UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE ENGENHARIA ELÉTRICA

Heitor Eugênio Gonçalves

Comparação da eficiência de modelos de Redes Neurais Artificiais na detecção de intrusões em redes de computadores

Uberlândia

2023

Heitor Eugênio Gonçalves

Comparação da eficiência de modelos de Redes Neurais Artificiais na detecção de intrusões em redes de computadores

Trabalho de conclusão de curso apresentado ao Curso de Graduação em Engenharia Eletrônica e de Telecomunicações da Universidade Federal de Uberlândia como parte dos requisitos para a obtenção do título de Engenheiro Eletrônico e de Telecomunicações grau bacharelado.

Área de concentração: Redes de Computadores.

Orientador: Prof. Dr. Éderson Rosa da Silva

Uberlândia-MG
2023

Dedicado à minha família, em especial à minha mãe, aos meus amigos e colegas de graduação, ao meu orientador e professor Éderson e a todos os demais profissionais da Universidade Federal de Uberlândia.

Agradecimentos

Agradeço ao meu professor e orientador Éderson por todos os seus ensinamentos e pelo tempo dedicado apoiando-me nesta pesquisa.

Agradeço à minha mãe, que sempre me incentivou a estudar.

Sou grato ao corpo docente da Universidade Federal de Uberlândia por todos os aprendizados que recebi durante os meus anos de graduação. Também agradeço aos demais funcionários da universidade por todos os serviços prestados.

Resumo

As Redes Neurais Artificiais (RNAs) são empregadas em diversas áreas, encontrando padrões, otimizando dados, prevendo ações, entre outros usos. Na segurança de rede de computadores, o uso de RNAs têm sido empregadas com o intuito de detectar intrusões, prevenindo ataques a rede. Porém, existem inúmeros modelos de RNAs, de modo que cada tipo de ataque pode ter um modelo que alcance melhores resultados que outros. Neste trabalho, são implementados 5 modelos de RNAs, que são usadas para detectar 7 tipos de ataques. Os resultados deste trabalho apontam, para cada RNA e para cada ataque, a quantidade de falsos positivos e negativos, acurácia e tempo de execução. Dessa forma, os resultados auxiliam projetistas de *softwares* de segurança de rede na escolha do modelo de RNA mais adequado.

Palavras chaves: aprendizado de máquinas, detecção de intrusões, redes de computadores, Redes Neurais Artificiais, segurança da informação.

Abstract

Artificial Neural Networks (ANNs) are used in several areas, finding patterns, optimizing data, predicting actions, among other uses. In computer network security, the use of ANNs has been employed in order to detect intrusions, preventing network attacks. However, there are numerous models of ANNs, so that each type of attack can have a model that achieves better results than others. In this work, ANNs of 5 ANN models are implemented, which are used to detect 7 types of attacks. The results of this work indicate, for each ANN and for each attack, the amount of false positives and negatives, accuracy and execution time. Thus, the results help network security software designers in choosing the most appropriate ANN model.

Keys: Artificial Neural Networks, computer networks, information security, intrusion detection, machine learning.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Siglas

2FA Duplo Fator de Autenticação
AD Active Directory
Adam Adaptive Moment Estimation
AE Autoencoders
BI-LSTM Bidirectional Long Short-Term Memory
CART Classification and regression tree
CIC-IDS2017 Canadian Institute for Cybersecurity - Intrusion Detection Systems
CNN Convolution Neural Network
CPU Unidade de Processamento Central
DAE Denoising Autoencoder
DC Domain Controller
DDoS Distributed Denial of Service
DOM Document Object Model
DoS Denial of Service
FN Falsos Negativos
FP Falsos Positivos
GMM Gaussian Misture Model
GPU Unidade de Processamento Gráfica
IDE Integrated Development Environment
KNN K-Nearest Neighbor
LOIC Low Orbit Ion Canon
LSTM Long Short-Term Memory
MLP Multilayer Perceptron
OWASP Open Web Application Security Project
ReLU Rectified Linear Unit
RNA Rede Neural Artificial
RNR Rede Neural Recorrente
SOM Self Organizing Map
XSS Cross-Site Scripting

Capítulo 1

Introdução

As Redes Neurais Artificiais (RNAs) possuem diversas aplicações, podendo prever resultados, otimizar dados, agrupar informações, entre outras. Por esses motivos, as RNAs têm sido empregadas em inúmeros setores, incluindo o da segurança da informação.

Por conseguir classificar informações, as RNAs podem ser usadas em detecção de intrusões em redes de computadores. Para isso, elas recebem como entrada um conjunto de informações de tráfego e, assim, consegue classificá-lo como ataque ou não.

Quando um ataque não é conhecido pelo *software* de segurança da rede, ele precisa analisar similaridade desse tráfego com outros ataques presentes em seu banco de dados. Nesse contexto, as RNAs destacam-se como uma excelente medida para detectar os ataques, pois por meio de aprendizado de intrusões conhecidas, elas conseguem classificar novos tráfegos como ataque ou não.

Devido a grande variedade de modelos de RNAs e de tipos de ataques, a tarefa de escolher a Rede Neural mais adequada pode ser complicada. Para exemplificar, salienta-se que existem redes com memória, que têm um melhor desempenho em ataques com uma relação temporal entre eles. Além disso, há RNA que agrupa os dados de acordo com suas semelhanças, sendo útil em ataques com elevada similaridade. Conclui-se, assim, que a depender do ataque um modelo de RNA pode ser bom ou ruim.

Além da acurácia de uma RNA para detectar um ataque, outro critério que deve ser considerado ao realizar o projeto de um *software* de segurança de redes é o custo computacional da RNA. Assim, destaca-se que é essencial que a RNA tenha um bom desempenho associado a uma utilização rápida visando não causar grande latência na rede, mesmo que o treinamento da RNA seja custoso.

É importante ressaltar que além da acurácia e do tempo de execução, é preciso identificar a quantidade de falsos positivos e de falsos negativos gerados por uma RNA. O primeiro termo refere-se aos pacotes que não são ataques e, erroneamente, foi classificado como tal. Em contraposição, os falsos negativos são pacotes que são ataques que não foram identificados.

Os falsos positivos podem causar o descarte indevido de pacotes legítimos, interrompendo a comunicação. Enquanto isso, os falsos negativos permitem que um ataque passe pela RNA, o

que seria o pior cenário possível.

Neste trabalho, busca-se fornecer informações de tempo de execução, acurácia e porcentagem de falsos negativos e positivos de 5 modelos de RNAs para diferentes tipos de ataques. Desse modo, a tarefa de escolher a RNA mais adequada para realizar a segurança de uma rede é facilitada.

Para realizar o treinamento das RNAs foi utilizado um banco de dados com 7 tipos de ataques: DoS (*Denial of Service*), DDoS (*Distributed Denial of Service*), Botnet, ataque Web, força bruta, infiltração e varredura de portas. Esse banco de dados é disponibilizado pelo (CIC-IDS2017) *Canadian Institute for Cybersecurity* [36], sendo uma base de dados referência no treinamento de RNA, com diversos trabalhos empregando-a, conforme mostrado na Seção 1.1.

1.1 Revisão Bibliográfica

Em Thapa et al. [1], os seguintes modelos de *machine learning* e *deep learning* são utilizados: KNN (*K-Nearest Neighbor*), XGBoost, CART (*Classification and regression tree*), CNN (*Convolution Neural Network*) e LSTM (*Long Short-Term Memory*). Em geral, o modelo que teve na maioria dos casos os melhores resultados foi a CART, que é um modelo de *machine learning*, mas não é classificada como RNA.

O trabalho de Thapa et al. usa o mesmo banco de dados empregado nesta pesquisa, o CIC-IDS2017. Entretanto, ele apresenta os resultados apenas para três tipos de ataques: DDoS, varredura de portas e Botnet, enquanto este trabalho inclui mais outros quatro ataques, totalizando sete diferentes formas de intrusão. Salienta-se ainda, que por usar Redes Neurais Artificiais Profundas (*deep learnings*), o tempo de execução foi bastante elevado comparado as RNAs desenvolvidas nesta pesquisa.

Em Jiyeon kim et al. [2] uma CNN é desenvolvida para detectar diferentes tipos de ataques. Ao analisar as acurácias de cada intrusão, observou-se um desempenho ruim em ataques Web, ataques de Infiltração e DoS.

O trabalho de Jiyeon kim et al. usa 79 informações de tráfego de rede como entrada da CNN. Todavia, essa grande quantidade de informações nem sempre está disponível para que o *software* de detecção possa desempenhar seu papel. Desta maneira, nesta pesquisa, optou-se por usar apenas 20 informações de tráfego.

No trabalho de Chin-Shiuh Shieh et al. [3] uma RNA do tipo Bidirectional Long Short-Term Memory (BI-LSTM) é combinada com um Gaussian Misture Model (GMM) para realizar a detecção de ataques DDoS. Para treinamento da RNA, foram usados os banco de dados CIC-IDS2017 e CIC-DDoS2019. A acurácia para os dados de DDoS do CIC-IDS2017 foi de 98,2% para a combinação de BI-LSTM e GMM, o que como pode ser visto no Capítulo 4, é menor que a obtida nesta pesquisa.

Em Zachariah Pelletier et al. [4] um algoritmo de *machine learning* chamado *Random Forest* e uma RNA são empregados para detectar intrusões usando o banco de dados CIC IDS-2017.

Nesse trabalho, apesar de ser possível identificar excelentes valores de acurácia, notou-se a ausência de tempos de execução e da matriz de confusão, o que impede a análise completa do desempenho do algoritmo. Afinal, caso a quantidade de amostras normais for muito maior que a de ataque, mesmo que nenhum ataque seja detectado, a acurácia ainda seria alta.

No trabalho de Adan Shahid Khan et al. [5] é proposta uma Rede Neural Convolucional Profunda que usa imagens de espectrogramas geradas por meio da transformada de Fourier de curta duração. Essa RNA obteve uma acurácia de 98,758%. Esse trabalho emprega o banco de dados CIC-IDS2017, porém, ele não utiliza os ataques de infiltração para treinar a RNA.

1.2 Motivações da Pesquisa

Como visto na Seção 1.1, numerosos artigos demonstram resultados de implementações de RNAs na detecção de intrusões. No entanto, falta um trabalho que compare modelos de RNAs distintos sobre as mesmas circunstâncias, ou seja, usando funções semelhantes e número de nós próximos, de modo que a diferença esteja no modelo em si, e não em um parâmetro que faltou em um modelo e não em outro. Essa comparação deve abordar de forma ampla três critérios de grande importância na construção de um *software* de segurança de redes: tempo de execução, acurácia e porcentagem de falsos negativos e positivos.

1.3 Objetivos

O objetivo desta pesquisa é determinar qual o modelo de RNA mais adequado para diferentes tipos de tentativas de intrusão que uma rede pode sofrer. Além disto, busca-se encontrar situações em que as RNAs podem ter um desempenho ruim na detecção de ataque,mostrando quando não devem ser empregadas.

Outro fator a ser discutido sobre as RNAs é o tempo de execução. Dessa forma, será possível verificar o atraso no tráfego causado por elas.

Acentua-se ainda que esse trabalho, além de comparar modelos de RNA, verifica a eficiência dos parâmetros usados nas RNAs, como funções de ativação, otimizador e função de custo.

1.4 Organização do Trabalho

O restante do trabalho está organizado conforme descrito a seguir. No Capítulo 2 aborda os fundamentos teóricos dos modelos de Redes Neurais Artificiais, bem como seus parâmetros, e dos ataques usados neste trabalho. Em seguida, no Capítulo 3, encontra-se a metodologia empregada, descrevendo o banco de dados usado e a arquitetura de cada modelo de rede.

Posteriormente, no Capítulo 4 são apresentados os resultados, que incluem tempo de execução, matriz de confusão, acurácia e porcentagem de falsos positivos e negativos. Por fim,

o Capítulo 5 mostra as conclusões da realização dessa pesquisa, como também as possíveis melhorias que podem ser feitas em trabalhos futuros.

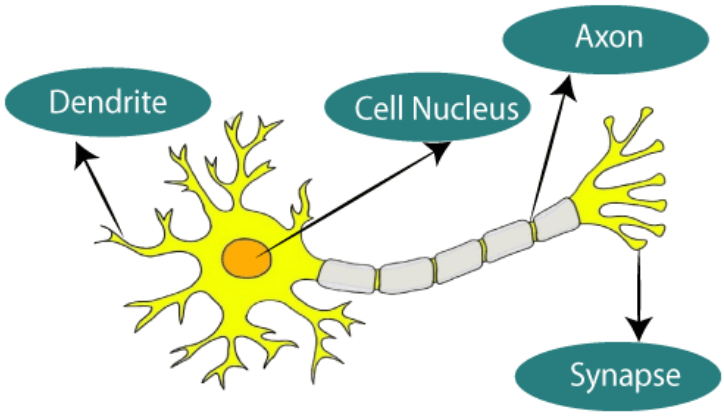
Capítulo 2

Fundamentação Teórica

2.1 Redes Neurais Artificiais

As Redes Neurais Artificiais são técnicas computacionais que se baseiam no funcionamento do sistema nervoso humano. A Figura 2.1 apresenta o diagrama de um neurônio biológico. Nesta figura, aparecem os dendritos, que recebem sinais elétricos de outros neurônios, o corpo celular, que tem a função de receber e integrar os estímulos e é onde está localizado o núcleo e o citoplasma. Por fim, há o axônio, que é um prolongamento extenso que transmite os impulsos elétricos, e a sinapse, que é a junção entre o neurônio e outra célula.

Figura 2.1: Estrutura de um neurônio.



Fonte: [6].

As RNAs são constituídas por associação de camadas por meio de pesos. A primeira camada recebe os dados de entrada. A partir destes dados a RNA é capaz de realizar uma previsão, agrupar os dados em grupos, reconhecer um padrão, otimizar resultados, entre inúmeras outras ações. As outras camadas da rede são constituídas por nós, que, por meio de funções matemáticas, simula o funcionamento do corpo celular de um neurônio. Dessa forma, pode-se relacionar as RNAs aos neurônios biológicos conforme a Tabela 2.1.

Tabela 2.1: Relação entre o neurônio biológico e a RNA.

Biológico	**Artificial**
Dendritos	Entradas
Corpo Celular	Nós
Axônio	Saídas
Sinapse	Pesos

Uma rede neural pode ser supervisionada ou não supervisionada. O primeiro tipo é caracterizado pelo aprendizado da rede baseado em comparações das saídas da RNA com valores que eram esperados. Por exemplo, caso uma rede seja usada para determinar se uma bola de futebol aparece em uma imagem, durante o treinamento da RNA, apresenta-se várias imagens à rede, sendo que algumas têm uma bola e outras não. Como são conhecidas as imagens que têm a bola, é verificado na saída da rede se ela classificou corretamente a imagem por meio do cálculo de um erro entre a previsão da RNA e a saída esperada. Por meio do cálculo desse erro, é possível reajustar os pesos da rede para chegar em um erro menor, permitindo, assim, o aprendizado desta.

