Universidade Federal de Minas Gerais
Departamento de Estatística
Programa de Pós-Graduação em Estatística

Bruce William Percílio Azevedo

CATEGORIZAÇÃO DE MALWARE EM BINÁRIOS UTILIZANDO TÉCNICAS DE NLP E DEEP LEARNING

Belo Horizonte
2025

Bruce William Percílio Azevedo

CATEGORIZAÇÃO DE MALWARE EM BINÁRIO UTILIZANDO TÉCNICAS DE NLP E DEEP LEARNING

Monografia de especialização apresentada ao Departamento de Estatística da Universidade Federal de Minas Gerais, como requisito parcial à obtenção do titulo de Especialista em Estatística.

Orientador: Marcos Antônio da Cunha Santos

Belo Horizonte
2025

Azevedo, Bruce William Percílio.

A994c Categorização de malware em binário utilizando técnicas de NPL deep learning [recurso eletrônico] / Bruce William Percílio Azevedo– 2025.
1 recurso online (41 f. il., color.) : pdf.

Orientador: Marcos Antônio da Cunha Santos
Monografia (especialização) - Universidade Federal de Minas Gerais, Instituto de Ciências Exatas, Departamento de Estatística.
Referências: f. 39-41

1. Estatística. 2. Probabilidade – Aprendizado de computador. 3. Aprendizado profundo. 4. Redes neurais (Computação) 5. Processamento de Linguagem Natural (Computação) I. Santos, Marcos Antônio da Cunha. II. Universidade Federal de Minas Gerais, Instituto de Ciências Exatas, Departamento de Estatística. III. Título.

CDU 519.2(043)

Ficha catalográfica elaborada pela bibliotecária Irénquer Vismeg Lucas Cruz
CRB 6/819 - Universidade Federal de Minas Gerais - ICEx

Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Estatística
Programa de Pós-Graduação / Especialização
Av. Pres. Antônio Carlos, 6627 - Pampulha
31270-901 – Belo Horizonte – MG

E-mail: pgest@ufmg.br
Tel: 3409-5923 – FAX: 3409-5924

ATA DO 334ª. TRABALHO DE FIM DE CURSO DE ESPECIALIZAÇÃO EM ESTATÍSTICA DE BRUCE WILLIAM PERCÍLIO AZEVEDO.

Aos vinte e dois dias do mês de janeiro de 2025, às 13:00 horas, com utilização de recursos de videoconferência a distância, reuniram-se os professores abaixo relacionados, formando a Comissão Examinadora homologada pela Comissão do Curso de Especialização em Estatística, para julgar a apresentação do trabalho de fim de curso do aluno **Bruce William Percílio Azevedo**, intitulado: "Categorização de Malware em Binário Utilizando Técnicas de NLP e Deep Learning", como requisito para obtenção do Grau de Especialista em Estatística. Abrindo a sessão, o Presidente da Comissão, Professor Marco Antonio da Cunha Santos – Orientador, após dar conhecimento aos presentes do teor das normas regulamentares, passou a palavra ao candidato para apresentação de seu trabalho. Seguiu-se a arguição pelos examinadores com a respectiva defesa do candidato. Após a defesa, os membros da banca examinadora reuniram-se sem a presença do candidato e do público, para julgamento e expedição do resultado final. Foi atribuída a seguinte indicação: o candidato foi considerado Aprovado condicional às modificações sugeridas pela banca examinadora no prazo de 30 dias a partir da data de hoje por unanimidade. O resultado final foi comunicado publicamente à candidata pelo Presidente da Comissão. Nada mais havendo a tratar, o Presidente encerrou a reunião e lavrou a presente Ata, que será assinada por todos os membros participantes da banca examinadora. Belo Horizonte, 22 de janeiro de 2025.

Prof. Marcos Antonio da Cunha Santos (Orientador)
DEST/UFMG

Prof. Luiz Henrique Duczmal
DEST/UFMG

Prof. Frederico R. B. Cruz
DEST/UFMG

Resumo

O uso da tecnologia vem aumentando com o passar dos anos, seu potencial para agilizar tarefas inerentes ao mundo moderno vem se mostrando imenso. Pessoas, empresas e até nações veem tendo os modos de realizar atividades corriqueiras transformados, melhorando sua execução. Porém o custo dessa transformação é a dependência. A ausência de tecnologias de comunicação, movimentação financeira ou fornecimento de recursos básicos, pode acarretar em perdas monetárias incalculáveis ou até mesmo em vidas.

Um dos principais recursos que pavimentaram essa guinada tecnológica é o software. O sistema operacional mais popular do mundo no momento, o Windows, suporta softwares no formato PE32. Porém dado a dimensão desta adoção, usuários mal intencionados se utilizam deste formato para escrever softwares maliciosos, sendo o principal formato utilizado atualmente, segundo o site de análises virustotal.

Existem diversas técnicas de análise de malware em formato PE32 atualmente, dividas em estático e dinâmico. No estático a análise se baseia em características do binário sem realizar sua execução. Enquanto que na análise dinâmica o binário é executado enquanto são extraídas informações durante o período de execução.

Ambas as análises são importantes, porém em ambientes modernos e automatizados, a velocidade é primordial. Para isso esse projeto propõe a utilização de métodos de Processamento de Linguagem Natural (PLN) e Deep Learning, para analisar pedaços do código de malware de binários no formato PE32 extraídos através de técnicas de desassembly, visando categorizar malwares e assim direcionar malwares conhecidos para análises especificas e que tomam mais tempo, visando criar uma estrutura de categorização de malware que pode servir de base para consciência situacional e tomada de decisões.

O Word2vec é um algoritmo que converte palavras em números levando em consideração o padrão que elas aparecem. Deste modo é possível classificar um malware a partir de partes de seu código extraídos direto do binário. Neste trabalho o Word2vec será alimentado com as funções/métodos extraídas. Sua saída irá alimentar um modelo rede neural recorrente do tipo Long-Short Term Memory (LSTM) que irá mapear os padrões, e usando as técnicas de aprendizado de máquina, realizar uma classificação do malware.

Palavras-chave: análise de malware; binário; redes neurais; aprendizado profundo.

Abstract

The use of technology has been steadily increasing over the years, and its potential to expedite tasks in our current era is proving to be significant. Individuals, businesses, and even nations have experienced a transformation in their daily activities, often leading to improvements. However, this transformation comes with a downside: dependency. The lack of technologies for communication, financial transactions, or the provision of essential resources can result in substantial financial losses or even loss of life.

One of the key factors driving this technological shift is software. Currently, the most widely used operating system is Windows, which supports software in the PE32 format. Unfortunately, due to its popularity, this format is often exploited by malicious users to create harmful software. According to VirusTotal, PE32 remains the primary format for such malicious applications today.

Currently, there are several techniques for analyzing malware in the PE32 format, which can be categorized into two types: static and dynamic analysis. Static analysis involves examining the characteristics of the binary without executing it. In contrast, dynamic analysis requires the execution of the binary while extracting information during its runtime.

Both analyses are important, but in modern automated environments, speed is crucial. This project proposes using natural language processing (NLP) and deep learning techniques to analyze malware code from binaries in PE32 format, which are extracted through disassembly methods. The goal is to categorize malware, allowing known threats to be directed toward more thorough analyses. This approach aims to create a malware categorization structure that can enhance situational awareness and support decision-making.

Word2Vec is a natural language processing (NLP) algorithm that converts words into numerical representations based on the patterns in which they appear. This technique can be used to classify malware based on its assembly code. For this purpose, Word2Vec will be trained using functions extracted from binaries in PE32 format. The resulting output will then be input to a Long Short-Term Memory (LSTM) recurrent neural network model, which will identify patterns and classify the malware effectively through machine learning techniques.

Key words: malware analysis; binary; neural network; deep learning.

Lista de tabelas

Tabela 1 - Número de malwares 26
Tabela 2- Parâmetro do modelo word2vec 28
Tabela 3 - Lista de labels e valores 29
Tabela 4 - Descrição de parametros para o Embedding 30
Tabela 5 - Resultados previstos e obtidos da análise 35
Tabela 6 - Tabela de comparação 36
Tabela 7 - Ferramentas e versões 36

Lista de Figuras

Figura 1 - Long-short term memory 18
Figura 2 - modelo RNN 19
Figura 3 - Modelo LSTM 20
Figura 4 - Modelo proposto 25
Figura 5 - Campos do arquivo CSV gerado pelo programa C++ 27
Figura 6 - Array de funções para array de números 28
Figura 7 - Camadas do modelo LSTM 29
Figura 8 - Acurácia do modelo 32
Figura 9 - Perda do modelo 33
Figura 10 - Matriz de confusão 34

Lista de siglas

SO – Sistema operacional
ML – Machine Learning
PE – Portable Executable
PLN – Processador de Linguagem Natural
RNN – Redes Neurais Recorrentes
CNN – Convolutional Neural Network
LSTM – Long-short Term Memory
NLP – Network Language Processing

Sumário

1. Introdução 10

2. Estado da arte 12

2.1. Executável portátil 12

2.2. Tipos de malware 13

2.3. Análise Léxica 14

2.4. Machine Learning 15

2.5. Aprendizagem profunda 17

2.6. Long-Short Term Network 19

2.7. Trabalhos correlatos 23

3. Modelo proposto 25

3.1. Construção de conjuntos de dados 25

3.2. Disassembly de binários para extrair funções 26

3.3. Análise léxica 27

3.4. Criação e utilização de modelos LSTM 28

4. Resultados 32

4.1. Resultados do modelo treinado 32

4.2. Estrutura de automação 34

4.3. Teste prático com amostras reais 35

4.4. Comparação de trabalhos relacionados 35

4.5. Ferramentas e versões 36

4.6. Contribuições 36

5. Conclusão e trabalhos futuros 38

Referências 39

1. Introdução

Sistemas computacionais tornaram-se parte inerente do cotidiano de indivíduos, empresas e até nações. A ampla adoção desses meios vem melhorando significativamente a eficiência e a velocidade de atividades diversas. Porém, esses ganhos vieram com custo da dependência. O preço da privação de sistemas de infraestruturas críticas ou financeiras, é incalculável, podendo até mesmo levar à perda de vidas.

