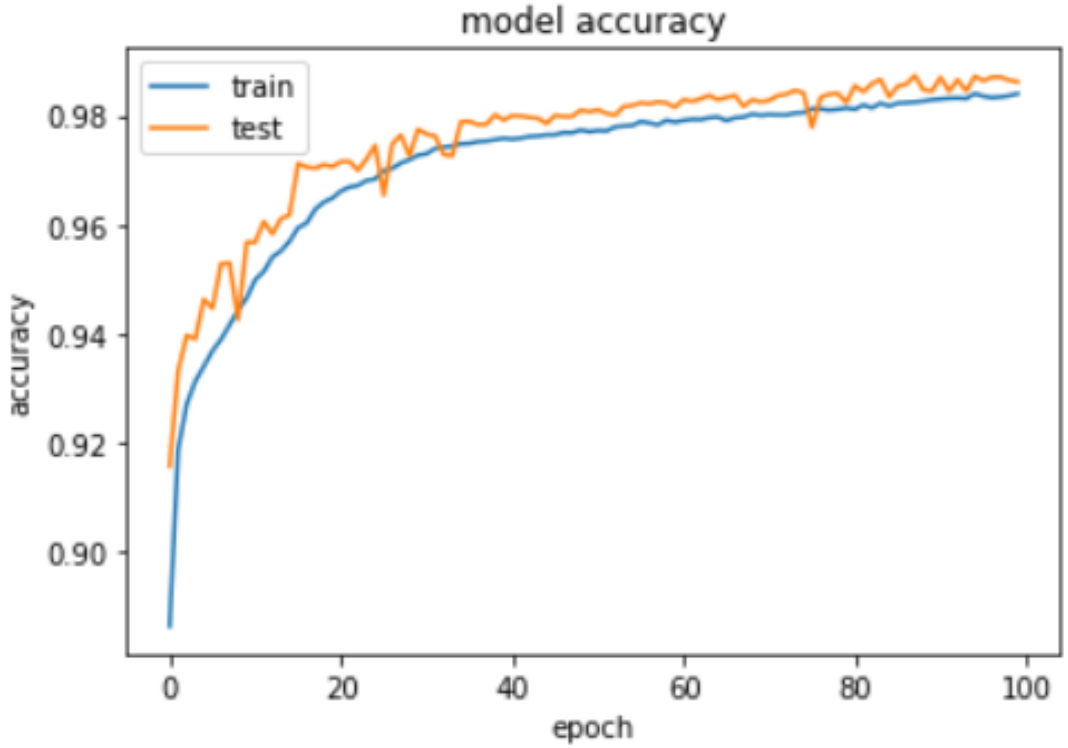

Figura 23 – Gráfico da Acurácia da Rede GRU

Fonte: Autoria Própria

Tabela 13 mostra que a LSTM não conseguiu detectar 255 ataques de intrusão, enquanto gerou 152 falsos positivos. Já a GRU detectou 34 ataques a mais que a LSTM e gerou 4 falsos positivo a mais.

Tabela 16 – Comparativo LSTM e GRU

Rede	Acurácia(%)	Erro	Precisão(%)	Recall(%)	F1-Score(%)
GRU	98.73	0.0446	98.73	98.73	98.73
LSTM	98.63	0.0452	98.63	98.63	98,63

Fonte: Autoria Própria

Um dos objetivos do trabalho é buscar boas taxas de detecção de ataques novos e que o sistema não gere falsos positivos, as redes propostas conseguiram chegar bem perto da condição ideal, que seria 100% de detecção e 0% de falso positivo. Para poder avaliar melhor os resultados, segue abaixo uma comparação com resultados obtidos no experimento de Choudhury e Bhowal (2015) utilizando outros tipos de tecnologia e que utilizaram a mesma base de dados.