Por outro lado, para a RNA não supervisionada, não se sabe se a saída em cada interação da rede está certa ou errada. Em geral, esse tipo de rede é usado para categorizar dados sem mostrar como essa categorização deve ocorrer.

2.1.1 Uma breve história sobre Redes Neurais Artificiais

A primeira rede neural foi modelada usando circuitos elétricos em 1943 pelo neurofisiologista Warren MacCulloch e o matemático Walter Pitts. Em 1949, Donald Hebb publicou um trabalho sobre o melhoramento das redes cada vez que são utilizadas, o que está relacionado a ideia de aprendizado da rede por meio do treinamento.

Em 1958, Frank Rosenblatt cria o Perceptron. Este é um algoritmo simples que utiliza apenas duas camadas com cálculos de soma e subtração para reconhecer padrões. Em 1959, Bernard Widrow e Marcian Hoff desenvolveram, os modelos ADALINE e MADALINE. O primeiro foi utilizado para reconhecer padrões binários, enquanto o segundo foi aplicado para elaboração de um filtro adaptativo que elimina os ecos nas linhas telefônicas.

Na década de 1980, Kunihiko Fukushima desenvolveu uma rede hierarquia e multicamada. Esta rede foi empregada em problemas de reconhecimentos de padrões. Ainda nessa década, David E. Rumelhart e James L. McClelland apresentaram o algoritmo de otimização *backpropagation*.

O surgimento da GPU (Unidade de Processamento Gráfica) em 1999, permitiu uma evolução bastante significativa na utilização de RNA, garantindo a empregabilidade de redes neurais

profundas. Isso se deve ao fato das GPUs terem mais processadores que as CPUs (Unidades de Processamento Central) e executarem várias operações em paralelo [7, 8].

Atualmente, as RNAs são aplicadas nas mais diversos áreas. Cita-se, por exemplo, a utilização de RNAs por parte dos bancos para detectar fraude em cartões de crédito. Outro exemplo, é o reconhecimento de imagens por meio de redes neurais que permitiu a criação de carros autônomos. Ademais, diversas redes sociais utilizam redes neurais para categorizar seus usuários e, desse modo, ter um marketing direcionado. Além disso, por meio das RNAs, mais especificamente, redes neurais de aprendizado profundo, foi possível construir robôs humanoides extremamente realísticos, como é o caso do Ameca, desenvolvida pela empresa Engineered Arts [9]. Este robô apresenta movimentos suaves e realistas gerados em tempo real de acordo com os dados captados por seus sensores. Existem inúmeras outras empregabilidades das RNAs, como no mercado de ação para previsão de preços, diagnóstico de imagens médicas e reconhecimento voz.

2.1.2 Parâmetros de uma Rede Neural

Em geral, os parâmetros mais importantes de uma rede neural são: função de ativação, otimizador, número de épocas e função de custo.

Função de Ativação

A função de ativação é responsável pelo processamento de cada nó. A função a ser empregada no nó depende da aplicação. A seguir são apresentadas as funções de ativação empregadas nas RNAs desenvolvidas neste trabalho.

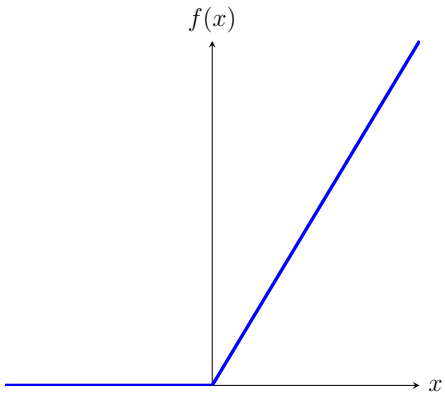
1) Função ReLU (Rectified Linear Unit)

É uma unidade linear retificada, isto é, o seu valor é 0 para entradas negativas e linear para valores maiores ou iguais a zero. Isso permite que apenas nós com valores positivos sejam ativados. Matematicamente, tem-se:

$$f(x) = \max(0,\, x). \tag{2.1}$$

A Figura 2.2 mostra a forma da função ReLU.

Figura 2.2: Função ReLU.



A função ReLU é amplamente utilizada nas camadas ocultas de redes profundas. Isso se deve ao fato de o gradiente desta função ser maior do que funções como a sigmoide. Funções cujo o gradiente é baixo, como a sigmoide, resulta em um dissipação do gradiente, isto é, a retropropagação não é realizada de forma efetiva para as primeiras camadas ocultas.

Outra vantagem da função ReLU é o seu baixo custo computacional, pois ela utiliza apenas o primeiro quadrante, e seu valor é a própria entrada neste.

A desvantagem da função ReLU é não ter aprendizado quando os pesos são negativos ou zero. Desse modo, ao utilizar essa função, deve-se evitar inicializar toda a rede com valores menores ou iguais as zero [10].

2) Função Softplus

A função softplus é uma suavização da ReLU. Essa suavização ocorre na transição entre os valores negativos e positivos, que no caso da ReLU é não linear.

A função softplus é dada por:

$$f(x) = \frac{1}{\beta} \cdot \log(1 + e^{\beta \cdot x}), \tag{2.2}$$

onde o β é o parâmetro responsável por determinar o quão suave a curva é. Em geral, o β é usado com o valor 1.

Ao contrário da função ReLU, a softplus é derivável em zero e não ignora os valores negativos. Desse modo, essa função permite a aprendizado para valores negativos e zero.

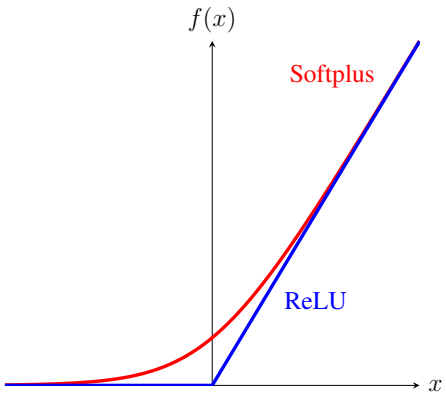
A derivada da função softplus é dada pela seguinte equação:

$$f'(x) = \frac{1}{1 + e^{x}} \cdot e^{x} = \frac{e^{x}}{1 + e^{x}} = \frac{1}{1 + e^{-x}}, \tag{2.3}$$

note que o resultado desta derivada é a função sigmoide.

A Figura 2.3 compara os gráficos da função softplus e da função ReLU. Note que para valores distantes da origem, o comportamento é o mesmo para ambas funções.

Figura 2.3: Função Softplus.



3) Função Sigmoide

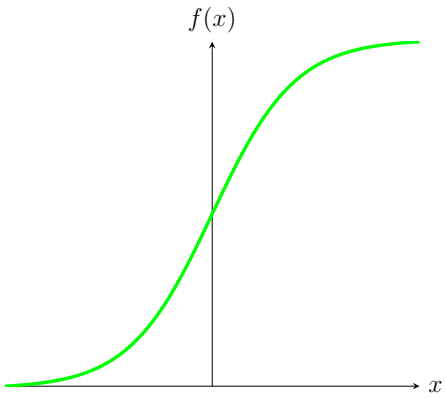
A função sigmoide é calculada por:

$$\sigma(x) = \frac{1}{1 + e^{-x}}. \tag{2.4}$$

A Figura 2.4 apresenta o gráfico da função sigmoide. Percebe-se que a imagem está no intervalo (0,1). Assim, mesmo se a entrada for um valor de módulo muito grande, a saída estará entre 0 e 1. Isso permite que a função de ativação não "exploda", isto é, tenha valores tendendo à $\pm\infty$.

Outra vantagem da função sigmoide, é sua curva em formato de "S"permitir a separação de classes não linearmente separáveis. Ressalta-se no entanto, que a sigmoide, devido ao cálculo exponencial em sua fórmula, tem um custo computacional maior que as funções lineares.

Figura 2.4: Função Sigmoide.



Função de custo

A função de custo é usada em RNAs supervisionadas para calcular o erro do valor previsto pela RNA e o valor esperado. Esta função permite verificar o desempenho da rede durante o treinamento, de modo a ajustar os pesos para minimizá-la.

Quando o objetivo é realizar previsões binárias, é comum utilizar a entropia cruzada binária como função de custo. Esta função é dada por:

$$Loss = -\frac{1}{n}\sum_{i=1}^{n}[y_i \cdot \log \hat{y}_i + (1 - y_i) \cdot \log(1 - \hat{y}_i)], \tag{2.5}$$

onde y_i é o valor previsto, $\hat{y}_i$ o valor esperado e n é o número de amostras. Nota-se pela Equação (2.5) que a função de custo entropia cruzada binária é sempre positiva e tende a zero quando y_i se aproxima de $\hat{y}_i$.

Otimizador

Os otimizadores são algoritmos responsáveis por atualizarem os atributos da RNA, como pesos e taxa de aprendizado, com o intuito de obter o mínimo global da função de custo. Na prática, nem sempre é encontrado o mínimo global, mas um valor suficientemente pequeno para o erro.

O processo de ajustes dos pesos após determinar o valor da função de custo na saída é chamado de *backpropagation*. Esta técnica consiste em atualizar os parâmetros partindo do fim até o começo da rede.

Uma forma de minimizar a função de custo por meio de *backpropagation* é usar o seu gradiente como base para ajustar dos pesos [12]. Isso é feito pelo fato da maior taxa de variação de uma função ocorrer na direção e sentido de seu vetor gradiente. Logo, o gradiente permite chegar mais rapidamente ao mínimo da função de custo.

A Descida de gradiente pode ser dividida em três variantes:

- Descida de gradiente em lote − Calcula o gradiente da função de custo $J(\theta)$ para os parâmetros θ da rede para todo o conjunto de dados de treinamento. Matematicamente, tem-se a atualização dos parâmetros θ da seguinte forma:

$$\theta = \theta_{t-1} - \eta \cdot \nabla_\theta J(\theta_{t-1}). \tag{2.6}$$

  Pelo fato de usar todo o conjunto de dados, essa variante de descida de gradiente é lenta e usa muito espaço de memória.

- Descida do gradiente estocástico − Calcula o gradiente da função de custo $J(\theta)$ para os parâmetros θ da rede para cada parte (lote) do conjunto de dados de treinamento $x^{(i)}$ e $y^{(i)}$. Matematicamente, tem-se:

$$\theta = \theta_{t-1} - \eta \cdot \nabla_\theta J(\theta_{t-1}; x^{(i)}; y^{(i)}). \tag{2.7}$$

  Esse tipo de variante torna a execução do algoritmo mais rápida em comparação com a descida do gradiente em lotes.

- Descida do gradiente em mini-lote − Obtém o gradiente da função de custo $J(\theta)$ para os parâmetros θ da rede para as n partes do conjunto de dados de treinamento $x^{(i)}$ e $y^{(i)}$. Ou seja,

$$\theta = \theta_{t-1} - \eta \cdot \nabla_\theta J(\theta; x^{(i:i+n)}; y^{(i:i+n)}). \tag{2.8}$$

  Como o cálculo do gradiente e atualização dos parâmetros são feitos para todos os lotes de treinamento, a variância das atualizações dos parâmetros é reduzida. Isso leva a uma convergência mais estável.

Para acelerar a descida do gradiente na direção relevante e amortecer as oscilações, muitos algoritmos de otimização utilizam um método conhecido por momento. Este método atualiza os parâmetros θ da rede da seguinte forma:

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta_{t-1}), \tag{2.9a}$$

$$\theta = \theta_{t-1} - v_t, \tag{2.9b}$$

onde o termo γ normalmente é definido como 0,9 ou um valor próximo deste.

O algoritmo de otimização utilizado neste trabalho é o Adam(*Adaptive Moment Estimation*) [13]. Este algoritmo é uma variação do gradiente descendente e usa os momentos m_t e v_t. Estes termos são, respectivamente, a média e a variância não centrada dos gradientes, e suas equações são:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t, \tag{2.10a}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2, \tag{2.10b}$$

em que β_1 e β_2 são os coeficientes de decaimento e o g_t é o gradiente da função de custo em t.

Os primeiros momentos se aproximam de zero, pois são inicializados com o valor zero. Para evitar isso, utiliza-se os coeficientes $\hat{m}_t$ e $\hat{v}_t$, que são dados por:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}, \tag{2.11a}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}, \tag{2.11b}$$

Os parâmetros da rede são atualizados pelo modelo de Adam da seguinte forma:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t, \tag{2.12}$$

onde $\epsilon \ll 1$ evita que o denominador seja zero.

Em Kingma et al. [13] é indicado adotar $\beta_1 = 0,9$, $\beta_2 = 0,999$ e $\epsilon = 10^{-7}$. Dessa forma, neste trabalho optou-se por também utilizar esses valores para os parâmetros.

O Algoritmo 10 mostra de forma genérica como deve ser a implementação do Adam.

Algoritmo 1: Adam

inicializa $\theta_0, m_0 \leftarrow 0, v_0 \leftarrow 0, t \leftarrow 0$
enquanto θ *não convergir* **faça**
  $t \leftarrow t + 1$
  $g_t \leftarrow \nabla_\theta J(\theta_{t-1})$
  $m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$
  $v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$
  $\hat{m}_t \leftarrow \frac{m_t}{1-\beta_1^t}$
  $\hat{v}_t \leftarrow \frac{v_t}{1-\beta_2^t}$
  $\theta_{t+1} \leftarrow \theta_t - \frac{\eta}{\sqrt{\hat{v}_t}+\epsilon}\hat{m}_t$
fim

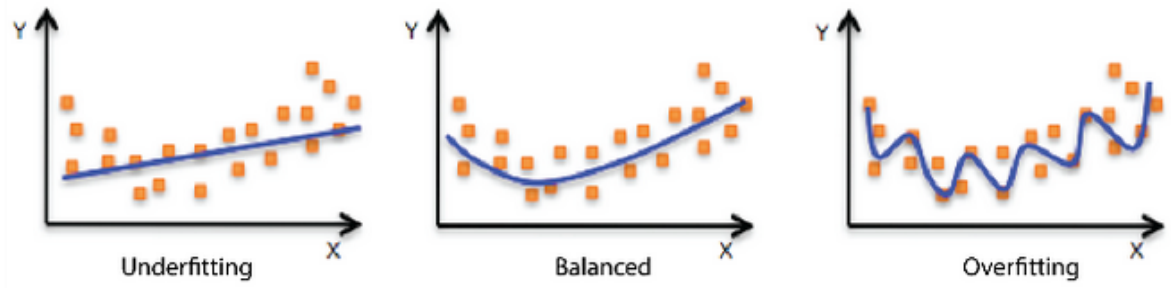
Número de épocas

O número de épocas refere-se a quantas vezes o conjunto de dados de treinamento passam pela rede para que ela possa ter seus pesos ajustados. Intuitivamente, pode-se pensar que quanto mais épocas, melhor será o desempenho da rede. No entanto, um número de épocas muito elevado torna a RNA excessivamente acomodada com os dados de treinamento. Como uma analogia, pode-se interpretar como a modulação do ruído presente em um sinal, o que não é o objetivo por reduzir a generalização. Assim, quando os dados são novos, a rede apresenta um resultado ruim. Esse tipo de evento é denominado *overfitting*.