Dentre as tecnologias que pavimentam a adoção de sistemas computacionais, o software é uma das principais. Dado sua natureza decentralizada qualquer um com conhecimento possui a capacidade de gerar um, tornando essa tecnologia acessível. Um dos formatos mais utilizados por desenvolvedores é o PE32, suportado pelo sistema operacional Windows.

No entanto, pessoas mal intencionadas se utilizam dessa estrutura tecnológica criada para facilitar o processo de desenvolvimento e adesão de softwares, para criar programas maliciosos. Existem diversas categorias de programas maliciosos, uma das mais conhecidas é o malware. Em tese, malware é qualquer código/binário que causa danos a um ativo computacional. Porém, devido a diversidade, foram criadas nomenclaturas com base no comportamento. Malwares criados na estrutura PE32 ocupam o primeiro lugar entre os softwares maliciosos identificados em análises feitas pelo virustotal (https://www.virustotal.com/gui/stats).

Os métodos atuais de análise de malwares binário no formato PE32 são divididos em 2 categorias: estáticos e dinâmicos [1]. No método estático, a análise é efetuada sem executar o binário, tendo como as técnicas mais utilizadas o uso de hashing [2] e as análises manuais [3] utilizando métodos de disassembly. Enquanto os métodos dinâmicos realizam a execução do binário, como processos que utilizam sandboxing [4] e monitorização de pacotes IP, visando identificar anomalias no comportamento [5] ou possíveis danos ao sistema computacional. Porém, devido ao elevado grau de complexidade da natureza dessas análises, as técnicas atuais, embora eficazes, são consideradas lentas, consumindo tempo considerável para categorizar o binário.

O objetivo e as contribuições desta investigação são propor uma estrutura que objetiva categorizar softwares maliciosos, visando agilizar o processo de análises para binários construídos

sobre a estrutura PE32. Esta análise não substitui as técnicas existentes, é uma maneira de verificar se determinada técnica deve ou não ser empregada na verificação de determinado tipo de malware.

O primeiro passo para realizar a análise foi a construção de uma base de dados composta por 6 tipos diferentes de malwares. Essa base de dados foi preenchida por softwares previamente analisados e compartilhados em um repositório publico de malware. Uma vez em posse desses malwares no formato binário, foi utilizado um programa em C++ para extrair as funções requeridas a serem executadas pelo sistema operacional, contidas dentro de uma lista, inerente ao binario, chamada “import”. Essas funções vão alimentar um Processador de Linguagem Natural (PLN) que identifica estas funções e utiliza técnicas matemáticas para realizar sua analise léxica e conversão em array numérico. Uma vez com a estrutura em formato número, ela alimentara um modelo de Deep Learning, criado para identificar os padrões e entender quais valores pertencem a cada tipo de malware. Em tese, utilizando entradas similares para treinar o modelo, será possível categorizar o tipo de malware.

Para realizar uma abordagem prática dessa análise, os modelos de NLP e Deep Learning serão inseridos dentro de uma estrutura automatizada construída sobre tecnologia de conteinerização. Essa estrutura recebe malwares em formato binário e vai realiza a categorização de forma automatizada.

Este documento está organizado em seções. A seção 2 apresenta os antecedentes teóricos, abrangendo a estrutura do arquivo PE32, os tipos de malware, a análise lexical e a aprendizagem profunda. A seção 3 se descreve trabalhos relacionados ao tema proposto. A seção 4 apresenta o método proposto, abrangendo a construção do conjunto de dados, a análise lexical e a memória de longo e curto prazo. Os resultados são apresentados e discutidos na seção de resultados e, por fim, a seção de conclusões termina com a conclusão e trabalhos futuros.

2. Estado da arte

O PE32 é uma estrutura muito utilizada por usuários mal intencionados na criação de malwares. No entanto, as técnicas atuais de análise deste tipo de estrutura, dada a complexidade exigida, são demoradas ou requerem intervenção manual. Este documento introduz uma proposta de modelo construído utilizando NLP e Deep Learning. Porém, devido à profundidade da proposta e às tecnologias atuais que embasam a proposta, é necessário compreender mais a fundo os detalhes dos componentes que a envolvem. As subsecções a seguir fornecem mais informações sobre esses componentes.

2.1. Executável portátil

Em 1993, a Microsoft lançou o Windows NT e introduziu o a estrutura Portable Executable (PE), que fazia parte das especificações originais que seriam reconhecidas pelos componentes que executam aplicações Win32. O termo “Portable Executable" foi criado para fornecer uma estrutura única que pudesse ser executada em todas as versões do Windows e fosse compatível com todos os modelos existentes de CPU. Este objetivo, em geral, foi alcançado, e a mesma estrutura tornou-se o padrão para sua versão mais robusta, o Windows Server, e posteriormente seus sucessores, como o Windows 95, 98 e etc [6].

O PE32 é uma estrutura de construção e mapeamento de arquivos que o sistema operacional utiliza para executar programas. Estes programas podem possuir diversas extensões de arquivos, desde que obedeçam às delimitações e posicionamento de pilhas requeridas. As extensões que incorporam esse formato mais comuns são .exe e .dll. Todos eles seguem a mesma estrutura, com algumas diferenças que indicam como os arquivo com extensão diferente deve ser tratados [6].

A maior parte do software na estrutura PE depende fortemente de funções de outras DLLs (bibliotecas disponibilizadas pelo criador do SO), que precisam de ser importadas. Durante o processo de carregamento do software no formato PE pelo SO, a estrutura responsável por realizar o carregamento dentro dos recursos controlados pelo Windows, localiza e importa essas funções, tornando-as acessíveis na memória como parte do software.

Com a introdução do Windows de 64 bits, foi necessário efetuar algumas alterações à estrutura PE. Isto resultou na necessidade da atualização do tamanho da pilha da estrutura, essa nova estrutura maior foi denominada PE32+. Durante esse processo não foram adicionados novos campos, porém

um campo foi removido para deixar lugar para dados. Os campos restantes foram alargados da estrutura de 32 bits para a estrutura de 64 bits. Na maioria dos casos, os arquivos dispostos na estrutura de 32 bits são automaticamente reconhecidos em sistemas operacionais com uma estrutura construída com base na estrutura de 64 bits.

A estrutura PE32 é a que vem sendo amplamente aderida por programadores de software; no entanto, a sua adesão extensiva tornou sua estrutura uma das mais eleitas por indivíduos com intenções maliciosas, visando atingir o maior número possível de vítimas no desenvolvimento de malware. Esse formato é classificado como a estrutura com mais casos de malwares reportados pelo Virustotal (https://www.virustotal.com/gui/stats).

2.2. Tipos de malware

O termo “malware” significa “software malicioso”. Tal como o nome indica, o malware é um software concebido para prejudicar os sistemas computacionais ou os seus utilizadores. Existem diversas formas utilizadas para isso, como por exemplo roubar informações, corromper arquivos, conceder acesso não autorizado a recursos sigilosos ou simplesmente incomodar os usuários [7].

Os criadores de malware estão constantemente em busca de novos métodos e ferramentas para atingir os seus objetivos escusos. As técnicas utilizadas por esses usuários mal intencionados passam por processos complexos visando esconder seu real objetivo para aumentar a taxa de êxito. São utilizadas técnicas como a ofuscação de código e a exfiltração de dados, visando infectar o maior número possível de vítimas. Ultimamente, o número de incidentes de segurança causados por malwares tem aumentado consideravelmente.

Com o objetivo de facilitar a análise de malwares, especialistas utilizam suas características visando categorizara-los, tais como o comportamento, assinaturas, objetos com quais eles se comunicam, funções utilizadas por eles. Comunidades, indivíduos, empresas e até governos, criam técnicas e ferramentas especificamente para esse fim. Porém, uma vez que não é possível estimar com exatidão a quantidade exata de malware existentes, estas categorias se limitam a acumular malwares similares. Alguns malwares possuem uma incidência maior e são mais efetivos, com exemplos estruturados inclusive no formato PE32. Veja a seguir alguns dos principais tipos de malware encontrados atualmente.