Percebe-se neste comparativo, que o resultado das redes recorrentes se saíram melhores do que as demais redes, trazendo uma alta taxa de detecção, precisão, recall e

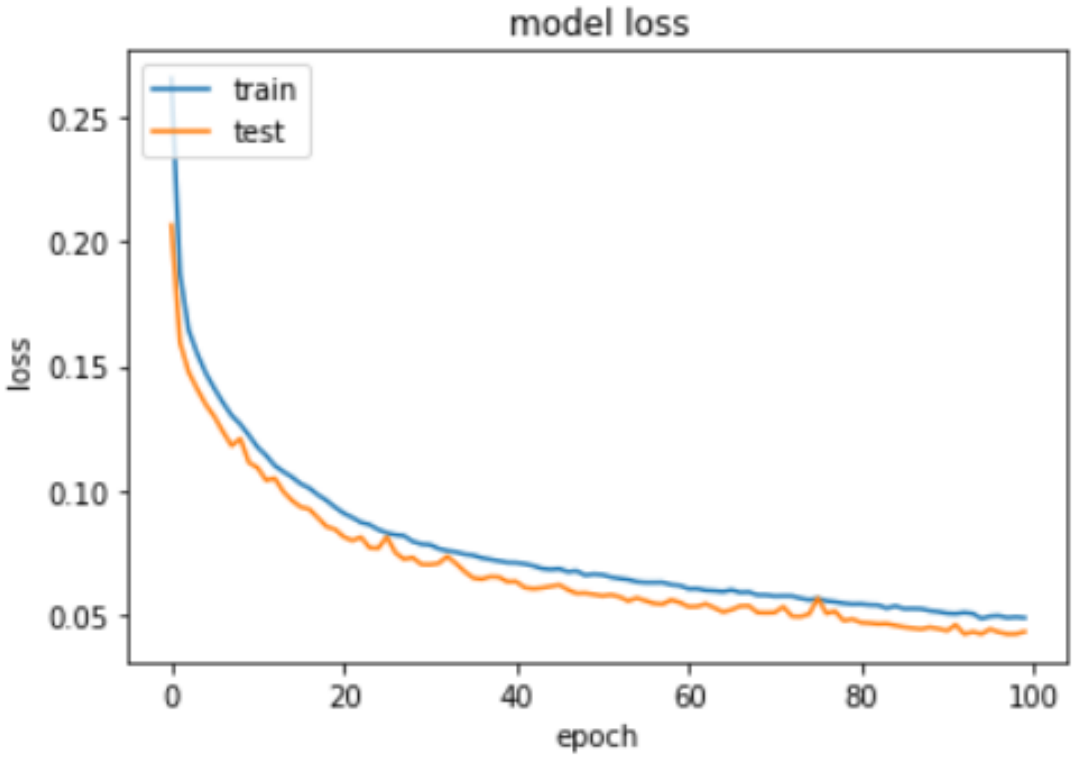

Figura 24 – Gráfico do Erro da Rede GRU

Fonte: Autoria Própria

f1-score, indicando uma maior confiabilidade na análise de tráfego malicioso em redes de computadores.

Tabela 17 – Comparativo dos Classificadores

Classificador	Acurácia(%)	Erro	Precisão(%)	Recall(%)	F1-Score(%)
GRU	**98.73**	0.0446	98.73	98.73	98.73
LSTM	**98.63**	0.0452	98.63	98.63	98,63
Random Forest	91.52	0.1135	95.10	88.68	91.70
RandomTree	90.88	0.9320	94.70	87.74	91.10
IBk	90.73	0.0934	93.60	88.68	91.00
BayesNet	90.67	0.1002	96.20	85.80	90.70
J48	89.68	0.1096	93.60	86.48	89.90

Fonte: Autoria Própria

6 CONCLUSÕES

Neste capítulo estão presentes as conclusões referentes as tarefas propostas de detecção de intrusão, verificando a satisfação de cada etapa definida como objetivo específico do projeto. Por fim, contempla possíveis abordagens futuras que possam utilizar as informações presentes neste projeto, dando origem a trabalhos futuros.

6.1 CONCLUSÕES

Os modelos avaliados não só tem uma forte capacidade de modelagem para detecção de intrusão, mas também tem alta precisão na classificação binária. O trabalho alcançou seus objetivos ao desenvolver um estudo comparativo entre técnicas de inteligência artificial na otimização de sistemas de detecção de intrusão em redes de computadores, a rede que obteve maior desempenho foi a GRU, alcançando taxas de detecção de 98,73%, os resultados mostraram que a rede obteve uma alta taxa de precisão e taxa de detecção com uma baixa taxa de falsos positivos.

