

UNIVERSIDAD COMPLUTENSE DE MADRID  
FACULTAD DE CIENCIAS MATEMÁTICAS

DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN



TRABAJO DE FIN DE GRADO

# Algoritmos de Aprendizaje Automático aplicados a problemas de Ciberseguridad

Presentado por: Pablo Jiménez Poyatos

Dirigido por: Luis Fernando Llana Diaz

Grado en Matemáticas

Curso académico 2023-24

## Agradecimientos

**Resumen**

***Palabras clave:***

**Abstract**

***Keywords:***



# Índice general

<b>1. Introducción</b>	<b>1</b>
1.1. Motivación y objetivos del trabajo . . . . .	1
1.2. Contexto y antecedentes del trabajo . . . . .	1
1.2.1. Redes neuronales . . . . .	1
1.2.2. Importancia de la detección y prevención de ataques . . . . .	1
1.2.3. Evolución de las amenazas cibernéticas . . . . .	1
1.2.4. Avances en el aprendizaje automático para ciberseguridad . . . . .	1
1.3. Estructura de la memoria . . . . .	1
1.4. Contribuciones . . . . .	1
<b>2. Fundamentos de las redes neuronales</b>	<b>3</b>
2.1. Revisión teórica . . . . .	3
2.2. Arquitecturas relevantes . . . . .	3
2.2.1. Autoencoder . . . . .	3
2.2.2. Deep Belief Networks . . . . .	4
2.2.3. Red Neuronal Convolutiva . . . . .	4
2.2.4. Red Neuronal Recurrente . . . . .	4
2.3. Implementación en Python . . . . .	4
2.3.1. Frameworks . . . . .	4
2.4. Principales Deep Learning Frameworks. Keras. . . . .	4
<b>3. Aplicación en la ciberseguridad</b>	<b>7</b>
3.1. Clasificación de Malware . . . . .	7

3.1.1. Microsoft Malware Classification Challenge . . . . .	7
3.1.2. Autoencoder . . . . .	11
3.1.3. Red Neuronal Convolutacional . . . . .	11
3.1.4. Resultados . . . . .	11
3.2. Detección de intrusiones . . . . .	11
3.2.1. KDD Cup 1999 . . . . .	11
3.2.2. Autoencoder . . . . .	11
3.2.3. Red Neuronal Convolutacional . . . . .	11
3.2.4. Red Neuronal Profunda . . . . .	12
3.2.5. Red Neuronal Recurrente . . . . .	12
3.2.6. Restricted Boltzmann Machine . . . . .	12
3.2.7. Resultados . . . . .	12
<b>4. Conclusiones y Trabajo Futuro</b>	<b>13</b>
4.1. Conclusiones . . . . .	13
4.2. Trabajo futuro . . . . .	13
<b>Bibliografía</b>	<b>15</b>

# Capítulo 1

## Introducción

### 1.1. Motivación y objetivos del trabajo

### 1.2. Contexto y antecedentes del trabajo

#### 1.2.1. Redes neuronales

#### 1.2.2. Importancia de la detección y prevención de ataques

Destaca la importancia crítica de la detección y prevención de ataques cibernéticos en entornos empresariales y gubernamentales, así como en la protección de datos sensibles y la infraestructura crítica.

#### 1.2.3. Evolución de las amenazas cibernéticas

Describe brevemente cómo han evolucionado las amenazas en el ámbito de la ciberseguridad a lo largo del tiempo, desde virus simples hasta ataques sofisticados como el ransomware y el phishing.

#### 1.2.4. Avances en el aprendizaje automático para ciberseguridad

Proporciona una visión general de cómo los algoritmos de aprendizaje automático han revolucionado el campo de la ciberseguridad, permitiendo la detección temprana de amenazas, el análisis de comportamiento anómalo y la automatización de respuestas.

### 1.3. Estructura de la memoria

### 1.4. Contribuciones





## Capítulo 2

# Fundamentos de las redes neuronales

### 2.1. Revisión teórica

Puedo introducir los tipos de funciones de activación. Está bien explicado en el TFG wuolah o en el artículo de KDD cup 199 de DNN network intrusion. Puedo añadir overfitting y underfitting. lo que es aprendizaje supervisado y no supervisado Partes de una neurona y cómo trabaja (bias, pesos...)

### 2.2. Arquitecturas relevantes

Mini tabla resumen en Deep Cybersecurity: A Comprehensive Overview from Neural Network and Deep Learning Perspective y miniresumen de todos los tipos en review Deep Cybersecurity: A Comprehensive Overview from Neural Network and Deep Learning Perspective y review

#### 2.2.1. Autoencoder

Leer y sacar la información del word autoencoders.

Los autoencoders son una clase de redes neuronales artificiales utilizadas en aprendizaje no supervisado para aprender representaciones eficientes de datos. Su objetivo principal es codificar la entrada en una representación comprimida y significativa, y luego decodificarla de manera que la reconstrucción sea lo más similar posible a la entrada original.