Downloader: Um downloader é um tipo de software criado para descarregar e instalar outros programas maliciosos em sistemas computacionais infectados. Os downloaders usam técnicas avançadas para esconder o seu verdadeiro objetivo e camuflam o tráfego de rede criado para comunicação com fontes externas, tornando-os difíceis de detectar [8].

Keylogger: Os keyloggers são programas criados para roubar informações confidenciais. Esses programas são desenvolvidos utilizando técnicas especificas para coletar dados inseridos por periféricos. Seu objetivo principal é localizar PINs, números de cartões de crédito, textos confidenciais e outros dados valiosos [9].

Miner: Miner é um tipo de malware projetado especificamente para tomar posse dos recursos computacionais de suas vítimas e usá-los para minerar criptomoedas. O principal objetivo desta ferramenta é gerar lucro financeiro para quem o projetou [10].

Ransomware: O ransomware é um tipo de software malicioso que, quando executado, desativa as funções dos sistemas computacionais de alguma maneira. Normalmente, o ransomware funciona encriptando arquivos em computadores e, em seguida, são programados para fazer exigências específicas em troca da restauração desses arquivos. Como se trata de uma forma de chantagem para devolver recursos inerentes na execução de atividades, esse tipo de malware pode ser muito prejudicial para indivíduos e organizações [11].

Rootkit: É um tipo de software malicioso desenvolvido para ocultar a existência de outros programas maliciosos de ferramentas responsáveis por analisa-los, como antivírus. É prática comum o malware utilizar estes programas para permitir a sua própria instalação e ativação em ambientes já infectados, mantendo acessos privilegiados. Uma vez instalados, estes programas podem modificar estruturas do sistema operacional, arquivos específicos e até mesmo ferramentas responsáveis pela detectação de softwares maliciosos [12].

Spyware: O spyware é um tipo de software malicioso que é instalado secretamente em sistemas computacionais de suas vítimas, sem o seu conhecimento ou consentimento. Uma vez que for ativado, começa a monitorizar o comportamento do sistema, os arquivos e as atividades dos usuários, sendo capaz de ate mesmo roubar credenciais de acesso e outras informações sensíveis. Os dados recolhidos pelo spyware são enviados para os criadores do malware [13].

2.3. Análise Léxica

A semelhança semântica é uma questão crucial em linguística, especialmente quando se trata de compreender o significado das palavras. Em termos linguísticos, duas palavras podem ter o mesmo significado. O processo de cálculo do grau de semelhança entre elas tem sido tradicionalmente baseado no pensamento humano. No entanto, este cálculo tem servido de base para o desenvolvimento de técnicas no domínio da informática, como o Processamento de Linguagem Natural (PLN) e a Extração de Texto.

A PLN é um dos ramos da inteligência artificial. Ela trata da interação entre os sistemas informáticos e a linguagem humana [14]. Visando a identificação de intenções dentro de textos, os sistemas informáticos estão sendo utilizados para processar a linguagem humana de entrada, realizar cálculos utilizando métricas definidas sobre os dados de entrada, e por fim serão capazes de interpretar com base no valor gerado pelos cálculos.

Uma das ferramentas desenvolvidas com base nesse conceito foi o Word2vec [15]. Criado em 2013 é atualmente uma das técnicas mais populares para calcular a semelhança e a relevância das palavras. Fornece uma representação vetorial única para cada palavra, tendo em conta a semântica ao atribuir tokens semelhantes para fornecer representações semelhantes.

O Word2vec não é considerado o melhor ou o único modelo de PNL baseado na incorporação de palavras, analogias e semelhanças. No entanto, é simples e acessível; qualquer pessoa pode descarregar e utilizar o seu código. E será empregado nesse trabalho a versão do da biblioteca em python, Gensim [16].

2.4. Machine Learning

Machine Learning (ML) (aprendizagem de máquina) é um termo utilizado para descrever algoritmos computacionais que permitem aos sistemas resolver problemas sem serem explicitamente programados para tal. O desenvolvimento de técnicas baseado nesse conceito permite aprender com a experiência, desenvolver conceitos a partir de exemplos e extrair padrões de dados brutos [17].

As técnicas criadas para suportar o aprendizado de máquina possuem suas raízes em abordagens probabilísticas, que visam seu emprego para modelar e inferir dados. A probabilidade estatística é um conjunto de ferramentas matemáticas que auxiliam no entendimento e descrição de

incertezas e na variabilidade de eventos. Utilizando-as como base, e contextualizando para ML, é possível modelar imprecisões e a fazer previsões de dados desconhecidos, utilizando como base dados observados [18]. Alguns dos modelos matemáticos que são amplamente utilizados dentro do campo de ML são os modelos de classificação probabilística e de regressão probabilística.

- **Classificação probabilística**: Quando queremos classificar algo em categorias, podemos modelar a probabilidade de cada classe dada as características do objeto. Em vez de simplesmente dizer "essa imagem é de um gato", podemos dizer "Existe 80% de chance de que essa imagem seja de um gato e 20% de que seja de um cachorro". Modelos como Naive Bayes utilizam probabilidades condicionais para tomar essas decisões.
- **Regressão probabilística**: Em vez de prever um valor exato, podemos prever uma distribuição de probabilidades para um valor, como em regressão logística ou outros modelos de regressão. Por exemplo, em vez de prever "a temperatura será 30°C", podemos prever uma distribuição de temperatura, indicando que a temperatura tem 90% de chance de estar entre 28°C e 32°C.

Outro conceito amplamente empregado e disseminado no campo de ML é a ideia de estimativas de parâmetros. A partir de modelos probabilísticos, objetivamos estimar os parâmetros do modelo que melhor explicam os dados observados. Os métodos mais disseminados para essas tarefas utilizam MLE e MAP [20].

- **Máxima Verossimilhança (MLE - Maximum Likelihood Estimation)**: Este método visa estimar os parâmetros que maximizam a função de verossimilhança, obtida a partir dos dados do modelo. Em outras palavras, escolhemos o modelo e estimamos os parâmetros que fazem com que os dados observados sejam os mais prováveis. A MLE é usada para ajustar muitos modelos de aprendizado supervisionado, como regressão linear e redes neurais.
- **Máxima a Posteriori (MAP - Maximum A Posteriori Estimation)**: É uma variação da MLE, que leva em consideração distribuição de **probabilidades a priori**. Ou seja, o MAP busca maximizar uma função de distribuição de probabilidades a posteriori, levando em conta os dados e o conhecimento prévio sobre os parâmetros (a priori). Isso pode ser útil, por exemplo, quando há informações adicionais ou quando queremos evitar soluções extremas para os parâmetros (um caso de regularização).

2.5. Aprendizagem profunda

As técnicas introduzidas pelo ML foram adaptadas e aplicadas em diversas áreas do conhecimento. Porém dado sua popularidade, problemas complexos que exigem soluções especificas ficaram evidentes, surgindo novas abordagens para esses problemas. Estas abordagens não se destinam a substituir o ML, mas sim a complementá-lo, especialmente quando se trata de especificidades. Uma dessas abordagens é a Deep Learning (aprendizagem profunda). Ela permite que os sistemas informáticos aprendam conceitos complexos dividindo-os em conceitos mais simples e criando uma hierarquia de camadas. Isto é conseguido através da criação de uma rede neural para atingir este objetivo [21].

Uma rede neuronal é um sistema complexo composto por muitas partes interligadas chamadas neurónios, com uma estrutura semelhante à do cérebro humano. Estes neurónios estão dispostos em camadas e a "aprendizagem profunda" refere-se à aprendizagem obtida através de redes neuronais com várias camadas de neurónios. As camadas de uma rede neuronal incluem a camada de entrada, as camadas ocultas e a camada de saída (Figura 1).