Um número de épocas muito baixo pode fazer com que a rede não alcance a minimização da função de custo para um valor razoável, o que é chamado de *underfitting*. Por esse motivo, a RNA deve ser treinada com diferentes números de épocas e pesos iniciais até chegar no melhor resultado.

A Figura 2.5 apresenta os gráficos dos modelos de previsão em cenários de *underfitting*, *overfitting* e número de épocas balanceado - quantidade que acompanha de maneira mais adequada o comportamento do conjunto de dados.

Figura 2.5: Modelos de previsão em cenários de *underfitting*, *overfitting* e número de épocas balanceado.



Fonte: [11].

2.2 Modelos de Redes Neurais Artificiais

Neste trabalho foram usados 5 modelos de redes neurais: Perceptron de Múltiplas Camadas, Rede Neural Convolucional, Memória de Curto Prazo Longa, Autoencoder e Mapas Auto-Organizáveis. Estes modelos são detalhados a seguir.

2.2.1 Perceptron de Múltiplas Camadas

O Perceptron de Múltiplas Camadas (em inglês, *Multilayer Perceptron* - MLP) é constituído por uma camada de entrada, uma ou mais camadas ocultas e uma camada de saída. Esta camadas apresentam um elevado grau de conectividade, sendo muito comum adotar redes densas. Neste tipo de rede, um nó em qualquer camada da rede tem conexão com todos nós da camada anterior.

A camada de entrada recebe os dados utilizados pela rede. Em seguida, os nós desta camada são conectados aos nós da camada oculta, também chamada de camada intermediária, que extraem e acentuam as características mais significativas antes de enviar os vetores de entrada para a saída. Por fim, a camada de saída pode ter um ou mais nós para fornecer algum resultado referente aos dados de entrada. Este resultado pode ser uma classificação, uma regressão linear, entre outros tipos.

O MLP foi inspirado no Perceptron. Este modelo contém apenas a camada de entrada e realiza uma classificação binária a partir de um aprendizado supervisionado. A maior desvantagem do Perceptron é o fato dele não conseguir classificar dados não lineares, o que foi alcançado com o uso da rede MLP.

O Perceptron de Múltiplas Camadas usa a técnica de *backpropagation* para realizar o aprendizado da rede. No caso, o algoritmo utilizado, nesta, e em todas redes com aprendizado supervisionado, será o Adam.

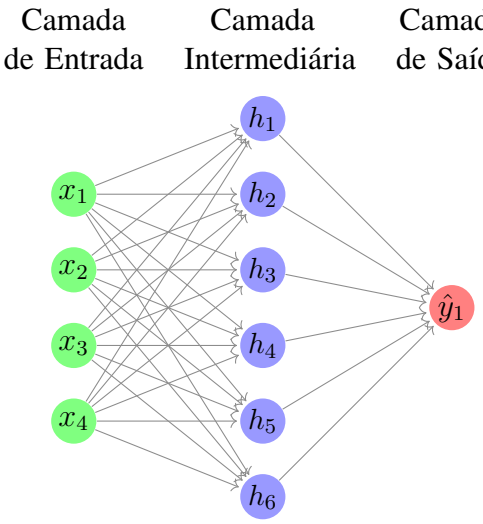
A Figura 2.6 ilustra uma rede MLP. Neste exemplo, a RNA tem 4 entradas. Além disso, esta MLP tem uma camada intermediária com 8 nós e uma saída com um nó. Ressalta-se que cada nó, além dos dados de entrada ou saídas de outros nós ponderadas por pesos, recebe um elemento chamado de *bias* cuja função é transladar a função de ativação. Ademais, percebe-se que a rede possui uma camada oculta com 6 nós cada e uma saída com um nó.

2.2.2 Rede Neural Convolucional

A Rede Neural Convolucional (em inglês, *Convolucional Neural Network* - CNN) aplica filtros em uma matriz para obter informações relevantes da matriz e enviá-las para a camada de entrada da rede. Esta rede que recebe os dados filtrados pode ser, por exemplo, uma MLP.

A CNN é bastante utilizada para trabalhar com imagens. Afinal, as imagens são definidas por matrizes. Nesse sentido, a CNN é capaz de identificar, por meio dos filtros, características relevantes nas imagens que permitem classificá-la.

Figura 2.6: Exemplo de um Perceptron de Múltiplas Camadas.



Fonte: Autoria própria.

A partir das CNN, objetos podem ser identificados em uma imagem independente de sua escala, rotação, translação, entre outras transformações. Por causa dessa característica, esse tipo de rede permitiu um avanço significativo em aplicações de visão computacional. Cita-se como exemplo, a evolução dos carros autônomos graças ao uso da CNN para mapear o trajeto a partir da análise de imagens geradas por suas câmeras.

A estrutura da Rede Neural Convolucional é baseada no córtex visual humano. Nesta região, a imagem visualizada é feita por uma sequência de camadas, onde cada uma delas consegue filtrar um tipo de informação, que pode ser uma face, um objeto, bordas, entre outros atributos.

A primeira camada da CNN é a de convolução. É nesta camada que as informações relevantes são extraídas da imagem. Como entrada, a camada de convolução recebe a matriz da imagem e os filtros de características, conhecidos como kernels.

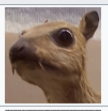
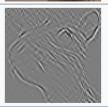
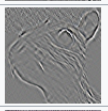
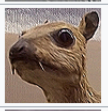
Inicialmente, é feita operações de convolução entre a matriz da imagem e os filtros. Estas operações resultam em matrizes denominadas mapas de características, que permitem a detecção de borda, o desfoque, alteração de nitidez, entre diferentes atributos a depender do tipo de filtro empregado [15]. A Figura 2.7 mostra um comparativo entre diferentes tipos de kernel.

Em algumas situações, o filtro não se ajusta à imagem de entrada. Nesses casos, pode-se preencher a imagem com zeros, denominados zeros de preenchimento, para que não sobre pixel que não "coube"no filtro. Outra opção é descartar a parte da imagem onde o filtro não cobriu.

Após a convolução, os mapas de características passam por uma função de ativação não linear, normalmente a ReLU. Posteriormente, esses mapas são enviados para a camada de *Pooling*. Nesta camada, a dimensionalidade de cada mapa é reduzida a partir da filtragem das informações mais importantes. Existem diferentes métodos de *pooling*. Cita-se como exemplo o *pooling* máximo, que colhe o maior elemento do mapa.

Consecutivamente a camada de *pooling*, as matrizes são transformadas em vetores que servem de entrada para uma rede neural. A Figura 2.8 apresenta o diagrama da arquitetura de uma

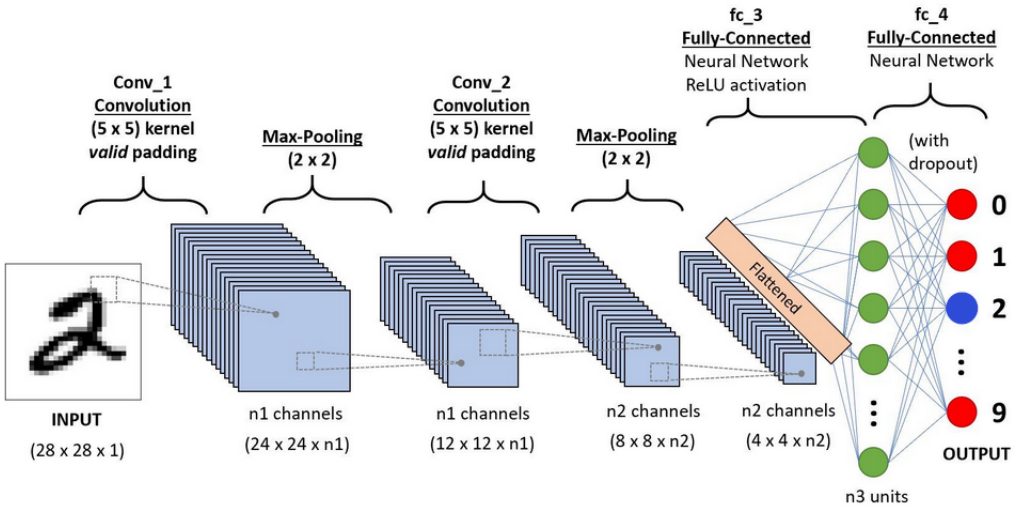
Figura 2.7: Amostras de kernels.

Operation	Kernel ω	Image result g(x,y)
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Ridge or **edge detection**	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur 3 × 3 (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	
Gaussian blur 5 × 5 (approximation)	$\frac{1}{256}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$	
Unsharp masking 5 × 5 Based on Gaussian blur with amount as 1 and threshold as 0 (with no image mask)	$\frac{-1}{256}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & -476 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$	

Fonte: [14].

Rede Neural Convolucional.

Figura 2.8: Arquitetura da Rede Neural Convolucional.



Fonte: [16] modificado pelo autor.

Na Figura 2.8 são usadas duas camadas de convolução e duas de *pooling*, as quais aplicam o método de *pooling* máximo. A etapa nomeada de *Flattened* refere-se a transformação das matrizes vindas do *pooling* em vetores de entrada para a rede neural.

2.2.3 Memória de Curto Prazo Longa

As redes de Memória de Curto Prazo Longa (em inglês, *Long Short Term Memory* - LSTM) é um tipo de Rede Neural Recorrente (RNR). Por isso, antes de definir LSTM, é preciso conhecer de forma geral o conceito de RNR.

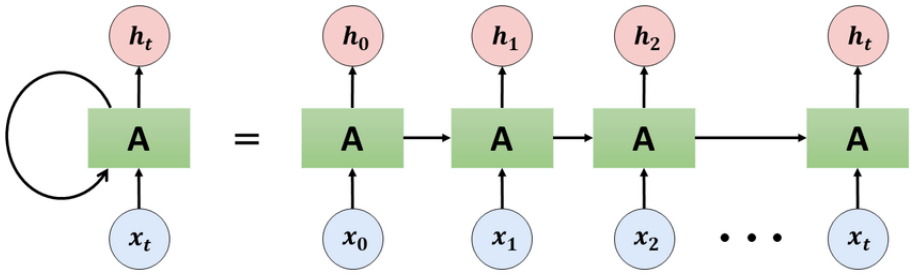
As Redes Neurais Recorrentes são RNAs com memória. Isto significa que para obter um resultado para um conjunto de dados atuais, é necessário consultar informações passadas. Este processo é semelhante a como o ser humano utiliza de experiências passadas em sua memória para processar acontecimentos atuais.

As RNRs são constituídas por loops que fazem com que a rede armazene informações passadas anteriormente por ela. Assim, uma RNR pode ser descrita como um conjunto de cópias da mesma rede em momentos passados e atual. A Figura 2.9 ilustra essa relação.

O principal questionamento que deve ser feito ao projetar uma RNR é quais informações devem ser armazenadas. Isso porque nem todas as informações anteriores são relevantes para determinar o resultado atual. Além disso, quando o número de informações for muito grande, o gradiente que chega nas primeiras camadas é extremamente pequeno. Esse fenômeno é chamado de desaparecimento do gradiente.

A LSTM, propostas inicialmente por Hochreiter e Schmidhuber [18] são redes capazes de reter apenas aquilo que é importante, independente se faz parte de uma memória de longo ou de curto prazo, descartando o que não é útil.

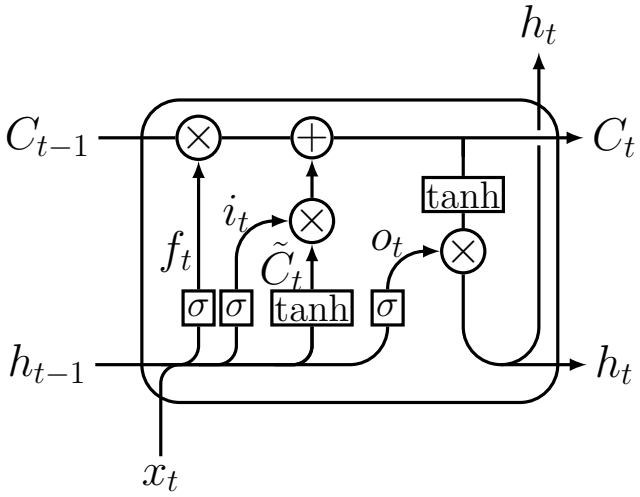
Figura 2.9: Arquitetura da Rede Neural Recorrente.



Fonte: [17].

A rede LSTM possui três estruturas denominadas portões para controlar o esquecimento e armazenamento de dados, conforme mostrado na Figura 2.10. O primeiro portão é o de esquecimento de dados, o segundo de entrada e o último de saída [19]. Nesta figura, os símbolos $\bigotimes$ e $\bigoplus$ indicam a exclusão e adição de informação, respectivamente.

Figura 2.10: Rede Neural Artificial Recorrente LSTM.



Fonte: Autoria própria.

O portão de esquecimento aplica a função sigmoide conforme mostrado na Equação (2.13a) para determinar se a informação deve ou não ser mantida. Caso a saída dessa função seja 0, deve-se excluir por completo a informação, porém, se for 1, deve ser mantida.

O portão de entrada utiliza as Equações (2.13b) e (2.13c) para determinar quais novas informações devem ser armazenadas. Outrossim, por meio da Equação (2.13d) é possível calcular o novo estado da célula - bloco de memória da LSTM - com as informações removidas e adicionadas.

O portão de saída realiza uma filtragem do estado da célula por meio das Equações (2.13e) e (2.13f) para produzir a saída da camada oculta.

$$f_t = \sigma\left(W_f \cdot x_t + U_f \cdot h_{t-1} + b_f\right), \tag{2.13a}$$

$$i_t = \sigma\left(W_i \cdot x_t + U_i \cdot h_{t-1} + b_i\right), \tag{2.13b}$$

$$\tilde{C}_t = \tanh\left(W_c \cdot x_t + U_c \cdot h_{t-1} + b_C\right), \tag{2.13c}$$

$$C_t = f_t \circ C_{t-1} + i_t \circ \tilde{C}_t, \tag{2.13d}$$

$$o_t = \sigma\left(W_o \cdot x_t + U_o \cdot h_{t-1} + b_o\right), \tag{2.13e}$$

$$h_t = o_t \circ \tanh(C_t), \tag{2.13f}$$

onde:

$\sigma -$ Função sigmoide.

$\tanh -$ Função tangente hiperbólica.

$W -$ Vetor de pesos que interliga a camada oculta anterior a atual.

$U -$ Vetor de pesos que conecta a entrada com a camada oculta.

$f_t, i_t, o_t -$ Vetores de ativação das portas de esquecimento, entrada e saída.

$\tilde{C}_t -$ Vetor de ativação da célula de entrada.

$C_t -$ Vetor da célula de memória.

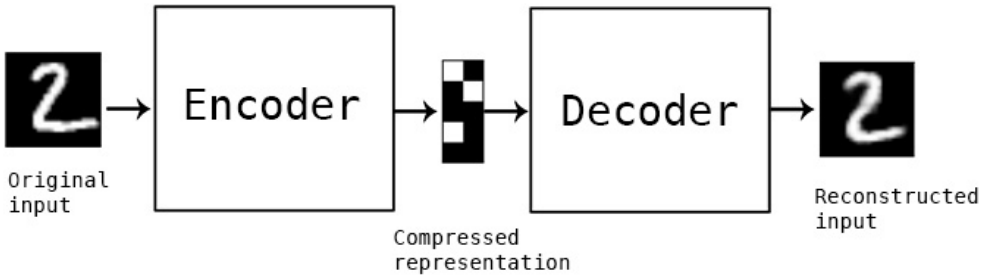
$b -$ Vetor dos bias.