Este trabalho teve como objetivo central agregar conhecimento para a detecção de intrusões em rede. Como abordado no Capítulo 1, este trabalho alcançou seus objetivos, mostrando que o LSTM e o GRU possuem altas taxas de detecção de tráfego malicioso e normal, aumento as taxas de detecções e reduzindo o número de falsos positivos na rede. Os resultados superaram o estado da arte, comparados aos trabalhos correlatos apresentados no Capítulo 3.

6.2 TRABALHOS FUTUROS

Durante a execução deste projeto, para os treinamentos voltados para a detecção de intrusão em redes, o processo de *fine tuning* da rede se mostrou satisfatório para aumentar a precisão classificatória dos experimentos. Sendo assim, em caso de tentar aumentar ainda mais a capacidade de reconhecimento da rede, testar outras formas de afinamento pode ser uma direção promissora. Como os ataques em redes estão sempre inovando e se reinventando, uma base de dados atualizada ou a análise de dados de tráfego em tempo real, pré-processados por alguma ferramenta de captura de pacotes.

Outro trabalho interessante seria o desenvolvimento de um *plug-in* para o Snort ou Suricata que foram descritos no Capítulo 2, de forma a testar o desempenho da rede em tráfego real e realizar novos ajustes.

REFERÊNCIAS

ASHFAQ, R. A. R.; WANG, X.-Z.; HUANG, J. Z.; ABBAS, H.; HE, Y.-L. Fuzziness based semi-supervised learning approach for intrusion detection system. **Inf. Sci.**, Elsevier Science Inc., New York, NY, USA, v. 378, n. C, p. 484–497, fev. 2017. ISSN 0020-0255. Disponível em: <https://doi.org/10.1016/j.ins.2016.04.019>.

AYCOCK, J. **Computer Viruses and Malware**. 1th. ed. Estados Unidos: Springer US, 2013.

BASTA, A.; BROWN, M.; BASTA, N. **Segurança de Computadores e Teste de Invasão**. 2th. ed. São Paulo: Cengage Learning, 2014.

BEAL, A. **Segurança da Informação. Princípios e Melhores Práticas Para a Proteção dos Ativos de Informação nas Organizações**. 1th. ed. Brasil: Atlas, 2005.

Bengio, Y.; Simard, P.; Frasconi, P. Learning long-term dependencies with gradient descent is difficult. **IEEE Transactions on Neural Networks**, v. 5, n. 2, p. 157–166, March 1994. ISSN 1045-9227.

BROWN, L.; STALLINGS, W. **Segurança de Computadores: Princípios e Práticas**. 1th. ed. Rio de Janeiro: Elsevier Editora Ltda, 2013.

BRUNEAU, G. The history and evolution of intrusion detection. **SANS Institute Information Security Reading Room**, 2001.

CHO, K.; MERRIëNBOER, B. van; GULCEHRE, C.; BOUGARES, F.; SCHWENK, H.; BENGIO, Y. Learning phrase representations using rnn encoder-decoder for statistical machine translation. 06 2014.

Choudhury, S.; Bhowal, A. Comparative analysis of machine learning algorithms along with classifiers for network intrusion detection. p. 89–95, 2015.

DIOGENES, Y.; MAUSER, D. **Certificação Security+. Da Prática Para o Exame SY0-401**. 3th. ed. Rio de Janeiro: Novaterra Editora e Distrubuidora Ltda, 2015.

FACELI, K.; LORENA, A. C.; GAMA, J.; CARVALHO, A. C. P. d. L. F. d. **Inteligência artificial: uma abordagem de aprendizado de máquina**. [S.l.]: LTC, 2011.

FERNANDES, A. M. **Inteligência artificial: noções gerais**. Visual Books, 2003. ISBN 85-7502-114-1. Disponível em: <http://search.ebscohost.com/login.aspx?direct=true&db=cat07269a&AN=utfpr.193146&lang=pt-br&site=eds-live&scope=site>.

FILHO, J. E. M. **Análise de Tráfego em Redes TCP/IP: Utilize tcpdump na análise de tráfegos em qualquer sistema operacional**. [S.l.]: Novatec, 2013.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep Learning**. [S.l.]: MIT Press, 2016. `http://www.deeplearningbook.org`.

HAYKIN, S. **Redes Neurais: Princípios e Práticas**. 2th. ed. Porto Alegre: [s.n.], 2011. Disponível em: <http://search.ebscohost.com/login.aspx?direct=true&db=edsmib&AN=edsmib.000011520&lang=pt-br&site=eds-live&scope=site>.