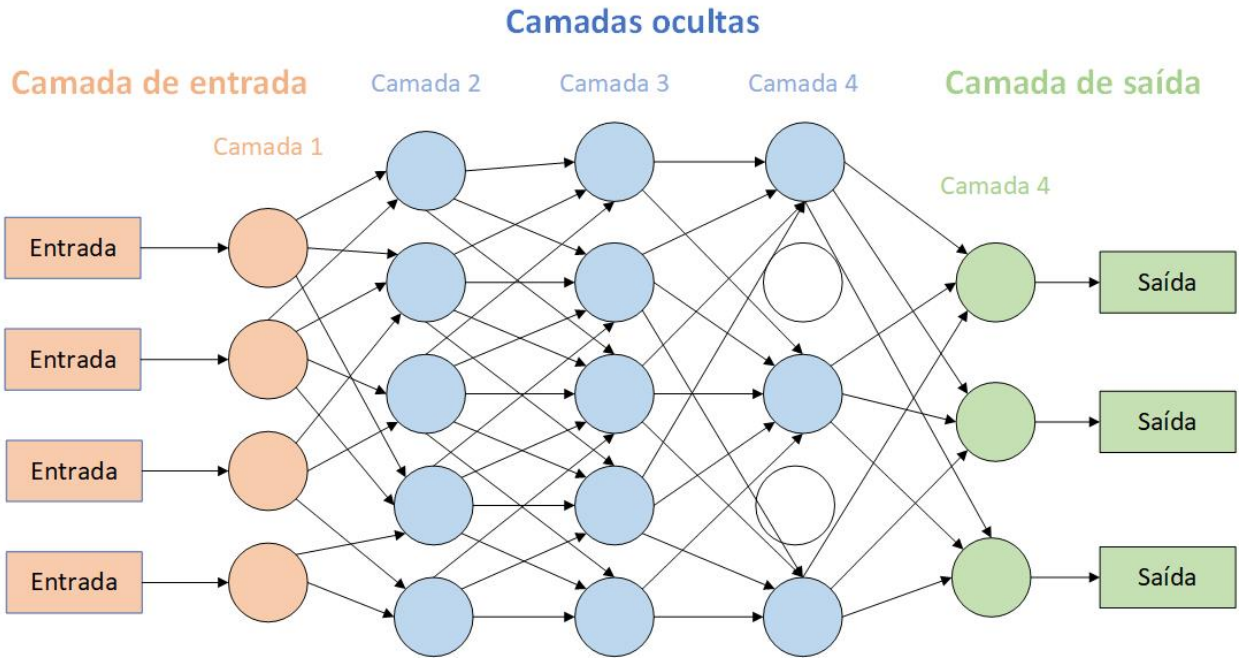

Figura 1 - Long-short term memory

Cada neurónio de uma rede neural recebe vários números como entrada e produz um único valor de saída, denominado ativação neuronal. Em redes unidirecionais mais simples este processo ocorre apenas uma vez e os dados nunca voltam a passar pelo mesmo neurónio. O processamento de dados em um neurónio depende apenas da entrada que recebe e não tem qualquer relação com dados processados anteriormente.

Um dos tipos de rede neural são as redes neuronais recorrentes (RNN), que é utilizada para lidar com o fluxo de dados que passam pelos neurónios, formando um ciclo dentro da rede. As RNN utilizam um estado interno que pode ser armazenado durante longos períodos de tempo e pode ser acessado a partir de outros neurónios.

Durante a fase de treino, a rede neuronal pode utilizar a retropropagação para calcular os pesos dos elos do grafo, de acordo com o algoritmo gradiente descendente. O gradiente descendente é o algoritmo de otimização utilizado para minimizar uma função de custo ou perda, frequentemente utilizado em problemas de aprendizado de máquina e redes neurais. O objetivo é ajustar os parâmetros do modelo (como os pesos de uma rede neural) para que obter previsões mais precisas, ou seja, para reduzir o erro entre as previsões do modelo e os valores reais. Os pesos em cada ligação dos neurônios

normalmente são iniciados com valores aleatórios e em seguida atualizados em cada iteração do algoritmo, tornando os parâmetros mais ajustados ao modelo a partir da base de dados utilizada.

Os cálculos dos gradientes descendentes nas redes neurais podem partir das camadas mais profundas e regressar à camada inicial no processo de retropogramação. No entanto, ao treinar uma RNN, os gradientes provenientes das camadas mais profundas devem passar por operações contínuas deste tipo. O algoritmo de retropropagação usa a regra da cadeia para calcular os gradientes, e quanto mais longe eles estiverem das camadas anteriores, mais multiplicações com números pequenos ocorrerão. Isto leva a um gradiente exponencialmente decrescente, o que significa que a informação é perdida e o modelo já não pode aprender com os dados iniciais.

2.6. Long-Short Term Network

Sepp Hochreiter e Jürgen Schmidthuber introduziram o conceito de Rede de Longo e Curto Prazo (LSTM) em 1997. A LSTM é uma arquitetura de rede neural que aborda o problema do gradiente descendente, impondo um fluxo de erro constante nos estados internos. Isto é conseguido através de um algoritmo baseado no gradiente que evita tanto a rutura como a fuga [23].

Todas as redes neuronais recorrentes têm a forma de uma cadeia de módulos repetitivos da rede neuronal (Recorrent Neural Network – RNN). Nas RNN normais, este módulo repetitivo terá uma estrutura muito simples, como uma única camada tanh (Tangente hiperbólica).

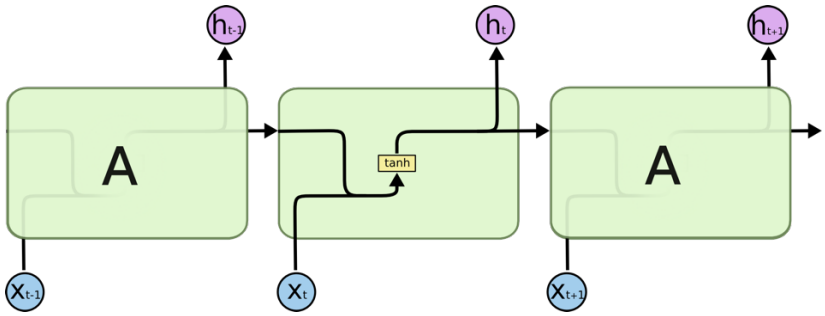

Figura 2 - modelo RNN

(imagem retirada de https://colah.github.io/posts/2015-08-Understanding-LSTMs)

A figura 2 mostra um exemplo de rede neuroal, onde X é o vetor passado como argumento de entrada e t é cada uma das posições do vetor. A entrada passa por pela função tanh e gera um vetor com saídas H, sendo t cada posição do vetor de saída.

Os LSTMs também têm a estrutura semelhante a uma cadeia, mas o módulo de repetição tem uma estrutura diferente. Em vez de ter uma única camada de rede neural, existem quatro, que interagem entre sí.

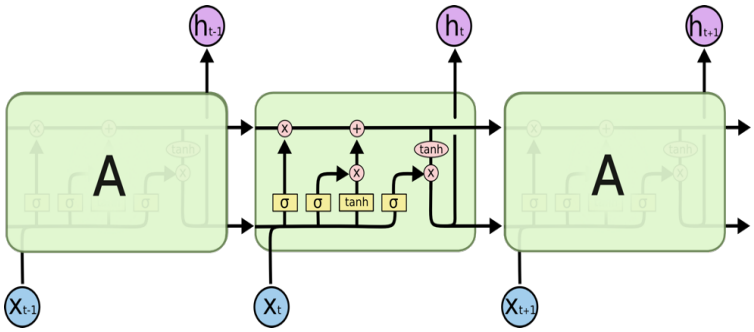

Figura 3 - Modelo LSTM

O LSTM é composto por uma arquitetura especial que inclui portões para controlar o fluxo de informações a cada vez que um neurônio executa determinada ação sobre o parametro passado, sendo esse processo de executar todos os dados através de uma rede neural conhecido como época. A estrutura do modelo é definida e junto com ela o número de épocas, ou seja, cada vez que os dados passam pelos portões do LSTM e o metodo matematico definido no modelo é empregado.

O principais portões do LSTM são:

- Portão do esquecimento: Decide que informações devem ser descartadas do estado do neurônio.
- Portão de entrada: Determina que novas informações são adicionadas ao estado do neurônio.
- Portão de saída: Controla quais as informações do estado do neurônio que são apresentadas como estado oculto.

Em cada epoch (t) um neurônio LSTM actualiza o seu estado interno e o seu estado oculto utilizando as seguintes equações:

Portão do esquecimento (f_t): Decide qual a proporção do estado anterior da neurônio C_{t-1} para esquecer.

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

Onde:

- σ é a função de ativação sigmoid
- W_f é a matriz de pesos para a porta de esquecimento.
- b_f é a polarização da porta de esquecimento.
- h_{t-1} é o estado oculto anterior (saída da neurônio LSTM anterior).
- x_t é a entrada de corrente.

Portão de entrada (i_t): Decide a quantidade de informação nova a adicionar ao estado do neurônio.

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

Estado da neurônio de candidatura $(\tilde{C}_t)$: Uma potencial atualização do estado do neurônio, que é depois escalada pela porta de entrada.

$$\tilde{C}_t = \tanh\,(W_C \cdot [h_{t-1}, x_t] + b_C)$$

Onde tanh é a função tangente hiperbólica.

Estado de atualização do neurônio(C_t): O estado anterior do neurônio é atualizado utilizando a porta de esquecimento e o estado da neurônio candidata (modulado pela porta de entrada).

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

Portão de saída (o_t): Determina a parte do estado do neurônio que deve ser apresentada como estado oculto.

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

Estado de atualização oculta (h_t): O novo estado oculto é calculado aplicando a porta de saída ao estado atualizado do neurônio.

$$h_t = o_t \cdot \tanh(C_t)$$

Neste passo, o LSTM produz o seu primeiro estado oculto que é passado para o passo de tempo seguinte. Em cada época, o LSTM atualiza os seus estados internos com base nos portões. O ponto chave é que a porta de esquecimento controla que informação passada deve ser descartada, enquanto a porta de entrada determina que nova informação é adicionada. O estado do neurônio atua como a memória da rede, e o estado oculto transporta a informação para o passo de tempo seguinte.