$h_t -$Vetor de estado da camada oculta.

2.2.4 Autoencoders

Os *Autoencoders* (AEs) são redes neurais que comprimem e descomprimem um conjunto de dados. A Figura 2.11 mostra o diagrama de um AE, onde nota-se que esse modelo de rede é composto por duas partes: um codificador e um decodificador. O codificador, primeira estrutura da rede, realiza a compactação dos dados, enquanto o decodificador descompacta a saída do decodificador.

Figura 2.11: Autoencoder.



Fonte: [20].

A ideia do *Autoencoder* é produzir na saída do decodificador uma cópia da entrada no codificador. Assim, como a entrada e a saída devem ser iguais, a entrada é usada para calcular o erro na saída e, desse modo, permitir o aprendizado da rede.

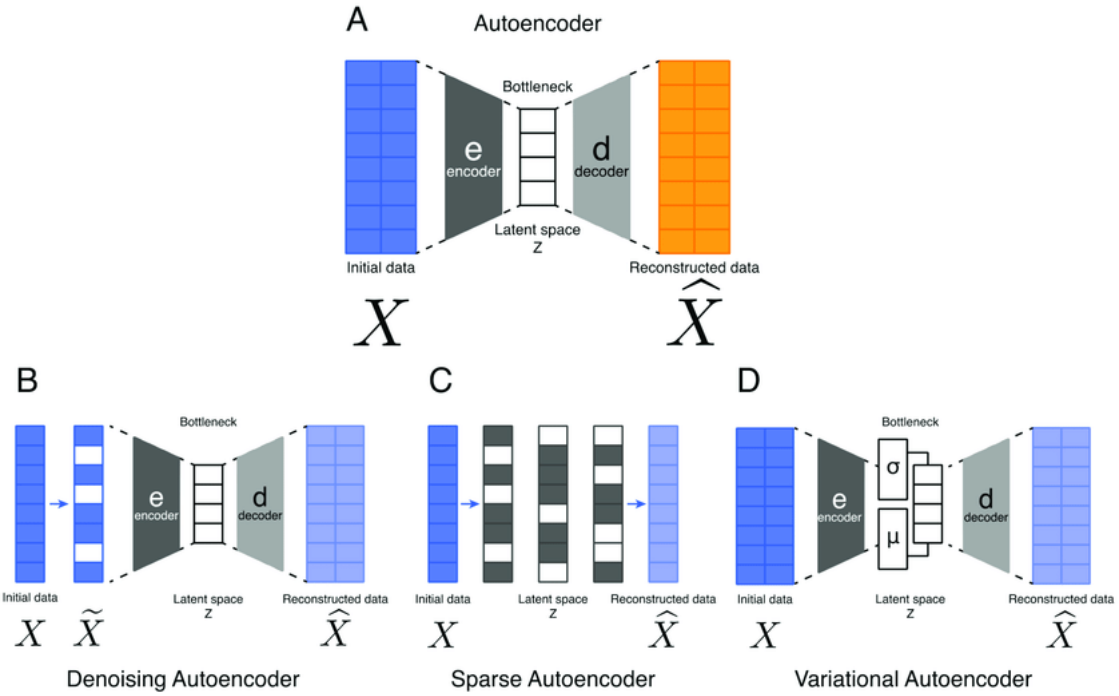
Os *Autoencoders* podem ser usados na remoção de ruídos e redução de dimensionalidade dos dados. Isso porque, os AEs conseguem, por meio de seu codificador, retirar apenas as informações úteis de um conjunto de dados. Ademais, o decodificador serve para poder regenerar os dados para dimensionalidade original quando necessário.

Quando o número de dados de entrada em uma RNA usada para classificação, como uma rede MLP, é muito grande, o tempo de processamento durante o treinamento torna-se muito elevado. Nesse contexto, o codificador de um *Autoencoder* pode ser utilizado para diminuir o tamanho da entrada, permitindo que a MLP classifique os dados mais rapidamente.

A Figura 2.12 apresenta os diferentes tipos de AE. Na representação "A" está o *Autoencoder* clássico, constituído por um codificador que reduz a dimensionalidade no espaço latente, e por um decodificador que recupera os dados. Na imagem "B", tem-se o *Denoising Autoencoder* (DAE), que consiste na adição de ruído branco nos dados de entrada.

Além disso, tem-se na Figura 2.12, em sua representação "C", um AE esparso, que possui apenas uma pequena quantidade de nós ativos por vez. Por fim, em "D" é mostrado o AE variacional, que aprende a distribuição do espaço latente por meio da definição de uma média μ e um desvio padrão σ. Este último tipo de AE é útil quando o espaço latente tem distribuição normal.

Figura 2.12: Diferentes tipos de *Autoencoder*.



Fonte: [21].

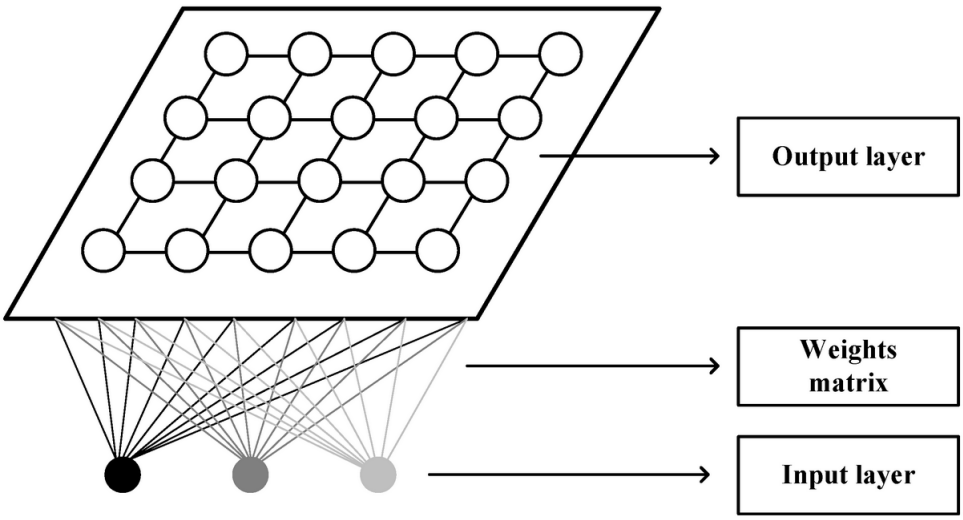
2.2.5 Mapas Auto-Organizáveis

Os Mapas Auto-organizáveis (em inglês, *Self Organizing Map* - SOM) são redes não supervisionadas utilizadas para separar os dados em grupos. Esse agrupamento de dados semelhantes recebe o nome de clusterização, onde os grupos são chamados de *clusters* [22–24].

Esse tipo de rede possui diversas aplicações. Uma plataforma de *streaming* que tenha um catálogo grande de filmes pode usar a SOM para agrupar os seus usuários de acordo com os seus históricos de filmes assistidos. Dessa forma, a plataforma poderá direcionar as indicações de filme para cada grupo de usuário.

Existem diferentes tipos de SOM. No tipo *Kohonen Networks* há uma única camada bidimensional. Os nós desta camada conectam-se a todas as entradas, onde pesos são aplicados em cada uma dessas conexões. A Figura 2.13 mostra a arquitetura *Kohonen Networks*.

Figura 2.13: SOM com arquitetura Kohonen Networks.



Fonte: [25].

O algoritmo utilizado para implementar o SOM é o K-means. Este algoritmo é baseado no cálculo da distância euclidiana entre centroides e os demais pontos para determinar qual cluster (grupo) o ponto faz parte.

A implementação do k-means é feita por meio das seguintes etapas [26].

1. Definir o número de clusters. Em geral, esse valor pode ser estimado pela Equação (2.14).

$$\text{número de clusters} = \sqrt{\frac{N}{2}}, \tag{2.14}$$

   em que N é a quantidade de pontos.

2. Definir um centroide para cada cluster. O primeiro centroide é escolhido aleatoriamente, porém os demais são os pontos mais distantes dele.

3. Calcular a distância euclidiana, dada pela Equação (2.15), e determinar o centroide mais próximo de cada ponto. O cluster que o ponto fará parte é o de seu centroide mais perto.

$$d(x, y) = \sqrt{\sum_{i=1}^{D}(x_i - y_i)^2}, \tag{2.15}$$

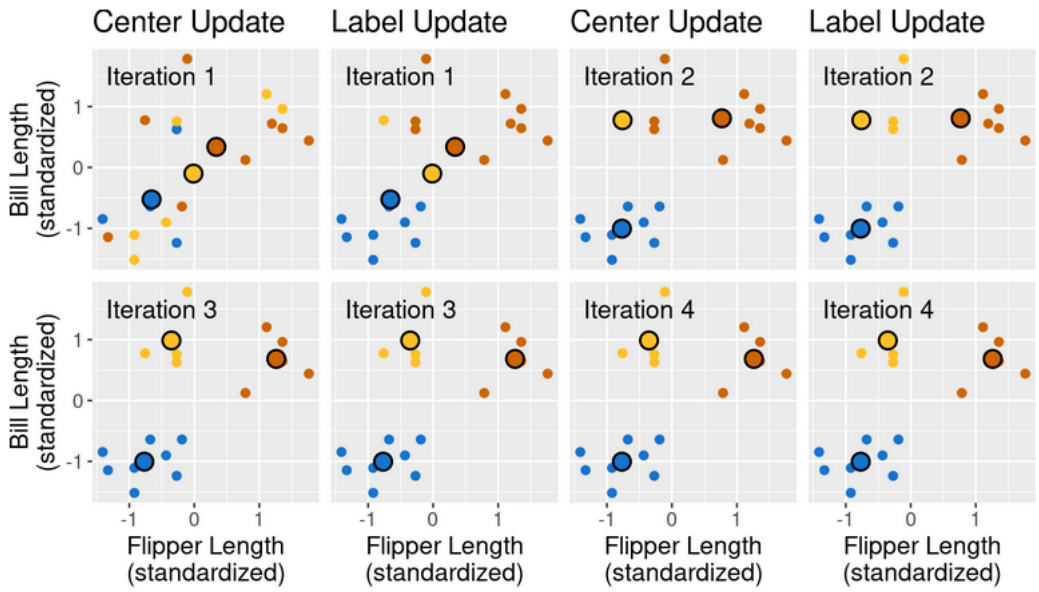
onde D é a dimensão das entradas, x é o centroide e y o ponto que se deseja calcular a distância d até o centroide.

4. Alterar a posição do centroide, sendo esta a média da posição de todos os pontos do cluster.

5. Os passos 3 e 4 são repetidos em cada época até encontrar uma posição em que o cluster dos pontos para de mudar.

A Figura 2.14 contém gráficos da clusterização (agrupamento) de um conjunto de pontos em 3 clusters em 4 iterações. Em "Center Update", ocorre a etapa de atualização da posição do centroide, enquanto em "Label Update", acontece a determinação do cluster que cada ponto faz parte.

Figura 2.14: Exemplo de clusterização usando o algoritmo k-means.



Fonte: [27].

Assim, finaliza-se a apresentação dos modelos de RNAs usadas neste trabalho. Na Seção 2.3, são detalhadas as características dos ataques empregados no trabalho.

2.3 Ataques

2.3.1 Negação de Serviço Distribuída

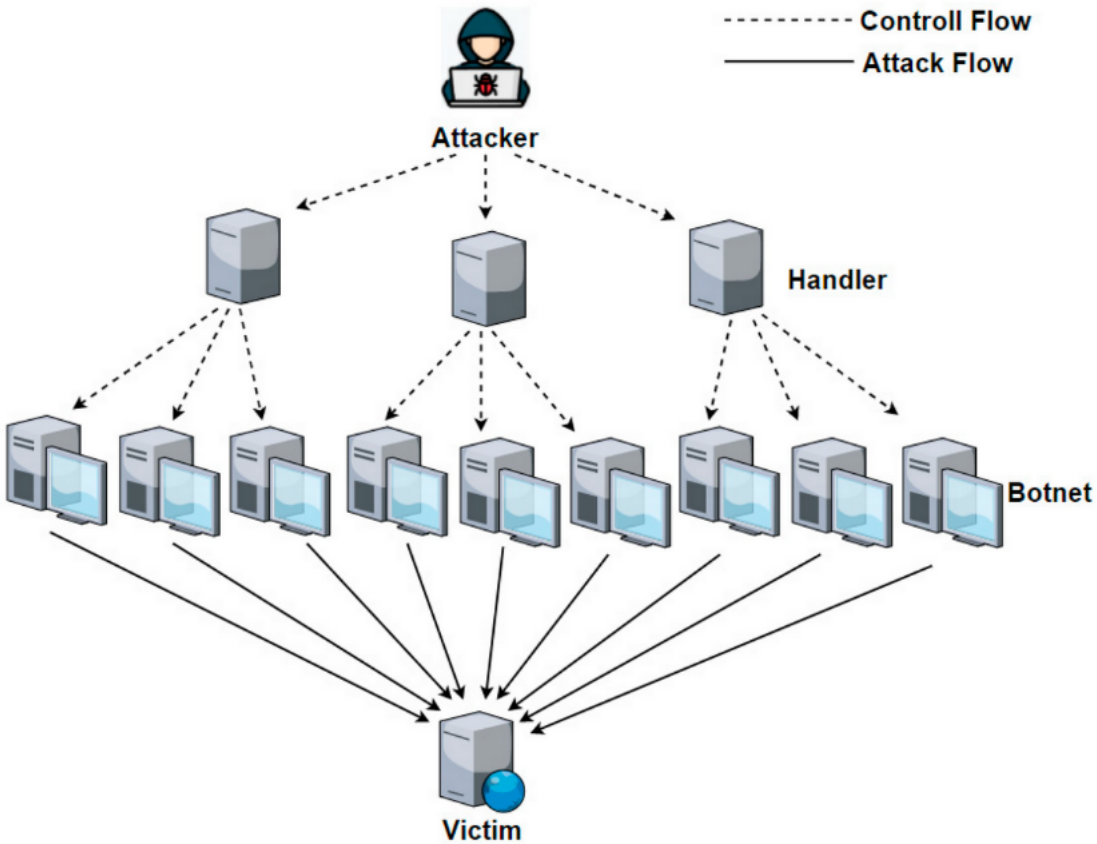
O ataque de Negação de Serviço (em inglês, DoS) visa sobrecarregar um servidor para que este fique indisponível. O ataque de Negação de Serviço Distribuída (em inglês, *Distributed Denial of Service* - DDoS) é um tipo de DoS que utiliza várias fontes para sobrecarregar o servidor.

Para realização de um ataque DDoS é preciso que o invasor tenha controle de diversas outras máquinas previamente infectadas por ele. Com isso, essas máquinas, denominadas zumbis ou bots, enviam pacotes definidos pelo invasor. Uma rede composta por bots é conhecida como botnet [28].