HETTICH, S.; BAY, S. D. **The UCI KDD Archive**. [S.l.]: University of California Department of Information and Computer Science, 1999. `http://kdd.ics.uci.edu`.

HINTON, G.; BENGIO, Y.; LECUN, Y. Deep learning. **Nature Publishing Group**, 2015.

HINTON, G.; OSINDERO, S.; TEH, Y.-W. A fast learning algorithm for deep belief nets. **Neural computation**, v. 18, p. 1527–54, 08 2006.

HOCHREITER, S. Untersuchungen zu dynamischen neuronalen netzen. **Diploma, Technische Universität München**, v. 91, n. 1, 1991.

HOCHREITER, S.; SCHMIDHUBER, J. Long short-term memory. **Neural computation**, v. 9, p. 1735–80, 12 1997.

KARPATHY, A. The unreasonable effectiveness of recurrentneural networks. **Andrej Karpathy Blog**, 2015. Disponível em: <http://karpathy.github.io/2015/05/21/rnn-effectiveness/>.

Kim, J.; Kim, J.; Thu, H. L. T.; Kim, H. Long short term memory recurrent neural network classifier for intrusion detection. In: **2016 International Conference on Platform Technology and Service (PlatCon)**. [S.l.: s.n.], 2016. p. 1–5.

KINGMA, D.; BA, J. Adam: A method for stochastic optimization. **International Conference on Learning Representations**, 12 2014.

LIN, W. H.; LIN, H. C.; WANG, P.; WU, B. H.; TSAI, J. Y. Using convolutional neural networks to network intrusion detection for cyber threats. **Proceedings of 4th IEEE International Conference on Applied System Innovation (ICASI)**, IEEE, p. 1107–1110, 2018.

MCHUGH, J. Testing intrusion detection systems: A critique of the 1998 and 1999 darpa intrusion detection system evaluations as performed by lincoln laboratory. **ACM Trans. Inf. Syst. Secur.**, ACM, New York, NY, USA, v. 3, n. 4, p. 262–294, nov. 2000. ISSN 1094-9224. Disponível em: <http://doi.acm.org/10.1145/382912.382923>.

MITNICK, K.; SIMON, W. **A Arte de Enganar**. 1th. ed. Brasil: Pearson Universidades, 2003.

NORVIG, P.; RUSSELL, S. **Inteligência Artificial**. 3th. ed. Brasil: Elsevier, 2013.

OLAH, C. **Understanding LSTM Networks**. 2015. Disponível em: <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>.

SÊMOLA, M. **Gestão da segurança da informação**. 2th. ed. Brasil: Elsevier, 2013.

SHAH, S. A. R.; ISSAC, B. Performance comparison of intrusion detection systems and application of machine learning to snort system. **Elsevier BV**, 2018.

SILVA, I. N.; SPATTI, D. H.; FLAUZINO, R. A.; LIBONI, L. H. B.; ALVES, S. F. dos R. **Artificial Neural Networks: A Practical Course**. 1st. ed. [S.l.]: Springer Publishing Company, Incorporated, 2016. ISBN 3319431617, 9783319431611.

SILVA, R. M. **Redes Neurais Artificiais aplicadas à Detecção de Intrusão em Redes TCP/IP**. Dissertação (Mestrado) — PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO - PUC-RIO, 2005.

SRIVASTAVA, N.; HINTON, G.; KRIZHEVSKY, A.; SUTSKEVER, I.; SALAKHUTDINOV, R. Dropout: A simple way to prevent neural networks from overfitting. **J. Mach. Learn. Res.**, JMLR.org, v. 15, n. 1, p. 1929–1958, jan. 2014. ISSN 1532-4435.

STAUDEMEYER, R. Applying long short-term memory recurrent neural networks to intrusion detection. **South African Computer Journal**, v. 56, 07 2015.

TAVALLAEE, M.; BAGHERI, E.; LU, W.; GHORBANI, A. A detailed analysis of the kdd cup 99 data set. **Second IEEE Symposium on Computational Intelligence for Security and Defense Applications (CISDA)**, IEEE, 2009.

YIN, C.; ZHU, Y.; FEI, J.; HE, X. A Deep Learning Approach for Intrusion Detection Using Recurrent Neural Networks. **IEEE**, v. 5, p. 21954–21961, 2017.