2.7. Trabalhos correlatos

Dai et al [25]. propuseram a utilização de redes neurais para classificar malware através da análise de dumps de memória retirados de programas durante sua execução. Executaram programas maliciosos numa sandbox e capturaram a memória à medida que os programas eram executados para criar uma base de dados, e depois utilizaram esta base de dados para treinar um modelo baseado em redes neuronais. Inicialmente, utilizaram uma rede neural convolucional (CNN) para construir um modelo inicial, sendo mais tarde atualizado para uma estrutura que utiliza LSTM. Em comparação com o modelo inicial, o atualizado deu melhores resultados, principalmente devido a um melhor tratamento do gradiente ofertado pelo algoritmo.

Young Jun Lee et al [26]. propuseram uma estrutura de análise léxica própria para analisar código descompilado para assembly, a fim de encontrar vulnerabilidades em formas semelhantes de código exposto. Para mapear as semelhanças no código, utilizaram a CNN. O procedimento revelou-se eficaz, com uma taxa de validação de 94%, indicando o sucesso da utilização combinada da análise léxica e da aprendizagem profunda. A utilização da análise léxica foi fundamental para o sucesso da comparação e validou a eficácia da utilização conjunta destas tecnologias.

Wartschinski [27] criou uma ferramenta de análise de vulnerabilidades para a sua tese de mestrado utilizando Python e recursos gestão de código, como github. Iniciou mapeando repositórios de Python primário no GitHub e descarregar os dados comprometidos. De seguida, utilizou o Word2Vec para mapear padrões de tokens dentro da linguagem e criou uma estrutura numérica para alimentar um modelo de rede neural baseado em LSTM. Este modelo utiliza a classificação binária para verificar se o código é vulnerável. Ao converter os scripts gerados numa ferramenta de código aberto, conseguiu reproduzir todo o processo com os modelos Word2Vec e LSTM.

Visweswaran [28] et al. analisaram arquivos na estrutura PE32, que culmino na extração de funções de chamada de API, a partir de um conjunto de dados de malware, sendo empregadas técnicas de disassembly, para possibilitar a extração, e tokenização, para converter os resultados obtidos em uma base numérica, preparada para ser manuseada. Em seguida, inseriram esta nova base numérica em algoritmo de rede neural CNN, cujo objetivo era mapear os padrões dos tipos de malware. No entanto, dentro do contexto do experimento, a solução não pode ser recriada devido a utilização da técnica utilizada para a conversão numérica, não armazenar detalhes necessários para recriação do

processo, obrigando o reprocessamento cada vez que o modelo é utilizado. Isso acarreta, mesmo com o mesmo conjunto de dados, em diferenças na estrutura em comparação com a conversão original, nem que seja apenas por alguns detalhes.

3. Modelo proposto

Os processos de análise de arquivos na estrutura PE32 podem ser demorados e requerem frequentemente a intervenção manual de um especialista. Para auxiliar na construção de uma estrutura que visa resolver este problema, utilizamos uma proposta que faz uso de técnicas de disassembly num conjunto de dados de malware para extrair funções de binários e criar um conjunto de dados em formato CSV. Este conjunto de dados irá treinar um modelo de análise léxica utilizando o algoritmo word2vec, que será depois alimentado como entrada para um modelo de aprendizagem profunda utilizando LSTM. A Figura 4 mostra uma representação visual do método proposto.

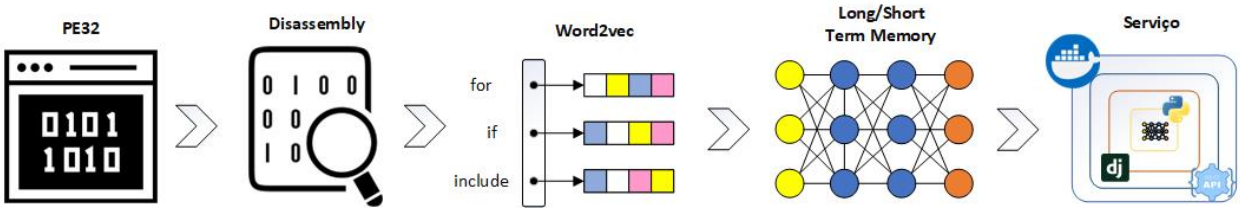

Figura 4 - Modelo proposto

Utilizando os modelos word2vec e LSTM, esta proposta cria uma estrutura de conteinerização capaz de receber binários de malware e identificar o seu tipo. Para completar cada tarefa planeada, é necessária uma série de passos, descritos a seguir.

3.1. Construção de conjuntos de dados

O primeiro passo para a implementação da solução proposta foi a coleta de um conjunto de dados constituído por diferentes tipos de malware. De forma a desenvolver uma proposta o mais próxima possível de um ambiente real, neste trabalho utilizamos tipos de malware reais provenientes de informações, datas e detalhes amplamente divulgados. Para construir o conjunto de dados, utilizámos o repositório de partilha de malware Virus Share (https://virusshare.com/), uma estrutura de partilha gratuita e aberta destinada exclusivamente ao armazenamento de malware, que contém malware em diferentes formatos e estruturas. Foi necessário filtrar as amostras para obter apenas binários na estrutura PE32.

Foram escolhidos seis tipos de malware, cujas amostras foram descarregadas e armazenadas localmente para compor o conjunto de dados de origem. A **Tabela 1** especifica a quantidade e o nome de cada tipo de malware adquirido da Virus Share e o seu valor total.

Tipo	Quantidade
Malware	200
Downloader	200
Miner	200
Ransomware	200
RootKit	200
Spyware	200
Total	1200

Tabela 1 - Número de malwares

3.2. Disassembly de binários para extrair funções

Em posse do conjunto de malwares, foi escrito um algoritmo na linguagem C++ para realizar seu tratamento, utilizando a biblioteca libpe (https://github.com/merces/libpe) como base.

Um programa escrito em C++ foi utilizado para extrair as funções importadas pelos malwares, que estão no formato binários. Essas funções são utilizadas para realizar chamadas por recursos que pertencem ao sistema operacional Windows. O programa extraí essas funções e depois as exporta em um arquivo no formato CSV. O arquivo possui rês colunas: as funções, o tipo e o hash do binário. Veja a Figura 5.

	A	B	C
1	function	type	hash
	CM_Add_Empty_Log_Conf CMP_Init_Detection CM_Add_Range		
	CM_Add_IDA CMP_Report_LogOn CryptMsgGetParam		
	CertFreeCTLContext CertGetStoreProperty CertCreateCRLContext		
	CertGetNameStringA CertOpenStore CertFindCTLInStore CryptMsgUpdate		
	CertEnumSystemStore CertDeleteCTLFromStore CryptProtectData		
	CertOIDToAlgId CertControlStore CoCreateActivity		
	CoEnterServiceDomain CoLoadServices SafeRef RecycleSurrogate		
	UrlGetLocationA PathIsPrefixW UrlIsW PathIsRootA UrlCreateFromPathA		
	SHDeleteKeyA UrlGetPartW PathCommonPrefixA PathCompactPathW		
	UrlIsNoHistoryW UrlCanonicalizeW UrlCompareA UrlEscapeW		
	MessageBoxA SetFocus InsertMenuW FindWindowW GetMessageA		
	DispatchMessageA GetDlgItemTextW DialogBoxParamW LoadMenuW		
	CharToOemA CreateDesktopW DrawStateA IsDialogMessageA		
	PeekMessageA WriteFile LoadLibraryExA GetCommandLineA		
	OpenSemaphoreW GetLogicalDriveStringsW lstrcpy LeaveCriticalSection		
	WaitForSingleObject CreateMutexA GetACP GetModuleHandleA		
	GetStringTypeW GetConsoleAliasW lstrcmp Heap32First CreateFileA		
2	GetOEMCP GetProcAddress	ransomware	00e829b2519f6506b3ddf8bc5eb5af7601018b99ccf362dc5bf25d651045a9c3

Figura 5 - Campos do arquivo CSV gerado pelo programa C++

Após feito o procedimento de extração em todos os binários que compõe a base dados, os arquivos CSV extraídos foram mesclados em somente um.

3.3. Análise léxica

Toda a estrutura utilizada no processo de análise foi escrita na linguagem Python, incluindo a parte da análise léxica. O primeiro passo para realizar as analises foi importar as bibliotecas utilizadas. As principais foram: keras, numpy, sklearn e gensim, todas voltadas para manipulação de dados e técnicas de machine learning e deep learning.

Depois de importadas as bibliotecas, foi importado o arquivo CSV, cujo seu processo de obtenção foi descrito na secção anterior. Como as funções foram importadas da maneira como foram recuperadas do arquivo CSV, primeiramente foi necessário realizar um processo de normalização de dados, convertendo tudo para minúsculo e quebrando a string importada contendo as funções em um array de string, sendo cada elemento do array uma função.