Caso a fonte de um ataque DoS fosse única, além de essa fonte ter dificuldades para produzir uma quantidade suficiente de pacotes para causar indisponibilidade, também haveria a possibilidade de um *firewall* entender que o IP de origem do invasor é suspeito e limitar seu tráfego ou até mesmo bloqueá-lo. No caso do DDoS, por haver várias fontes de pacotes, aumenta-se o número de pacotes simultâneos, facilitando a sobrecarga, e a grande quantidade de IPs de origem torna a tarefa do *firewall* mais complexa.

A Figura 2.15 mostra como ocorre um ataque de Negação de Serviço Distribuída.

Figura 2.15: Negação de Serviço Distribuída.



Fonte: [3].

Os ataques de DoS ocorrem geralmente nas Camadas 7, 4 e 2 do modelo OSI, conforme detalhado a seguir:

- Camada 7

  Vulnerabilidades em aplicações são exploradas para tornar o serviço indisponível. Como exemplo, cita-se a vulnerabilidade que servidores Apache de versões inferiores à 2.2.22 têm de permitir que vários sockets abertos via estado Keep-alive do protocolo HTTP não sejam finalizados, o que sobrecarrega o servidor.

- Camada 4

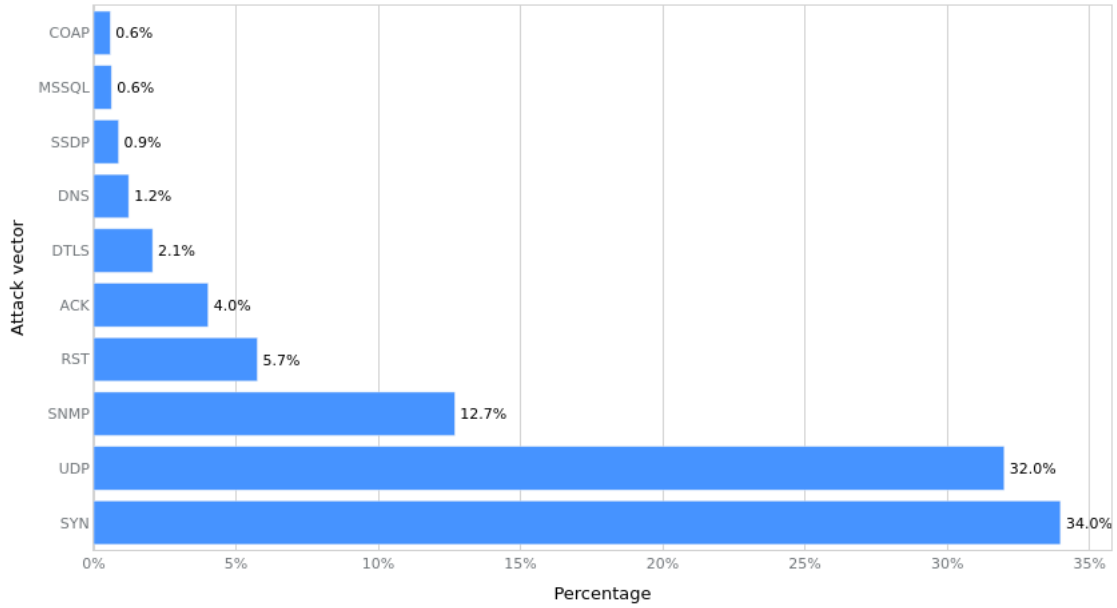
  Nesse caso, é enviada uma grande quantidade de pacotes com a intenção de sobrecarregar o servidor. Este fenômeno é conhecido como inundação de pacotes. O pacote TCP, por exemplo, pode ser usado por meio de uma das suas flags: SYN, ACK, FIN e RST. Pode-se usar também, ICMP, UDP, entre outros protocolos, como o RIP e OSPF. O tipo de protocolo a ser usado pelo invasor é determinado pela análise que este faz da segurança da rede.

- Camada 2

  Os ataques são feitos sobre dispositivos da camada de enlace, como o switch. Um exemplo de ataque à esse tipo de dispositivo é o MAC Flooding, que consiste na inundação do switch de pacotes com diferentes endereços de MAC. Desta maneira, o switch fica sobrecarregado e faz broadcast de cada pacote que entra, não enviando somente para porta correta.

Na Figura 2.16, observa-se o gráfico de uso dos principais vetores de ataques DDoS na Camada 4 no ano de 2021 de acordo com a Cloudflare Radar. Nota-se que o protocolo TCP com a flag SYN ativa foi o que aconteceu o maior número de ocorrências.

Figura 2.16: Uso dos vetores de ataque de DDoS na Layer 4 no ano de 2021.



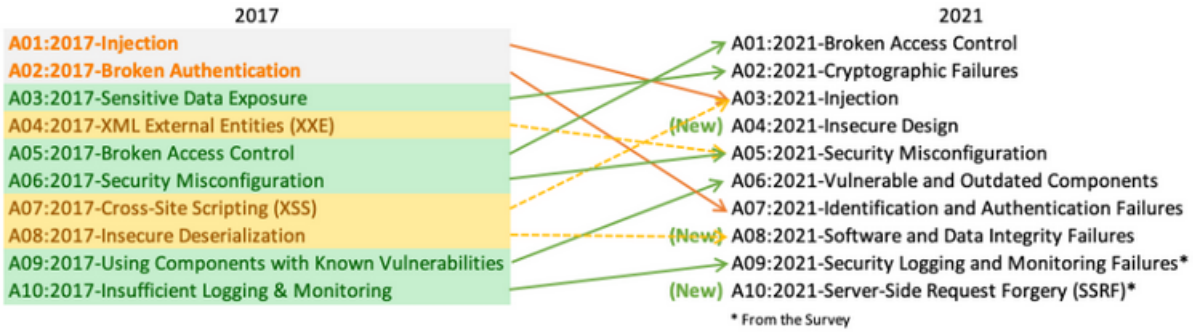
Fonte: [29].

2.3.2 Ataques Web

Quando se fala de ataques web, é importante ressaltar o papel da OWASP (*Open Web Application Security Project*), que é uma fundação sem fins lucrativos que produz documentos, treinamentos, entre outros recursos com o intuito de melhorar a segurança dos *softwares*.

Entre os projetos desenvolvidos pelo OWASP, destaca-se o OWASP Top 10, que é uma lista dos principais riscos de segurança em aplicações web. A Figura 2.17 faz uma comparação entre a lista de 2017 e a de 2021, que é a mais atual na data de escrita desse trabalho.

Figura 2.17: OWASP Top 10 em 2017 e 2021.



Fonte: [31]

Neste trabalho, serão utilizados ataques Injection, 3° na lista da OWASP em 2021. Os ataques de Injection serão XSS e SQL Injection. Sublinha-se que o XSS foi incluído na categoria de Injection a partir de 2021.

Outrossim, também será usado ataques de *Identification and Authentication Failures*, que caiu da 2° para a 7° posição de 2017 para 2021. O tipo de ataque dessa categoria será o de força bruta, cujo os detalhes são mostrados detalhadamente na próxima seção.

1) XSS (Cross-Site Scripting)

Este tipo de ataque consiste na injeção de scripts em JavaScript maliciosos em aplicações web [30]. Em geral, essa inclusão de código ocorre por falhas na filtragem de dados de entrada por parte do *software*. O XSS possibilita que o atacante roube cookies de sessão e dados de acesso, redirecione o usuário para outra página, permitindo a captura dos valores digitados, entre diversas outras ações.

Os ataques XSS podem ser divididos em três tipos: refletido, armazenado e DOM.

- XSS refletido − o código é executado no lado do cliente como uma resposta a uma solicitação. Nesse tipo de ataque, a entrada com script não é armazenada permanentemente, é somente retornada (refletida) após ser enviada. Pode-se, por exemplo, obter o cookie de uma sessão com esse ataque injetando o seguinte script:

  <script>alert(document.cookie)</script>

- XSS armazenado − a entrada do usuário é armazenada no servidor. Assim, o ataque não é direcionado a um usuário, e, sim, a todos usuários que acessam o servidor que tem o script malicioso armazenado. Uma forma de realizar esse tipo de ataque é digitar o seguinte script em um campo de comentário que é mostrado a todos usuários de uma página web:

  <script src = "http://IP.com/script-exploit.js" </script>

  onde o IP é o endereço de uma página criada pelo atacante com um arquivo de exploit, que no caso do exemplo, recebeu o nome de script-exploit.

- XSS baseado em DOM − a requisição é executada no browser do usuário, que opera o DOM (Document Object Model). Nesse caso, o código malicioso nunca é enviado ao servidor, dificultando que este identifique um ataque.

2) SQL Injection

O SQL Injection é caracterizado pela injeção de códigos SQL em uma consulta ao banco de dados de uma aplicação web. Por meio desse tipo de ataque, o atacante pode consultar no banco de dados informações como login e senha dos usuários, dados pessoais de usuários, entre diversos outros dados que o invasor conseguir acessar.

A Figura 2.18 exemplifica o funcionamento do SQL Injection. Neste exemplo, o ataque ocorre da seguinte forma:

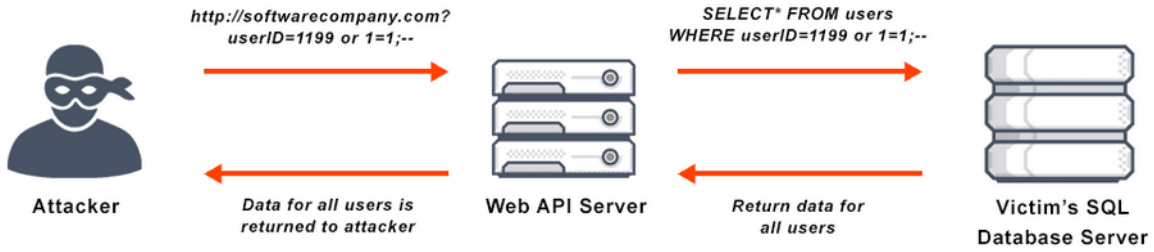
- O atacante faz uma consulta SQL pelo URL da página.

- O servidor de banco de dados recebe esta solicitação por meio da API do Servidor Web.

- O servidor de banco de dados retorna uma lista com dados de todos os usuários, pois a relação "1=1"é, obviamente, verdadeira para todos os usuários.
- Os dados dos usuários são recebidos pelo atacante enviados pelo servidor Web.

O uso de dois hifens (–) foi feito para que todo código que vier depois seja tratado como comentário, não interferindo na consulta do atacante. Uma alternativa aos dois hifens – seria usar cerquilha (#).

Figura 2.18: Exemplo de ataque de SQL Injection.



Fonte: [32].

2.3.3 Ataque de Força Bruta

O ataque de força bruta tenta adivinhar senhas, dados de usuários, URLs escondidas, entre outras informações. Em outras palavras, este tipo de ataque consiste na realização de tentativas sucessivas de descobrir uma informação.

Esse tipo de ataque é um dos mais antigos e simples de ser realizado. Porém, a força bruta ainda é um ataque bastante utilizado e que pode ser muito efetivo quando usuários utilizam senhas fracas.

Quando o invasor tem conhecimento de informações pessoais da vítima, como data de nascimento, nome dos filhos, nome do animal de estimação, filme preferido, entre outras, ele consegue criar uma lista de possíveis senhas com mais chance de ser assertiva. Com as redes sociais, informações como as citadas são facilmente obtidas.

Uma das ferramentas mais utilizadas para realizar o ataque de força bruta é a Hydra, que pode ser usada em sistemas operacionais Windows e distribuições Linux, além de suportar vários protocolos. Um exemplo de uso do Hydra em um terminal de uma distribuição Linux é mostrado abaixo.

hydra -l root -P list-password.txt X.X.X.X -t 6 ssh

onde o -l indica o usuário, -P o arquivo com as senhas a serem testadas, X.X.X.X o endereço de IP do alvo e -t o número de threads empregados durante o ataque, e o ssh indica que este é o serviço que sofrerá força bruta.

Existem diversas medidas que podem ser tomadas para evitar o ataque de força bruta. Entre essas medidas, cita-se a limitação por tempo da realização de tentativas de login, o uso de senhas

com uma grande quantidade de bits e que não contenha informações pessoais, e utilização de Duplo Fator de Autenticação (2FA).

A Figura 2.19 mostra a estimativa da quantidade de tempo gasto por um invasor de acordo com as características da senha. As informações para construir essa tabela foram obtidas em [29]

Figura 2.19: Tempo gasto para descobrir uma senha.

Nº DE CARACTERES	SOMENTE NÚMEROS	LETRAS MINÚSCULAS	LETRAS MINÚSCULAS E MAIÚSCULAS	MINÚSCULAS + MAIÚSCULAS + NÚMEROS	MINÚSCULAS + MAIÚSCULAS + NÚMEROS + SÍMBOLOS
4	IMEDIATAMENTE	IMEDIATAMENTE	IMEDIATAMENTE	IMEDIATAMENTE	IMEDIATAMENTE
5	IMEDIATAMENTE	IMEDIATAMENTE	IMEDIATAMENTE	IMEDIATAMENTE	IMEDIATAMENTE
6	IMEDIATAMENTE	IMEDIATAMENTE	IMEDIATAMENTE	1 SEGUNDO	5 SEGUNDOS
7	IMEDIATAMENTE	IMEDIATAMENTE	25 SEGUNDOS	1 MINUTO	6 MINUTOS
8	IMEDIATAMENTE	5 SEGUNDOS	22 MINUTOS	1 HORA	8 HORAS
9	IMEDIATAMENTE	2 MINUTOS	19 HORAS	3 DIAS	3 SEMANAS
10	IMEDIATAMENTE	58 MINUTOS	1 MÊS	7 MESES	5 ANOS
11	2 SEGUNDOS	1 DIA	5 ANOS	41 ANOS	400 ANOS
12	25 SEGUNDOS	3 SEMANAS	300 ANOS	2K ANOS	34K ANOS
13	4 MINUTOS	1 ANO	16K ANOS	100K ANOS	2M DE ANOS
14	41 MINUTOS	51 ANOS	800K ANOS	9M DE ANOS	200M DE ANOS
15	6 HORAS	1K ANOS	43M DE ANOS	600M DE ANOS	15 BI DE ANOS
16	2 DIAS	34K ANOS	2 BI DE ANOS	37 BI DE ANOS	1 TRI DE ANOS
17	4 SEMANAS	800K ANOS	100 BI DE ANOS	2 TRI DE ANOS	93 TRI DE ANOS
18	9 MESES	23M DE ANOS	100 TRI DE ANOS	100 TRI DE ANOS	7.10^{48} DE ANOS

Fonte: [33].

2.3.4 Varredura de Portas

A varredura de portas (em inglês, *Port Scanning*) é usada para determinar quais portas estão abertas, de modo a identificar os serviços disponíveis. Além disso, a varredura pode detectar a presença de firewalls.

A varredura de porta pode ser feita por meio do TCP ou UDP conforme detalhado a seguir.

- Varredura TCP – primeiramente é enviado um pacote com a flag SYN. As possíveis respostas serão:

  i) SYN+ACK – a porta está aberta.

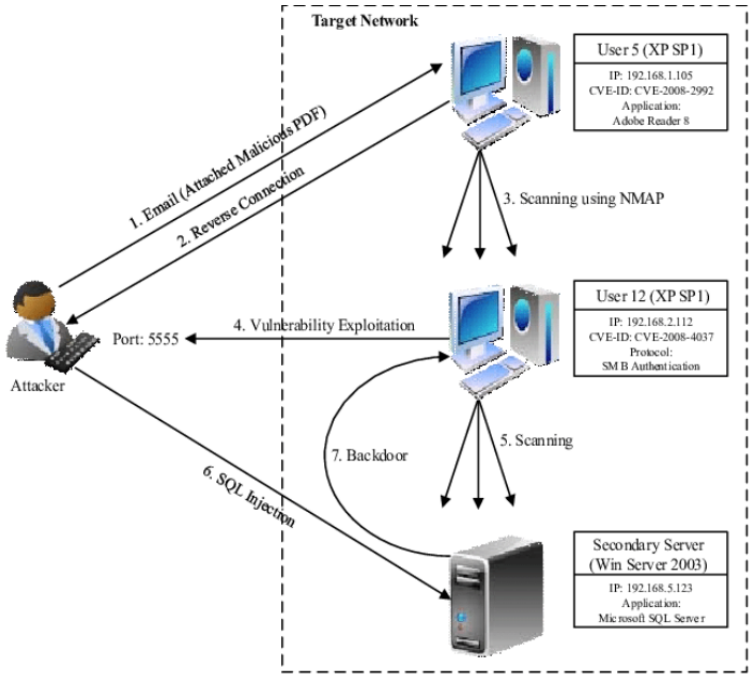
  ii) RST+ACK – a porta está fechada.

  iii) ICMP Port Unreachable – porta está filtrada por algum firewall.

- Varredura UDP – é enviado um pacote UDP. As possíveis respostas serão:

  i) ICMP Port Unreachable – porta pode estar fechada ou está filtrada por algum firewall.

Figura 2.20: Exemplo de Infiltração.



Fonte: [35].

ii) Nenhuma resposta − porta está aberta ou o pacote foi recusado por um firewall que não envia mensagem informando esse descarte.

Um exemplo de ferramenta que realiza varredura de porta é o Nmap [34] , que é um *software* de código aberto conhecido pela grande variedade de recursos que oferece. A seguir é mostrado um comando em Nmap

nmap –top-ports 10 1.1.1.0/24

onde –top-ports 10 indica que serão varridas as 10 portas mais comuns para o conjunto de IPs 1.1.1.0/24.

2.3.5 Ataque de Infiltração

A infiltração ocorre quando o invasor consegue penetrar a rede, em geral, por meio da exploração de alguma vulnerabilidade em *software*. Posteriormente a exploração, é aberta uma porta de acesso pelo invasor para administrar o computador da vítima, o que é chamado de *backdoor*.

A Figura 2.20 mostra um exemplo de infiltração, onde o arquivo malicioso é enviado por e-mail, a vulnerabilidade é explorada e é estabelecida uma *backdoor*.

Destarte, com este capítulo, pode-se conhecer as características dos ataques e modelos de RNAs usadas neste trabalho. No Capítulo 3, será possível entender como foram montadas e treinadas as RNAs, bem como o banco de dados usado.

Capítulo 3

Metodologia

3.1 Banco de Dados

O banco de dados utilizado para treinar e testar as RNAs desenvolvidas neste trabalho é o CIC-IDS2017 (*Canadian Institute for Cybersecurity - Intrusion Detection Systems*) [36]. Este banco de dados contém informações de tráfego normal e de ataques.

A captura de tráfego é obtida por meio do CICFlowMeter, que disponibiliza uma série de recursos para a análise de tráfego. Ao todo, são fornecidas 84 informações de tráfego de 25 usuários com base nos protocolos HTTP, HTTPS, FTP, SSH e de e-mail. Esses dados são ofertados em formato CSV ou em PCAP.

A coleta de dados pelo CIC ocorreu em 5 dias, começando às 9 horas da segunda-feira, 3 de julho de 2017, e terminando às 17 horas da sexta-feira, 7 de julho de 2017. Os ataques ocorreram na manhã e na tarde de terça, quarta, quinta e sexta-feira, sendo assim, na segunda-feira foram realizados apenas tráfegos normais.

A Figura 3.1 mostra a arquitetura de rede utilizada na geração dos tráfego. Nota-se que há duas redes separadas e distintas: a rede ataque e a rede vítima. A rede de ataque possui um roteador, um switch e quatro computadores cujos sistemas operacionais são o Kali Linux e o Windows 8.1. Enquanto isso, a rede vítima contém dez computadores interconectados por um controlador de domínio (DC) e um diretório ativo (AD) e três servidores, sendo dois servidores de web e um de captura.

As Tabelas 3.1 e 3.2 apresentam os sistemas operacionais e os IPs dos dispositivos usados, respectivamente, na rede vítima e na rede ataque.

O tráfego que ocorreu em cada dia é detalhado a seguir.

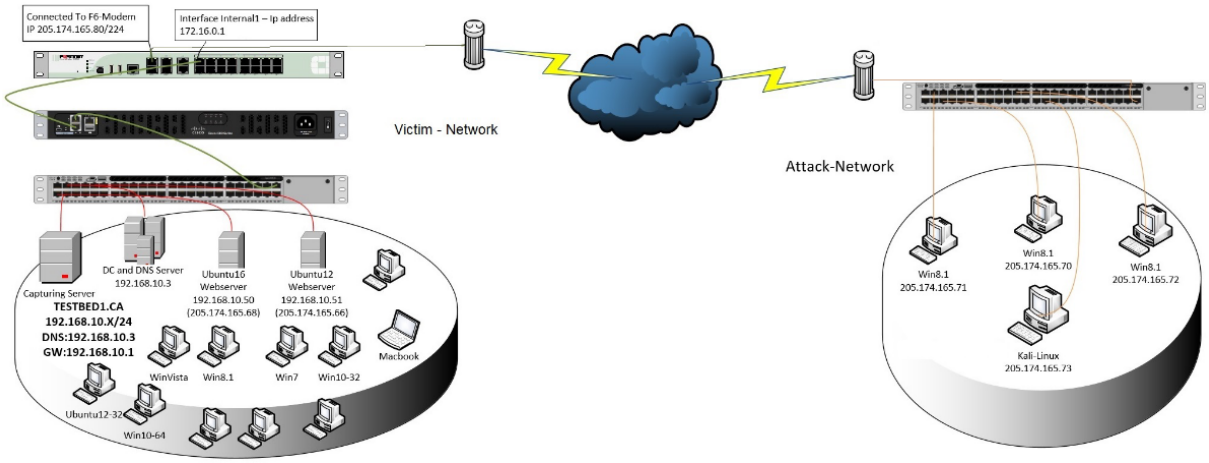
1) Segunda-feira, 3 de julho de 2017

- **Manhã e tarde − apenas tráfego normal**.

2) Terça-feira, 4 de julho de 2017

- **Manhã e tarde − Ataque de força bruta**.

Figura 3.1: Arquitetura de Rede Utilizada para Gerar o Tráfego.



Fonte: [36].

Para este ataque, utilizou-se a ferramenta Patator [37], que é escrita em Python e com diversos recursos para realização de ataque de força bruta. Foram feitos ataques em:

i) FTP (9h:20min - 10h20min)

ii) SSH (14h:00min - 15h00min)

3) Quarta-feira, 5 de julho de 2017

- **Manhã – DoS**

  Foram feitos ataques DoS com as seguintes ferramentas:

  i) Slowloris (9h47 – 10h10)

  ii) Slowhttptest (10h14 – 10h35)

  iii) Hulk (10h43 – 11h)

  iv) GoldenEye (11h10 – 11h23)

- **Tarde – Ataques Heartbleed**

  Ocorreu das 15h:12min até 15h:32min utilizando a ferramenta Heartleech e é o único tipo de ataque que não foi utilizado para desenvolver as redes neurais deste trabalho.

4) Quinta-feira, 6 de julho de 2017

- **Manhã – Ataques Web**

  Realizou os seguintes ataques:

  i) Força Bruta (9:20 – 10:00)

  ii) XSS (10h15 – 10h35)

  iii) SQL Injection (10h40 – 10h42)

Tabela 3.1: Sistemas Operacionais e os IPs das Máquinas da rede vítima.

Máquina	Sistema Operacional	Endereços de IP
Servidores	Windows Server 2016 (DC e DNS)	192.168.10.3
	Ubuntu 16 (Servidor Web)	192.168.10.50 - 205.174.165.68
	Ubuntu 12 (Servidor Web)	192.168.10.51 - 205.174.165.66
PCs	Ubuntu 14.4 (32 e 64 bits)	192.168.10.19 - 192.168.10.17
	Ubuntu 16.4 (32 e 64 bits)	192.168.10.16 - 192.168.10.12
	Windows 7 Pro	192.168.10.9
	Windows 8.1 (64 bits)	192.168.10.5
	Windows Vista	192.168.10.8
	Windows 10 Pro (32 e 64 bits)	192.168.10.14 - 192.168.10.15
	Mac	192.168.10.25
Firewall	Fortinet	

Tabela 3.2: Sistemas Operacionais e os IPs das Máquinas da rede ataque.

Máquina	Sistema Operacional	Endereços de IP
PCs	Kali Linux	205.174.165.73
	Windows 8.1	205.174.165.69
	Windows 8.1	205.174.165.70
	Windows 8.1	205.174.165.71

- **Tarde – Ataques de Infiltração**

  O Arquivo malicioso foi feito por Dropbox, no caso do Windows Vista e por USB para o MacBook. Posteriormente, o Atacante executa o Nmap e faz varredura de porta.

5) Sexta-feira, 7 de julho de 2017

- **Manhã – Ataque Botnet**

  O Botnet foi feito usando a ferramenta Ares, que é baseada no Python. O Ataque ocorreu das 10h:02min às 11h:02min.

- **Tarde – Ataques DDoS e varredura de portas**

  O ataque DDoS é feito por meio da ferramenta LOIC (*Low Orbit Ion Canon*). Ademais, a varredura de porta foi feita empregando a ferramenta Nmap.

Para o desenvolvimento das RNAs, foram usadas 20 informações de tráfego das 84 disponibilizadas no banco de dados. Essas informações são: porta de destino, protocolo, total

de pacotes encaminhados, total de pacotes recebidos, tamanho total de pacotes encaminhados, tamanho total de pacotes recebidos, tamanho máximo dos pacotes encaminhados, tamanho mínimo dos pacotes encaminhados, tamanho médio dos pacotes encaminhados, desvio padrão do tamanho dos pacotes encaminhados, tamanho máximo dos pacotes recebidos, tamanho mínimo dos pacotes recebidos, tamanho médio dos pacotes recebidos, desvio padrão do tamanho dos pacotes recebidos, pacotes por segundo encaminhados, pacotes por segundo recebidos, tamanho mínimo do pacote, tamanho máximo do pacote, tamanho médio do pacote e desvio padrão do tamanho do pacote. Essas informações foram escolhidas por permitirem uma análise dos serviços e protocolos usados no ataques, além de informações estatísticas, como média e desvio padrão, do tamanho e do tempo de envio e recebimento dos ataques.

A maioria dos ataques têm uma quantidade muito maior de tráfego normal do que de ataque. Logo, se uma RNA não identificar nenhum ataque, classificando tudo como tráfego normal, sua acurácia ainda seria alta. Pensando nisso, optou-se por utilizar uma quantidade de tráfego normal próxima da quantidade de ataque. Essa divisão é vista na Tabela 3.3.

Tabela 3.3: Quantidade de registros de tráfego normais e de ataques.

	Força Bruta	**DoS**	**Web**	**Infiltração**	**Botnet**	**Varredura de Portas**	**DDoS**
Normal	17.283	286.020	2.186	289	1.891	127.537	97.718
Ataque	13.835	252.661	2.180	36	1.966	158.930	128.027

Pela Tabela 3.3, identifica-se que no ataque de infiltração, não foi possível ter uma quantidade próxima entre tráfego normal e de ataque. Isso ocorreu porque a quantidade de pacotes maliciosos é muito pequena, assim, mesmo retirando 99,9% do tráfego normal, este ainda ficou 88,923% maior que o tráfego de ataque.

Selecionou-se 70% dos dados para treinamento e 30% para teste. Esta escolha foi feita de forma aleatória para todos os modelos, com exceção do LSTM, que tem as informações de tráfego definidas sequencialmente por utilizar relações temporais. A Tabela 3.4 contém a quantidade de registros de tráfego usada para teste e treinamento para cada ataque.

Tabela 3.4: Quantidade de registros de tráfego para treinamento e teste.

	Força Bruta	**DoS**	**Web**	**Infiltração**	**Botnet**	**Varredura de Portas**	**DDoS**
Treinamento	21782	377.076	3.056	227	2.699	200.526	158.021
Teste	9336	161605	1.310	98	1.158	85.941	67.724

3.2 Arquitetura das Redes Neurais

Com exceção da rede SOM, todas as outras redes utilizam o Adam como otimizador e a entropia cruzada binária é a função de custo. A estrutura de cada modelo de rede é mostrado a seguir. Este modelos foram construídos de forma empírica, testando variadas combinações de parâmetros.

3.2.1 Perceptron de Múltiplas Camadas

A rede possui 20 entradas, 3 camadas ocultas e apenas uma saída, conforme descrito a seguir.

- Camada de entrada:
  - Função de ativação = ReLU
  - Quantidade de nós = 9
- Primeira camada oculta:
  - Função de ativação = ReLU
  - Quantidade de nós = 15
- Segunda camada oculta:
  - Função de ativação = ReLU
  - Quantidade de nós = 9
- Terceira camada oculta:
  - Função de ativação = Softplus
  - Quantidade de nós = 9
- Camada de saída:
  - Função de ativação = Sigmoide
  - Quantidade de nós = 1

3.2.2 Autoencoder

A rede possui 20 entradas no codificador e 5 nós no espaço latente. Ou seja, a dimensão é reduzida de 20 para 5. O espaço latente do autoencoder é usado como entrada em uma RNA MLP usada para classificação com baixas quantidades de nós comparada a MLP da seção 3.2.1.

Autoencoder:

- Camada de entrada (Codificador):
  - Função de ativação = ReLU
  - Quantidade de nós = 5
- Camada de saída (Decodificador):
  - Função de ativação = Sigmoide
  - Quantidade de nós = 20

MLP com 5 entradas, sendo estas o espaço latente do Autoencoder:

- Camada de entrada:
  - Função de ativação = ReLU
  - Quantidade de nós = 2
- Primeira camada oculta:
  - Função de ativação = ReLU
  - Quantidade de nós = 4
- Segunda camada oculta:
  - Função de ativação = ReLU
  - Quantidade de nós = 2
- Terceira camada oculta:
  - Função de ativação = Softplus
  - Quantidade de nós = 2
- Camada de saída:
  - Função de ativação = Sigmoide
  - Quantidade de nós = 1

3.2.3 Rede Neural Convolucional

O vetor de entrada de 20 valores foi transformado em uma matriz 4×5. Além disso, foram empregados 32 filtros de dimensão 2×2. A dimensão do pool é 2×2. A rede possui 20 entradas, utiliza o Adam como otimizador e a entropia cruzada binária é a função de custo.