A análise léxica foi feita utilizando algoritmo word2vec, cuja implementação foi importada da biblioteca gensim. O array criado anteriormente foi passado como argumento para a criação do modelo.

A Tabela 2 mostra a descrição dos argumentos passados na criação do modelo.

Parâmetro	Descrição	Valor
vector_size	Dimensionality of the word vectors.	100
min_count	Ignores all words with total frequency lower than this.	10
Workers	Use these many worker threads to train the model.	8

Tabela 2- Parâmetro do modelo word2vec

O resultado do modelo word2vec é semelhante a um dicionário, constituído por chaves e valores. Também tem um método que devolve o valor numérico correspondente a qualquer parâmetro de cadeia de caracteres.

Um dos objetivos da análise léxica é possibilitar a conversão de bases de dados em arrays de caracteres números, que são exigidos para criação de modelos como o LSTM. Para isso foi criado uma função que realiza esse processo com o modelo criado pelo word2vec. A Figura 6 mostra o antes e o depois do processo.

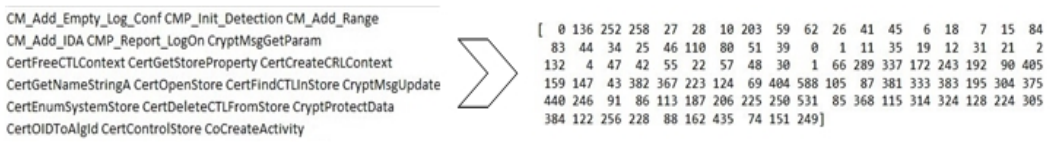

Figura 6 - Array de funções para array de números

3.4. Criação e utilização de modelos LSTM

O resultado da análise léxica é utilizado como entrada na análise utilizando redes neurais através do LSTM. O passo seguinte da análise foi dividir os dados a serem utilizados com treinamento e como validação, sendo 15% do total para testes de validação.

A parte do algoritmo que constrói o modelo LSTM foi desenvolvida utilizando a biblioteca Keras, criada em 2015 como software livre e depois adoptada como interface pela equipa Tensorflow da Google.

O Keras requer um conjunto estruturado de dados numa matriz bidimensional. A primeira dimensão da matriz contém uma lista de números da secção, denotada X. A segunda dimensão representa o tipo de malware, com valores que variam de 0 a 5 conforme definido na Tabela 3. Esta dimensão é designada por Y.

Id	Malware
0	Downloader
1	keylogger
2	Miner
3	Ransomware
4	RootKit
5	Spyware

Tabela 3 - Lista de labels e valores

Para utilizar o modelo LSTM nos padrões necessários em conformidade com a biblioteca Keras, utilizamos a criação prévia de uma camada de incorporação que é dimensionada a partir dos dados passados como argumentos de entrada. Para isso precisamos de extrair informações específicas do modelo word2vec, como o tamanho do vocabulário (que se refere ao número de funções diferentes extraídas do binário), e o tamanho do vocabulário.

Após a recuperação da dimensionalidade e das características dos resultados da análise léxica através da ferramenta word2vec, foram utilizados esses dados para o treinamento de um modelo de rede neural LSTM. O modelo foi dividido em camadas conforme apresentado na Figura 7.

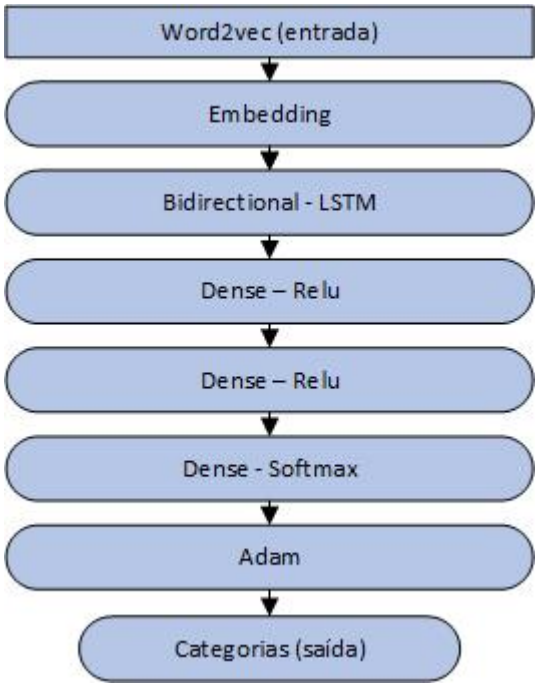

Figura 7 - Camadas do modelo LSTM

Primeiramente foi realizado uma instancia de modelo keras que trabalha de forma sequencial, sendo a instancia a estrutura que vai suportar as entradas e saídas do modelo, que são feitas de forma sequencial. Em seguida, como primeira camada do modelo, foi adicionada uma camada de Embedding, responsável por receber os parâmetros de entrada (provenientes do resultado do word2vec) e remover qualquer tipo de valor negativo da base de dados, necessitando parâmetros de tamanho e dimensionalidade, para funcionar corretamente. Veja esses parâmetros a seguir:

```
model = Sequential()

model.add(Embedding(input_dim=vocab_size,
            output_dim=embedding_size,
            weights=[w2v_weights],
            input_length=MAX_SEQUENCE_LENGTH,
            mask_zero=True,
            trainable=False))
```

Os valores do parâmetros utilizados ao se o se adicionar o Embedding são mostrados na Tabela 4.

Parâmetro	Descrição	Valor
input_dim	Size of the vocabular.	1406
output_dim	Dimension of the dense embedding.	100
weights	Optional floating-point matrix of size.	100
input_length	Size of the input.	100
mask_zero	whether or not the input value 0 is a special "padding" value that should be masked out.	True
trainable	Make the model train by itself, ignoring the inputs metrics.	False

Tabela 4 - Descrição de parametros para o Embedding

Logo após instanciar o Embedding foi adicionado uma camada Bidirectional que passa como argumento o modelo LSTM, que vai atuar como principal responsável por identificar os padrões nos arrays numéricos passados como argumento. Foi escolhida uma estrutura bidirecional no lugar de uma unidirecional, por que a unidirecional apenas preserva os dados passados anteriormente, enquanto que a bidirecional é executada nas duas direções simultaneamente, levando em consideração o passado e o futuro.

Após a camada LSTM, foram adicionadas duas camadas ocultas com função de ativação “ReLU”. ReLU é uma função de ativação simples e amplamente usada que calcula a saída como ReLU(x)=max(0,x).

Em seguida há outra camada oculta com função de penalização regularizadora para ajustar a camada de saída. A função softmax, frequentemente usada na camada final de um modelo de rede neural para tarefas de classificação, converte pontuações de saída brutas (também conhecidas como logits) em probabilidades, tomando o exponencial de cada saída e normalizando esses valores dividindo pela soma de todos os exponenciais. Esse processo garante que os valores de saída estejam no intervalo (0,1) e somem 1, tornando-os interpretáveis como probabilidades. Veja a seguir sua implementação:

Uma vez que todos para parâmetros foram setados, e as camadas do modelo foram adicionadas, é hora de treinar o modelo. Esse processo foi feito com o uso de análise de entropia cruzada categórica como função de perda, cujo a qual é frequentemente utilizada em problemas de multiclassificação. O optimizador selecionado foi o Adam, um método de descida de gradiente estocástico baseado na estimativa adaptativa, que é amplamente utilizado neste tipo de análise. E a acurácia foi utilizada como métrica para medir a taxa de sucesso do modelo.

4. Resultados

Esta secção está dividida em cinco partes: os resultados do modelo treinado, a estrutura de automatização do método proposto, um teste prático com amostras reais como entrada para o modelo, uma comparação com trabalhos relacionados e a lista das ferramentas utilizadas e respectivas versões

4.1. Resultados do modelo treinado

A principal métrica utilizada para medir o modelo treinado foi a acurácia. O processo de treino atingiu uma acurácia de 91%, e a taxa de perda, obtida a partir do cálculo da entropia cruzada de multiclassificação, foi de cerca de 0,3.

A Figura 8 mostra a taxa de acurácia por época.

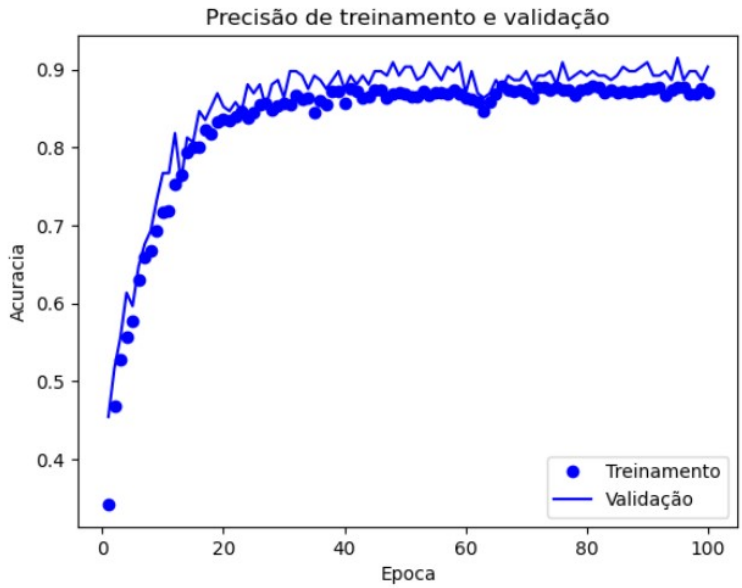

Figura 8 - Acurácia do modelo

A Figura 9 mostra a taxa de perda por época.

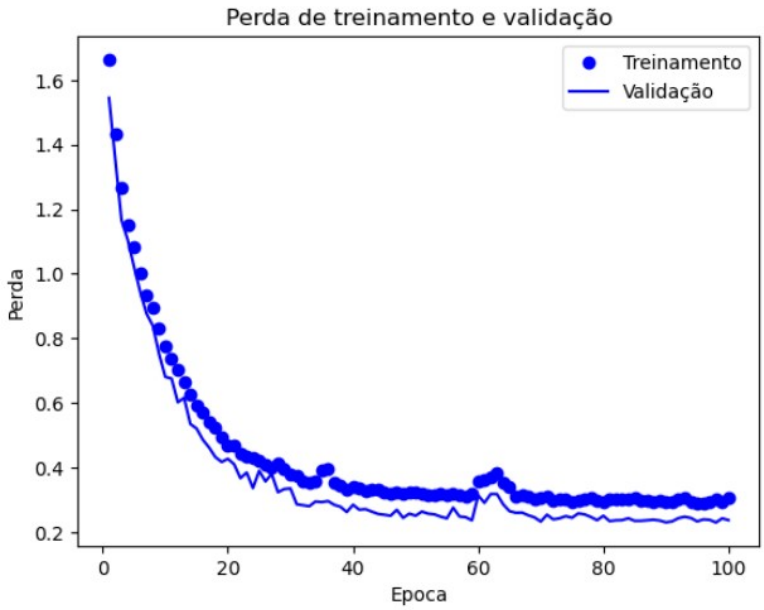

Figura 9 - Perda do modelo

Os resultados obtidos pelo treino dos modelos foram muito próximos dos resultados esperados pelos validadores, como se pode ver na Figura 6 e a comparação das perdas do modelo treinado e da sua validação na Figura 8.

A Figura 10 apresenta a matriz de confusão do modelo treinado. As classes baseiam-se na informação da Tabela 3, e mostra o número de acertos e erros tendo em conta o número de classes em que o conjunto de dados se baseou.

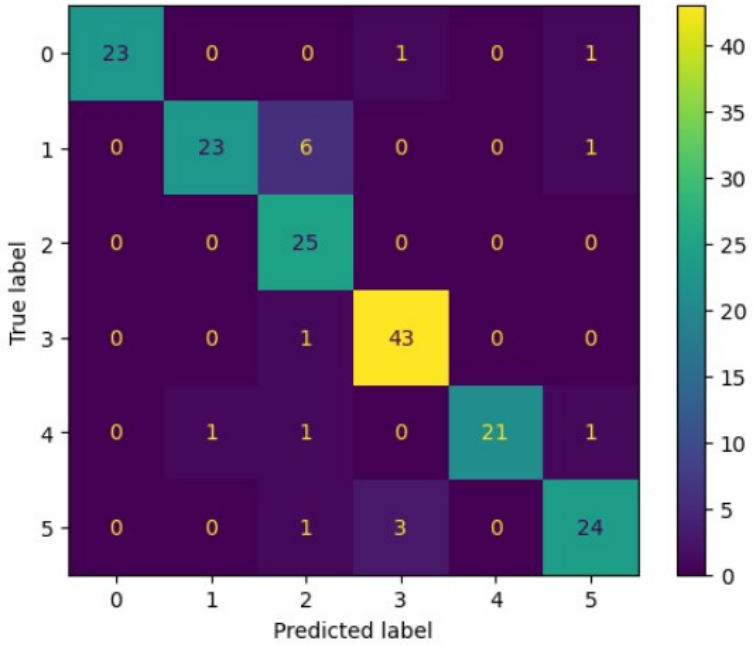

Figura 10 - Matriz de confusão

O resultado apresentado pela matriz de confusão é uma escala de 0 a 5 nos eixos X e Y, representando os valores de malware especificados na Tabela 3. O valor dos blocos varia de 0 a 45, o que significa o número de vezes que um malware foi previsto como pertencente à classe prevista ou a uma outra. Quanto maior o valor, maior a taxa probabilista de uma previsão correta utilizando um conjunto de dados semelhante. Por exemplo, com X=0 e Y=0, o malware foi previsto como sendo um Downloader, o número de acertos é 23, em comparação a valores das linhas e colunas é muito superior, sendo então uma excelente taxa de acerto. Se compararmos com os valores obtidos quando X=2 e Y=2, um Miner, vemos que número obtido é maior (25), porém o número de erros também aumenta consideravelmente, nos levando a conclusão de que se o malware enviado para categorização for desse tipo, a porcentagem de ser categorizado errado aumenta consideravelmente.

Após a análise dos dados, pode concluir-se que os Ransomware e o spyware têm maior probabilidade de serem detectados em comparação com um conjunto de dados semelhante. Em contrapartida, os Miners e os rootkits têm a taxa de acerto mais baixa.

4.2. Estrutura de automação

Para realizar o processo validação primeiramente é feito a ação de disassembly necessário pra extrair as funções inerentes aos arquivos binários no formato PE32, foi utilizado a ferramenta objdump. A estrutura carrega os 2 modelos (Word2vec e LSTM) realiza os processos de analise e retorna o tipo de malware.

Uma vez que o binário recebido passa pelo processo previamente analisado, a aplicação responde com a categoria que mais adequa aos padrões identificados utilizando o formato de REST API.

4.3. Teste prático com amostras reais

Um conjunto de amostras de malware não utilizado no treino e nem no teste foi enviado para ser analisado, através da API REST, visando testar o processo de categorização automatizado, proposto pela automação. Os detalhes do malware como hash, resultado esperado e resultado obtido podem ser encontrados na Tabela 5.

Hash	Esperado	Obtido
fe93d59439b3ac1df36215872d6daf16c68a38eba8bb05b892108d0a13ac4d7b	downloader	miner
c199782b8aa80fa9f4bed0d2fa0b5befdf8df070feb2acf2b967ae8752e04cab	keylogger	keylogger
ff6b9d9786c4505bd80d5156a083648e16ec4a821d89d0fe1687080f3bed60be	miner	miner
ffc338c0c7be278245747c53266b86517c3fd3ccff8a243d9537ffc0afeb9d2c	ransomware	Ransomware
000b2d3b52831632a410ec905de2e97499f4f0fa867efdcab31194f3952b1d6f	rootkit	Rootkit
f43a19ef1e0e2d0b727f34629140e50f35522269e8e094729a65a15c6188965f	spyware	spyware

Tabela 5 - Resultados previstos e obtidos da análise

A taxa média de acerto foi de 83% por cento neste pequeno teste com dados exatos. Este resultado é considerado razoável em termos de acurácia, mas é essencial ter em conta que a ferramenta foi criada com o objetivo demonstrativo e não está pronta para produção.

4.4. Comparação de trabalhos relacionados

Visweswaran et al. [28] analisaram arquivos PE32 utilizando aprendizagem profunda, que é o trabalho mais próximo dos resultados desta investigação. O seu modelo alcançou uma taxa de acurácia de 95%, enquanto o nosso modelo alcançou 91% nos primeiros testes. Utilizaram CNN, enquanto este trabalho utilizou LSTM (uma versão melhorada), e o seu conjunto de dados era mais

significativo do que o incorporado neste trabalho, o que teve um impacto no mapeamento de padrões. No entanto, este trabalho utilizou o word2vec, o que significa que tem o mesmo dicionário que o utilizado para criar o modelo. Esta abordagem permite a qualquer interessado utilizar o mesmo processo para verificar os padrões de um novo binário analisado, ao passo que o trabalho de Visweswaran et al. [28] necessita de reconstruir todo o processo para verificar um padrão binário. Por conseguinte, é difícil tornar possível um processo espontâneo e automatizado. A tabela 6 mostra uma panorâmica geral.

-	Proposta atual	Trabalho de Visweswaran
Algoritmo	LSTM	CNN
Classificação	Multi-class	Multi-class
Acurácia	91%	96%
Quantidade de dados	1200	10000
Epoch	100	100
String para número	Word2vec	Tokenizer
Reproduzível	Yes	No

Tabela 6 - Tabela de comparação

4.5. Ferramentas e versões

As ferramentas e versões utilizadas para construir o modelo LSTM, a estrutura de automação e a ferramenta de desmontagem estão listadas na Tabela 7.