Rede usada para classificação (recebe os dados da camada de flattened).

- Camada de entrada:
  - Função de ativação = ReLU
  - Quantidade de nós = 15

- Primeira camada oculta:
  - Função de ativação = ReLU
  - Quantidade de nós = 9
- Segunda camada oculta:
  - Função de ativação = Softplus
  - Quantidade de nós = 9
- Camada de saída:
  - Função de ativação = Sigmoide
  - Quantidade de nós = 1

3.2.4 Rede LSTM

A rede possui em suas duas primeira camadas células de memória LSTM, conforme detalhada a seguir.

- Camada de entrada:
  - Quantidade de células LSTM = 9
- Primeira camada oculta:
  - Quantidade de células LSTM = 15
- Segunda camada oculta:
  - Função de ativação = ReLU
  - Quantidade de nós = 9
- Terceira camada oculta:
  - Função de ativação = Softplus
  - Quantidade de nós = 9
- Camada de saída:
  - Função de ativação = Sigmoide
  - Quantidade de nós = 1

3.2.5 Rede SOM

A função de vizinhança é a triangular. Outrossim, para calcular a dimensão dos mapas, calcula-se o coeficiente D, dado por:

$$D = \sqrt{5} \cdot \sqrt{N}^{1/4} \tag{3.1}$$

onde N é o tamanho da base de treino. Dessa forma, a dimensão é $d \times d$, em que d é o menor inteiro de D. Por exemplo, se o tamanho da base de treino é de 21782, tem-se:

$$D = \sqrt{5} \cdot \sqrt{21782}^{1/4} = 27,165 \tag{3.2}$$

logo, a dimensão é 27$\times$27.

3.3 Implementação das RNAs

Os algoritmos das RNAs desenvolvidas neste trabalho foram escritos na linguagem Python 3.10.6 por meio das bibliotecas TensorFlow 2.11.0 e scikit-learn 1.1.3. As RNAs foram executadas no sistema operacional Linux Mint 21.1 em um computador de 8 GB de memória RAM e processador Intel Core i3-6006U. Os resultados desta implementação são mostrados e discutidos no Capítulo 4. Ademais, no Apêndice A, é fornecido um link com códigos em Python para utilizar as RNAs implementadas, além de conter os pesos e as estruturas destas, e os dados de treinamento usados para treinar e testar as RNAs.

Capítulo 4

Resultados

A rede SOM foi treinada com 1000 iterações, enquanto para as RNAs LSTM e Autoencoder foi estabelecido 10 épocas. Para as demais Redes Neurais, especificou-se um limite de 50 épocas, porém, estabeleceu-se que quando o valor da função de custo não evoluir em pelo menos 10^{-4} em 10 épocas seguidas, o treinamento é interrompido. A Tabela 4.1, mostra o número de épocas de cada RNA.

Tabela 4.1: Número de épocas de cada modelo de RNA de acordo com o tipo de ataque.

	Força Bruta	**DoS**	**Web**	**Infiltração**	**Botnet**	**Varredura de Portas**	**DDoS**
MLP	36	50	36	17	49	49	34
CNN	50	50	50	50	50	50	50
LSTM	10	10	10	10	10	10	10
AE	10	10	10	10	10	10	10
MLP pós AE	50	50	50	50	50	50	50
SOM	1000	1000	1000	1000	1000	1000	1000

A Tabela 4.2 apresenta a acurácia doe diferentes modelos de RNA para cada tipo de ataque. A acurácia é calculada dividindo a quantidade de detecções incorretas (falsos positivos mais falsos negativos) pelo número total de dados (falsos positivos e negativos mais verdadeiros positivos e negativos). Nota-se, que para todas as RNAs, o ataque DDoS apresentou a maior média aritmética de acurácia, sendo esta de 99,433%. Isso ocorre porque estas intrusões tem características de tráfego que facilitam a sua detecção, como um alto número de encaminhamento de pacotes de solicitação de conexão (SYN) em um curto espaço de tempo, sem que a conexão seja estabelecida, ou ainda pelo fato de ter endereços de IP que realizam a mesma consulta repetidamente a um mesmo conjunto de dados, o que pode acontecer, por exemplo, em uma inundação de DNS, entre diversas outras particularidades do ataque DDoS.

O ataque Web apresentou uma acurácia média de 78,901%, sendo a menor de todos os

ataques. A possível causa desse resultado é as estatísticas de tráfego usadas conterem apenas informações obtidas pela análise de cabeçalhos dos pacotes, porém, os ataques Webs costumam ser melhor evidenciados pelo conteúdo em seus *payloads*.

Outro motivo que explica a baixa acurácia dos ataques Webs é a pouca semelhança entre os 3 tipos de ofensivas usadas: injeção de SQL, XSS e força bruta. Isso é evidenciado pelo péssimo resultado utilizando a rede SOM, que agrupa os dados por suas similaridades. Caso esse modelo de rede não for usado no cálculo da acurácia média, esta aumentaria para 87,271%.

A rede LSTM teve a melhor acurácia em 3 ataques: Web, varredura de portas e DDoS. Porém, este modelo de RNA foi a rede com pior acurácia nos ataques de força bruta e botnet. Como a rede LSTM é uma rede com memória, pode-se concluir que isso ocorreu devido a baixa relação temporal entre os pacotes dessas intrusões.

A redes MLP e CNN têm as melhores precisões em 2 tipos de ataque. No caso da MLP, ela é superior nos ataques DoS e Botnet, enquanto a CNN é melhor em força bruta e infiltração. Acentua-se que a CNN pode ser definida como uma MLP com filtragem de informações, o que foi bastante efetivo em ataques de força bruta.

Tabela 4.2: Acurácia de cada modelo de RNA de acordo com o tipo de ataque.

	Força Bruta	**DoS**	**Web**	**Infiltração**	**Botnet**	**Varredura de Portas**	**DDoS**
MLP	96,540%	96,568%	94,198%	96,939%	94,387%	97,285%	99,792%
CNN	99,679%	95,225%	85,038%	97,959%	81,779%	99,426%	99,877%
LSTM	62,886%	94,000%	94,504%	94,898%	63,817%	97,779%	99,793%
MLP pós AE	87,532%	94,940%	75,343%	86,735%	90,587%	93,510%	98,872%
SOM	91,613%	86,322%	45,420%	96,939%	88,946%	95,538%	98,831%

Além da acurácia, outro critério importante na escolha do modelo de RNA é o tempo gasto por ela. Nesse sentido, a Tabela 4.3 mostra o tempo de execução em µs de cada modelo de RNA para cada tipo de ataque. Este valor refere-se ao tempo de execução total de todos os registros de tráfego usados no teste divido pela quantidade de registros de tráfego, ou seja, tempo de detecção por registro de tráfego.

Percebe-se, pela Tabela 4.3, que o tempo médio de execução da MLP após passar pelo autoencoder é de 298,871 µs, que é a menor média entre os modelos de RNA. Porém, quando o tempo do autoencoder é combinado com essa MLP, o tempo é médio é de 505,697 µs, enquanto da MLP sem autoencoder é de 375,541 µs. Logo, não foi observada vantagem em usar o autoencoder para o banco de dados utilizado.

Caso o número de entradas fosse maior, a diminuição tempo devido a compreensão destas pelo codificador do autoencoder seria mais significativo. Neste caso, haveria vantagem em usar o autoencoder mesmo que sua acurácia seja mais baixa do que quando é empregado somente o

MLP, pois a latência na rede devido a aplicação da RNA seria menor.

A partir da Tabela 4.3, identifica-se que o tempo de execução da RNA SOM é o mais sensível em relação a quantidade de dados. Isso é esperado por causa que a dimensão do K-means varia com a quantidade de dados que ele recebe no treinamento, conforme a Equação (3.1). No caso da Infiltração, que apresenta a menor quantidade de dados, a SOM teve o menor tempo de processamento, apenas 30,612 µs, além de ter uma acurácia considerável de 96,939%.

Tabela 4.3: Tempo gasto em µs por cada modelo de RNA de acordo com o tipo de ataque por registro de tráfego.

	Força Bruta	**DoS**	**Web**	**Infiltração**	**Botnet**	**Varredura de Portas**	**DDoS**
MLP	111,075	62,925	195,420	1.918,367	212,435	64,160	64,408
CNN	96,615	64,918	180,153	1.571,429	189,983	68,175	66,107
LSTM	339,332	210,940	972,519	10.571,429	1.056,131	220,710	220,025
AE	65,660	59,998	121,374	938,776	139,033	59,215	63,729
MLP pós AE	73,907	58,061	162,595	1500,000	179,620	60,030	57,882
SOM	98,865	326,258	50,382	30,612	50,950	242,341	216,231

Da Figura 4.1a até a 4.5g são exibidas as matrizes de confusão de cada modelo de RNA para cada tipo de ataque. A matriz de confusão apresenta a frequência de classificação para cada classe, de modo que nas linhas tem-se as classes esperadas e nas colunas as classes previstas pelas RNAs. Ademais, a classe com valor 1 indica ataque, enquanto a de valor 0, aponta que o tráfego é normal. Logo, se o valor previsto é 1 e o valor esperado é 0, tem-se um falso positivo. De forma análoga, se o valor previsto é 0 e o valor esperado é 1, tem-se um falso negativo.

As Tabelas 4.4 e 4.5 mostram, respectivamente, a porcentagem de falsos positivos (FP) e de falsos negativos (FN), que são calculadas para cada matriz de confusão de acordo com as Equações (4.1a) e (4.1b).

$$FP = \frac{C_{1,0}}{C_{0,0} + C_{0,1}} \cdot 100\% \tag{4.1a}$$

$$FN = \frac{C_{1,0}}{C_{1,0} + C_{1,1}} \cdot 100\% \tag{4.1b}$$

onde $C_{1,0}$ e $C_{0,1}$ são, respectivamente, as quantidades de falso positivo e falso negativo, e $C_{0,0}$ e $C_{1,1}$ são, respectivamente, as quantidades de verdadeiro positivo e de falso negativo.

Tabela 4.4: Porcentagem de FP de cada modelo de RNA de acordo com o tipo de ataque.

	Força Bruta	DoS	Web	Infiltração	Botnet	Varredura de Portas	DDoS
MLP	6,251%	6,284%	1,210%	0,000%	10,664%	5,606%	0,212%
CNN	0,565%	7,589%	26,021%	0,000%	0,000%	0,787%	0,171%
LSTM	17,848%	0,362%	2,632%	0,000%	8,767%	2,924%	0,298%
MLP pós AE	19,552%	6,301%	48,260%	0,000%	18,007%	8,341%	2,533%
SOM	0,896%	9,171%	13,616%	0,000%	2,448%	4,545%	1,362%

Tabela 4.5: Porcentagem de FN de cada modelo de RNA de acordo com o tipo de ataque.

	Força Bruta	DoS	Web	Infiltração	Botnet	Varredura de Portas	DDoS
MLP	0,048%	0,208%	10,478%	23,077%	0,683%	0,396%	0,206%
CNN	0,024%	1,594%	3,698%	15,385%	36,007%	0,403%	0,086%
LSTM	81,004%	99,738%	24,706%	100,000%	74,227%	1,693%	0,071%
MLP pós AE	3,809%	3,656%	0,616%	100,000%	1,024%	5,007%	0,057%
SOM	17,545%	18,775%	96,302%	23,077%	19,454%	4,397%	1,023%

Figura 4.1: Matrizes de Confusão da MLP para cada tipo de ataque.

(a) Ataque de força bruta.

	0	1
0	4814	321
1	2	4199

(b) Ataque de DoS.

	0	1
0	80369	5389
1	158	75689

(c) Ataque de Web.

	0	1
0	653	8
1	68	581

(d) Ataque de Infiltração.

	0	1
0	85	0
1	3	10

(e) Ataque de Botnet.

	0	1
0	511	61
1	4	582

(f) Ataque de varredura de portas.

	0	1
0	36100	2144
1	189	47508

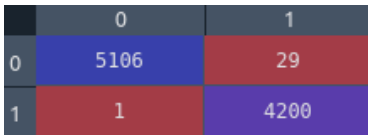
(g) Ataque de DDoS.

	0	1
0	29232	62
1	79	38351

Fonte: Autoria própria

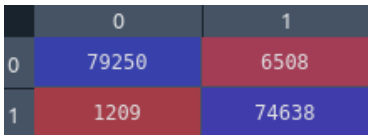
Figura 4.2: Matrizes de Confusão da CNN para cada tipo de ataque.

(a) Ataque de força bruta.



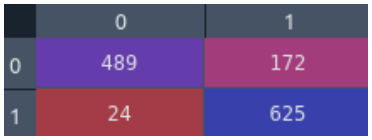
	0	1
0	5106	29
1	1	4200

(b) Ataque de DoS.



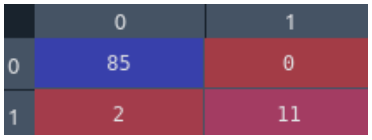
	0	1
0	79250	6508
1	1209	74638

(c) Ataque de Web.



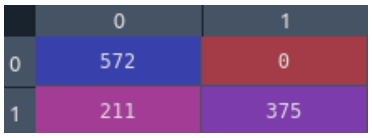
	0	1
0	489	172
1	24	625

(d) Ataque de Infiltração.



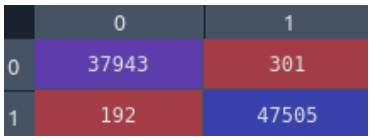
	0	1
0	85	0
1	2	11

(e) Ataque de Botnet.



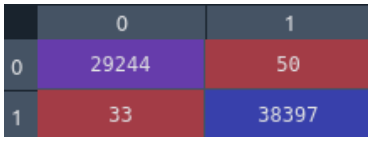
	0	1
0	572	0
1	211	375

(f) Ataque de varredura de portas.



	0	1
0	37943	301
1	192	47505

(g) Ataque de DDoS.

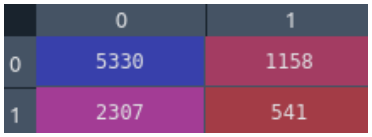


	0	1
0	29244	50
1	33	38397

Fonte: Autoria própria

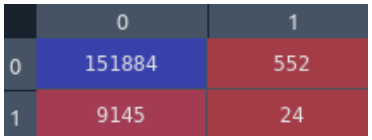
Figura 4.3: Matrizes de Confusão da LSTM para cada tipo de ataque.

(a) Ataque de força bruta.



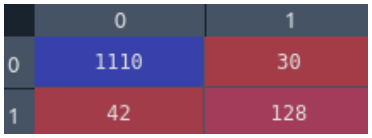
	0	1
0	5330	1158
1	2307	541

(b) Ataque de DoS.



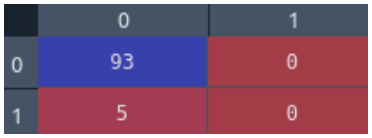
	0	1
0	151884	552
1	9145	24

(c) Ataque de Web.



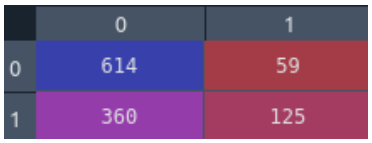
	0	1
0	1110	30
1	42	128

(d) Ataque de Infiltração.



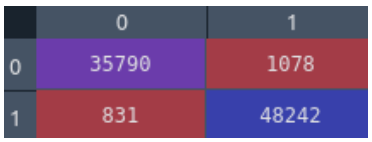
	0	1
0	93	0
1	5	0

(e) Ataque de Botnet.



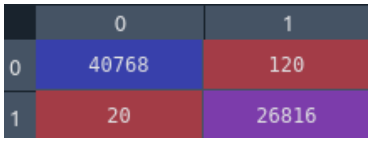
	0	1
0	614	59
1	360	125

(f) Ataque de varredura de portas.



	0	1
0	35790	1078
1	831	48242

(g) Ataque de DDoS.

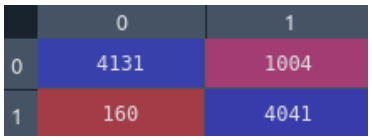


	0	1
0	40768	120
1	20	26816

Fonte: Autoria própria

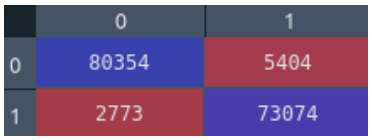
Figura 4.4: Matrizes de Confusão da MLP pós Autoencoender para cada tipo de ataque.

(a) Ataque de força bruta.



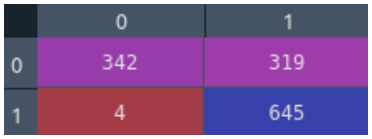
	0	1
0	4131	1004
1	160	4041

(b) Ataque de DoS.



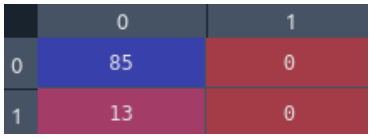
	0	1
0	80354	5404
1	2773	73074

(c) Ataque de Web.



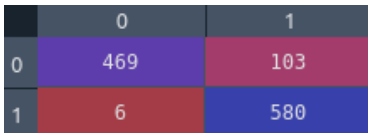
	0	1
0	342	319
1	4	645

(d) Ataque de Infiltração.



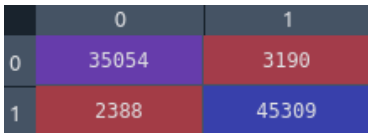
	0	1
0	85	0
1	13	0

(e) Ataque de Botnet.



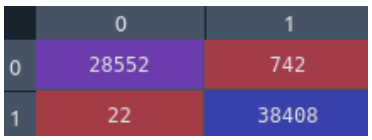
	0	1
0	469	103
1	6	580

(f) Ataque de varredura de portas.



	0	1
0	35054	3190
1	2388	45309

(g) Ataque de DDoS.

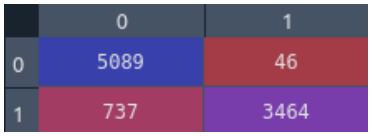


	0	1
0	28552	742
1	22	38408

Fonte: Autoria própria

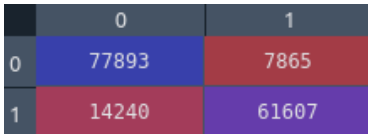
Figura 4.5: Matrizes de Confusão da SOM para cada tipo de ataque.

(a) Ataque de força bruta.



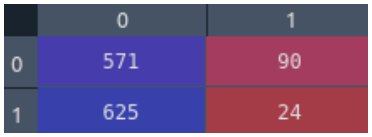
	0	1
0	5089	46
1	737	3464

(b) Ataque de DoS.



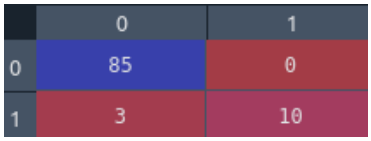
	0	1
0	77893	7865
1	14240	61607

(c) Ataque de Web.



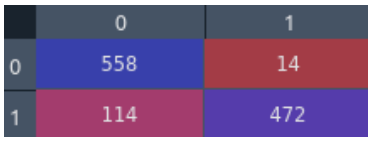
	0	1
0	571	90
1	625	24

(d) Ataque de Infiltração.



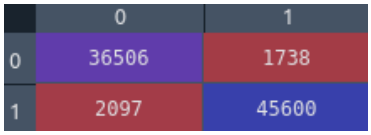
	0	1
0	85	0
1	3	10

(e) Ataque de Botnet.



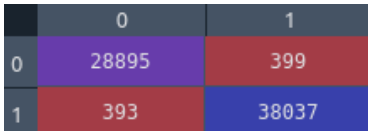
	0	1
0	558	14
1	114	472

(f) Ataque de varredura de portas.



	0	1
0	36506	1738
1	2097	45600

(g) Ataque de DDoS.



	0	1
0	28895	399
1	393	38037

Fonte: Autoria própria

Analisando as matrizes de confusão, é possível notar que para alguns casos, a porcentagem de falsos positivos é muito maior que a de falsos negativos. Como exemplo, cita-se a rede MLP em ataques de força bruta que obteve uma quantidade de falsos positivos 160,5 vezes maior que a de falsos negativos. A consequência disso, é que uma pouquíssima quantidade de ataque passa pelo detector, porém, 6,251% do tráfego foi erroneamente dado como intrusão. Caso o sistema que usa a RNA estivesse configurado para descartar todo pacote indicado como ataque, teria-se interrupção desnecessária na comunicação.

Há casos em que a porcentagem de falsos negativos é muito maior que a de falsos positivos. Nesses casos, tem-se que a RNA está deixando passar muito ataque. Esse cenário ocorre, por exemplo, na LSTM durante ataques Web.

Ao observar as matrizes de confusão, nota-se que, como esperado, os ataques DDoS apresentaram baixas porcentagens de falsos positivos e negativos. A LSTM, apesar de ter apresentado a maior acurácia para DDoS, não teve a menor porcentagem de falsos negativos, sendo que este feito foi obtido pela MLP pós Autoencoder, com 0,057% de falsos negativos. Isso exemplifica a importância de realizar uma análise tanto da acurácia, quanto da matriz de confusão.

Por meio das matrizes de confusão e das Tabelas 4.2 e 4.3, um engenheiro de redes é capaz de determinar qual o modelo mais adequado de RNA a ser empregado em uma rede de acordo com os tipos de ataques que esta recebe, com a quantidade de tráfego (*throughput*) que passa por ela e com a latência aceitável.

Capítulo 5

Conclusão

Neste trabalho foram desenvolvidos 5 modelos de Redes Neurais Artificiais para detecção de intrusão. Estas RNAs foram treinadas e testadas por meio de uma banco de dados com 7 tipos de ataques: DoS, DDoS, Botnet, ataque Web, força bruta, infiltração e varredura de portas.

Foi mostrado, para cada modelo de RNA e tipo de ataque, a acurácia, o tempo de execução e a porcentagem de falsos positivos e negativos. Desta maneira, este trabalho apresenta informações importantes para a escolhe do modelo de RNA a ser utilizado no projeto de um *software* de detecção de intrusão em uma rede de computadores.

Como trabalho futuro, pode-se aumentar o número de modelos de RNAs usadas e a quantidade de ataques. Além disso, seria interessante que as RNAs fossem frequentemente treinadas com novos ataques. Nesse sentido, poderia ser desenvolvido um *software* que captura novos ataques e envia-os para o banco de dados de treinamento da RNA. Dessa forma, a Rede Neural estaria preparada para ataques de dia zero.

Apêndice A

O link abaixo contém os pesos e estruturas das redes bem como os dados de treinamento usados para treinar e testar as RNAs e os códigos em Python para utilizar as RNAs implementadas.

https://drive.google.com/drive/folders/1Liop_Tw0NkaGozkRC_ut-pK4EOsEb0D2

Caso o link deixe de estar disponível por qualquer motivo, entrar em contato com o autor ou com o orientador por meio dos respectivos e-mails abaixo para que seja enviado um novo link:

heitor.eugenio.goncalves@gmail.com

ersilva@ufu.br

Referências Bibliográficas

[1] Thapa, N.; Liu, Z.; KC, D.B.; Gokaraju, B.; Roy, K. "Comparison of Machine Learning and Deep Learning Models for Network Intrusion Detection Systems". *Future Internet* 2020, 12, 167. https://doi.org/10.3390/fi12100167

[2] Kim, Jiyeon, Yulim Shin, and Eunjung Choi. "An intrusion detection model based on a convolutional neural network."*Journal of Multimedia Information System* 6.4 (2019): 165-172.

[3] Shieh, Chin-Shiuh, et al. "Detection of unknown ddos attacks with deep learning and gaussian mixture model."*Applied Sciences* 11.11 (2021): 5213.

[4] Pelletier, Zachariah, and Munther Abualkibash. "Evaluating the CIC IDS-2017 dataset using machine learning methods and creating multiple predictive models in the statistical computing language R."International Research Journal of Advanced Engineering and Science 5.2 (2020): 187-191.

[5] Khan, Adnan Shahid, et al. "A spectrogram image-based network anomaly detection system using deep convolutional neural network."*IEEE Access* 9 (2021): 87079-87093.

[6] Artificial Neural Network Tutorial. javaTpoint, ano. Disponível em: https://www.javatpoint.com/artificial-neural-network. Acesso em: 02 de dezembro de 2022.

[7] Neural Networks – History. Stanford Computer Science. Disponível em: https://cs.stanford.edu/people/eroberts/courses/soco/projects/neural-networks/History/history1.html.

[8] Capítulo 2 – Uma Breve História das Redes Neurais Artificiais. Deep Learning Book. Disponível em: https://www.deeplearningbook.com.br/uma-breve-historia-das-redes-neurais-artificiais/.

[9] Ameca: The Future Face of Robotics. Engineered Arts. Disponível em: https://www.engineeredarts.co.uk/robot/ameca/.

[10] A desvantagem da função ReLU. Toni Esteves. Medium. Disponível em: https://estevestoni.medium.com/a-desvantagem-de-utilizar-relu-4478589ef834.

[11] Model Fit: Underfitting vs. Overfitting. Amazon Machine Learning. Disponível em: https://docs.aws.amazon.com/machine-learning/latest/dg/model-fit-underfitting-vs-overfitting.html.

[12] An overview of gradient descent optimization algorithms. Sebastian Ruder. Disponível em: https://ruder.io/optimizing-gradient-descent/.

[13] Kingma, Diederik P., and Jimmy Ba. "Adam: A method for stochastic optimization."*arXiv preprint arXiv*:1412.6980 (2014).

[14] Kernel (image processing). Wikipédia. Disponível em: https://en.wikipedia.org/wiki/Kernel_(image_processing).

[15] Understanding of Convolutional Neural Network (CNN) – Deep Learning. Prabhu. Medium. Disponível em: https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148.

[16] What is the Convolutional Neural Network Architecture?. Analytics Vidhya. Disponível em: https://www.analyticsvidhya.com/blog/2020/10/what-is-the-convolutional-neural-network-architecture/.

[17] Farias, Thiago S., Rafael G. Rossi, and Mato Grosso do Sul–MS–Brazil. "Estudo Comparativo de Arquiteturas de Redes Neurais em Análise de Sentimentos."Disponível em: https://www.researchgate.net/publication/353347502_Estudo_Comparativo_de

_Arquiteturas_de_Redes_Neurais_em_Analise_de_Sentimentos.

[18] Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory."*Neural computation* 9.8 (1997): 1735-1780.

[19] Redes Neurais Recorrentes — LSTM. José R. F. Júnior. Medium. Disponível: https://medium.com/@web2ajax/redes-neurais-recorrentes-lstm-b90b720dc3f6.

[20] Building Autoencoders in Keras. The Keras Blog. Disponível em: https://blog.keras.io/building-autoencoders-in-keras.html.

[21] Pratella, David, et al. "A Survey of Autoencoder Algorithms to Pave the Diagnosis of Rare Diseases."*International journal of molecular sciences* 22.19 (2021): 10891.

[22] Descobrindo SOM, uma Rede Neural com aprendizado não supervisionado. Gisely Alves. Medium. Disponível em: https://medium.com/neuronio-br/descobrindo-som-uma-rede-neural-com-aprendizado-n%C3%A3o-supervisionado-f22bc1e55eca.

[23] Mapas auto-organizáveis (SOM). André Pacheco. Computação Inteligente. Disponível em: https://computacaointeligente.com.br/algoritmos/mapas-auto-organizaveis/.

[24] Kohonen, Teuvo. "Self-organized formation of topologically correct feature maps."*Biological cybernetics* 43.1 (1982): 59-69.

[25] Saldarriaga-Zuluaga, Sergio D., Jesús M. López-Lezama, and Nicolás Muñoz-Galeano. "Optimal coordination of over-current relays in microgrids using unsupervised learning techniques."*Applied Sciences* 11.3 (2021): 1241.

[26] K-means: o que é, como funciona, aplicações e exemplo em Python. Bruno Anastacio. Medium. Disponível em: https://medium.com/programadores-ajudando-programadores/k-means-o-que-%C3%A9-como-funciona-aplica%C3%A7%C3%B5es-e-exemplo-em-python-6021df6e2572.

[27] Tiffany Timbers, Trevor Campbell, and Melissa Lee. "Clustering". *Data Science: A First Introduction*. Disponível em: https://datasciencebook.ca/clustering.html.

[28] Moreno, Daniel. Introdução ao Pentest. *Novatec Editora*, 2019.

[29] DDoS Attack Trends for 2021 Q4. Cloudflare Radar. Disponível em: https://radar.cloudflare.com/reports/ddos-2021-q4.

[30] Moreno, Daniel. Pentest em Aplicações Web. *Novatec Editora*, 2017.

[31] OWASP TOP10. Disponível em: https://owasp.org/Top10/.

[32] SQL Injection Attack Definition. AVI Networks. Disponível em: https://avinetworks.com/glossary/sql-injection-attack/.

[33] Quanto Tempo para Descobrirem sua Senha?. IT Soluciona. Disponível em: https://itsoluciona.com.br/2022/08/18/quanto-tempo-para-descobrirem-sua-senha/nmpa

[34] Nmap: The Network Mapper. Disponível: https://nmap.org/

[35] Khosravi-Farmad, Masoud, Ali Ahmadian Ramaki, and Abbas Ghaemi Bafghi. "Moving target defense against advanced persistent threats for cybersecurity enhancement."2018 8th International Conference on Computer and Knowledge Engineering (ICCKE). IEEE, 2018.

[36] Sharafaldin, Iman, Arash Habibi Lashkari, and Ali A. Ghorbani. "Toward generating a new intrusion detection dataset and intrusion traffic characterization."*ICISSp* 1 (2018): 108-116.

[37] Patator. Lanjelot. Github. Disponível em: https://github.com/lanjelot/patator.