Modelo LSTM	Estrutura de automação	Ferramenta de disassembly
Python 3.10	Docker 20.10.21	VS 2019
TensorFlow 2.12.0	Python 3.10	MSVC 14.38.33130
Keras 2.9.0	Tensorflow 2.15	
Gensim 4.1.2	Keras 2.15.0	
Jupyter notebook 6.4.12	Django 5.0.3	
	Objdump 2.4	
	Gensim 4.3.2	

Tabela 7 - Ferramentas e versões

4.6. Contribuições

Umas das pesquisas que motivaram esse trabalho foi de Visweswaran et al. [28], que utiliza CNN para realizar a classificação de malwares de binários PE32 a partir de funções que são

obrigatoriamente importadas pelos binários. A técnica utilizada nessa pesquisa está desatualizada, uma vez que no âmbito academio o LSTM vem substituindo o CNN nas pesquisas, e por sua vez o Bi-LSTM vem substituindo o LSTM. Wartschinski [27] por sua utiliza LSTM junto com word2vec para analisar tokens extraídos de linguagens de programação, que dentre os tokens extraídos alguns são funções.

O objetivo desse trabalho foi utilizar a técnica proposta por Wartschinski [27] no experimento proposto por Visweswaran et al. [28], e atualizar de LSTM para Bi-LSTM. Os resultados da pesquisa são considerados bom, sendo promissor. Porém a pesquisa foi limitada pelo dataset utilizado, sendo de 1.200 exemplos de malware, enquanto que na pesquisa feita por Visweswaran [27] et al. [28] foi de 10000. Com base na literatura sobre disponível seria possível em trabalhos futuros aumentar a taxa de acurácia do modelo do modelo proposto com um dataset maior.

Além da melhorar a técnica e utiliza-la em outro contexto, um outro objetivo da pesquisa também obteve êxito,ou seja, reproduzir a técnica utilizando os modelos gerados. Isso resultou em uma ferramenta demonstrativa que pode ser usada com dados reais, validando a ideia e auxiliando no processo de metrificação do uso da ferramenta.

5. Conclusão e trabalhos futuros

Com base nas caraterísticas do método proposto, que tem como objetivo acelerar a deteção de malware na estrutura PE32, é possível: a) automatizar as técnicas de desmontagem e utilizar os seus resultados como entrada para automatizar a categorização de malware; b) considerar a análise léxica com word2vec como uma técnica promissora com os seus parâmetros para realizar a análise de malware ML; c) compreender que a análise de aprendizagem profunda com LSTM obteve resultados significativos na categorização de malware.

Como os resultados da aplicação consideram condutas seguras, a estrutura de análise automatizada recém-construída funciona como uma plataforma de validação, permitindo a combinação e a integração com soluções de categorização de malware.

A infraestrutura desenvolvida também tem potencial para suportar a análise em tempo real de arquivos na estrutura PE32 em cenários reais, ajudar em pipelines complexos destinados a automatizar o processo de implementação e pode ser muito bem utilizada em DevSecOps. Os resultados primários tiveram uma taxa razoável de sucesso, mostrando que a análise proposta é promissora.

Como trabalho futuro, o próximo passo seria alargar o conjunto de dados existente para melhorar a acurácia do modelo. Além disso, é necessário criar um novo ramo que primeiro categorize os binários PE32 como maliciosos ou não maliciosos utilizando um classificador binário e depois utilize o modelo atual para categorizar o seu tipo. Outro caminho possível é criar uma estrutura de análise automatizada a partir do novo modelo de categorização e aplicá-la em situações do mundo real, como condutas de verificação, para a integrar com a DAST e a análise de sandbox.

Referências

1. M. Egele, T. Scholte, E. Kirda, and C. Kruegel, “A survey on automa- ted dynamic malware-analysis techniques and tools,” *ACM computing surveys (CSUR)*, vol. 44, no. 2, pp. 1–42, 2008.

2. F. Pagani, M. Dell’Amico, and D. Balzarotti, “Beyond precision and recall: understanding uses (and misuses) of similarity hashes in binary analysis,” in *Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy*, 2018, pp. 354–365.

3. K. Yakdan, S. Dechand, E. Gerhards-Padilla, and M. Smith, “Helping johnny to analyze malware: A usability-optimized decompiler and malware analysis user study,” in *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2016, pp. 158–177.

4. S. Iqbal, A. Ullah, S. Adlan, and A. R. Soobhany, “Malware prediction using lstm networks,” in *Proceedings of International Conference on Information Technology and Applications: ICITA 2021*. Springer, 2022, pp. 583–604.

5. J. A. Morales, A. Al-Bataineh, S. Xu, and R. Sandhu, “Analyzing and exploiting network behaviors of malware,” in *Security and Privacy in Communication Networks: 6th Iternational ICST Conference, Secure- Comm 2010, Singapore, September 7-9, 2010. Proceedings 6*. Springer, 2010, pp. 20–34.

6. M. Pietrek, “An in-depth look into the win32 portable executable file format, part 2,” *MSDN Magazine, March*, 2002.

7. G. D. Webster, B. Kolosnjaji, C. von Pentz, J. Kirsch, Z. D. Hanif, A. Zarras, and C. Eckert, “Finding the needle: A study of the pe32 rich header and respective malware triage,” in *Detection of Intrusions and Malware, and Vulnerability Assessment: 14th International Confe- rence, DIMVA 2017, Bonn, Germany, July 6-7, 2017, Proceedings 14*. Springer, 2017, pp. 119–138.

8. R. Tahir, “A study on malware and malware detection techniques,” *International Journal of Education and Management Engineering*, vol. 8, no. 2, p. 20, 2018.

9. C. Rossow, C. Dietrich, and H. Bos, “Large-scale analysis of malware downloaders,” in *Detection of Intrusions and Malware, and Vulnerabi- lity Assessment: 9th International Conference, DIMVA 2012, Heraklion, Crete, Greece, July 26-27, 2012, Revised Selected Papers 9*. Springer, 2013, pp. 42–61.

10. A. Singh, P. Choudhary *et al.*, “Keylogger detection and prevention,” in *Journal of Physics: Conference Series*, vol. 2007, no. 1. IOP Publishing, 2021, p. 012005.

11. Z. Salehi, M. Ghiasi, and A. Sami, “A miner for malware detection based on api function calls and their arguments,” in *The 16th CSI international symposium on artificial intelligence and signal processing (AISP 2012)*. IEEE, 2012, pp. 563–568.

12. S. Kim, J. Park, K. Lee, I. You, and K. Yim, “A brief survey on rootkit techniques in malicious codes.” *J. Internet Serv. Inf. Secur.*, vol. 2, no. 3/4, pp. 134–147, 2012.

13. M. Egele, C. Kruegel, E. Kirda, H. Yin, and D. Song, “Dynamic spy- ware analysis,” in *Proceedings of the 2007 USENIX Annual Technical Conference*. Carnegie Mellon University, 2007.

14. A. Handler, *An empirical study of semantic similarity in WordNet and Word2Vec*. Columbia University, 2014.

15. T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, “Distri- buted representations of words and phrases and their compositionality,” *Advances in neural information processing systems*, vol. 26, 2013.

16. X. Rong, “word2vec parameter learning explained,” *arXiv preprint arXiv:1411.2738*, 2014.

17. J. Heaton, “Ian goodfellow, yoshua bengio, and aaron courville: Deep learning: The mit press, 2016, 800 pp, isbn: 0262035618,” *Genetic programming and evolvable machines*, vol. 19, no. 1-2, pp. 305–307, 2018.

18. Draper, D., 1995. Assessment and propagation of model uncertainty. Journal of the Royal Statistical Society Series B: Statistical Methodology, 57(1), pp.45-70.

19. Murphy, K.P., 2012. *Machine learning: a probabilistic perspective*. MIT press.

20. Braga-Neto, U.M. and Dougherty, E.R., 2020. Machine Learning Requires Probability and Statistics [Perspectives]. *IEEE Signal Processing Magazine*, *37*(4), pp.118-122.

21. Kelleher, J.D., 2019. *Deep learning*. MIT press.

22. N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Sa- lakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

23. S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

24. F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: Continual prediction with lstm," *Neural computation*, vol. 12, no. 10, pp. 2451–2471, 2000.

25. Y. Dai, H. Li, Y. Qian, and X. Lu, "A malware classification method based on memory dump grayscale image," *Digital Investigation*, vol. 27, pp. 30–37, 2018.

26. Y. J. Lee, S.-H. Choi, C. Kim, S.-H. Lim, and K.-W. Park, "Learning binary code with deep learning to detect software weakness," in *KSII the 9th international conference on internet (ICONI) 2017 symposium*, 2017.

27. L. Wartschinski, "Detecting software vulnerabilities with deep lear- ning," Ph.D. dissertation, Master's thesis, Humboldt University, Berlin, 2014.

28. N. Visweswaran, M. Jeevanantham, C. T. Kani, P. Deepalakshmi, and S. Sathiyandrakumar, "Automated pe32 threat classification using import table and deep neural networks," in *2019 IEEE International Conference on Clean Energy and Energy Efficient Electronics Circuit for Sustainable Development (INCCES)*. IEEE, 2019, pp. 1–5.