La arquitectura básica de un autoencoder consta de dos partes: el encoder y el decoder. El encoder mapea los datos de entrada a una representación oculta de menor dimensión utilizando funciones principalmente no lineales, mientras que el decoder reconstruye los datos de entrada a partir de esta representación oculta. Durante el entrenamiento, los parámetros del autoencoder se optimizan para minimizar la diferencia entre la entrada y la salida reconstruida, utilizando una función de pérdida que mide esta discrepancia.

Los autoencoders se han utilizado en una amplia variedad de aplicaciones, incluida la reducción de dimensionalidad, la extracción de características, la eliminación de ruido en los datos de entrada y la detección de anomalías. Su versatilidad y capacidad para aprender representaciones

útiles de los datos los hacen herramientas poderosas en el campo del aprendizaje automático y la inteligencia artificial.

### 2.2.2. Deep Belief Networks

#### Red Neuronal Profunda

### 2.2.3. Red Neuronal Convolutacional

### 2.2.4. Red Neuronal Recurrente

#### Restricted Boltzmann Machine

## 2.3. Implementación en Python

### 2.3.1. Frameworks

numpy matplotlib sklearn

#### Deep neural network

## 2.4. Principales Deep Learning Frameworks. Keras.

A medida que las técnicas de aprendizaje profundo han ido ganando popularidad, muchas organizaciones académicas e industriales se han centrado en desarrollar marcos de trabajo para facilitar la experimentación con redes neuronales profundas de manera sencilla para el usuario. En esta sección, ofrecemos una visión general de las bibliotecas más conocidas: TensorFlow, PyTorch y Keras.

**TensorFlow** [11] es una biblioteca de código abierto que fue desarrollada en 2016 por el equipo de Google Brain del departamento de IA de Google. Es una plataforma que se centra principalmente en el aprendizaje automático y la computación de alto rendimiento. Su modelo de programación permite que los datos fluyan de una operación a otra de manera flexible gracias a su estructura de grafo computacional y al concepto de flujo de datos. Los gráficos son estructuras de datos que contienen un conjunto de objetos `tf.Operation` (nodos), que representan unidades de cálculo y objetos `tf.Tensor`<sup>1</sup>, que representan las unidades de datos que fluyen entre esas operaciones (Figura 2.1). Esto permite una ejecución eficiente y paralela de operaciones en diferentes dispositivos de hardware, como en una CPU, GPU, dispositivos móviles y sistemas distribuidos a

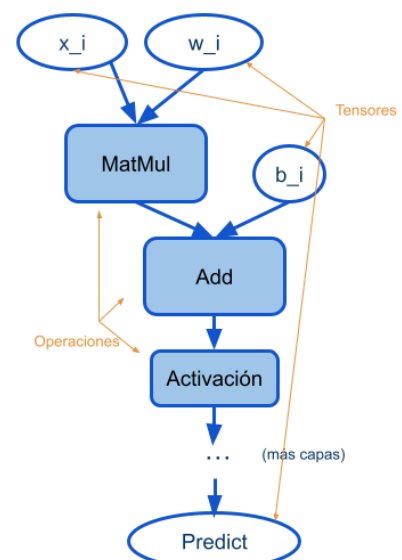


Figura 2.1: Gráfico de una red neuronal.

<sup>1</sup>arreglos multidimensionales que pueden contener datos de cualquier tipo y forma

gran escala con cientos de nodos. Esta flexibilidad y además su capacidad de diferenciación automática lo hacen una biblioteca útil para una gran variedad de aplicaciones, desde la investigación hasta el desarrollo y la producción de modelos de aprendizaje automático.

**PyTorch** [8], presentado por el equipo de investigación en IA de Facebook en 2016, es una biblioteca de aprendizaje profundo desarrollado en Python que simplifica la creación de modelos complejos a través de una interfaz de programación sencilla. A diferencia de otros marcos populares que emplean gráficos de computación estáticos, PyTorch se basa en la computación dinámica (su topología puede variar durante la ejecución del programa), lo que permite una mayor flexibilidad para diseñar arquitecturas complejas. Cambiar el comportamiento de una red neuronal típicamente implica reiniciar desde cero, pero PyTorch emplea una técnica llamada auto-diferenciación en modo inverso, que permite realizar cambios en el comportamiento de la red con poco esfuerzo. PyTorch se ha vuelto popular tanto en la comunidad científica como en la industria debido a su facilidad de uso y su capacidad para crear modelos complejos de manera eficiente. Además, ha sido adoptado por varias organizaciones importantes, incluidas Facebook, Twitter y NVIDIA, lo que garantiza su continuo desarrollo y soporte.

**Keras** [10] es un marco de redes neuronales de código abierto desarrollado por François Chollet, un miembro del equipo de IA de Google. Se considera un meta-marco de trabajo que interactúa con otros marcos. En particular puede ejecutarse en la parte superior de TensorFlow y Theano. Está implementado en Python y proporciona APIs de redes neuronales de alto nivel para desarrollar modelos de aprendizaje profundo. En lugar de manejar operaciones de bajo nivel (diferenciación y manipulación de tensores), Keras depende de una biblioteca especializada que sirve como su motor de backend. Keras minimiza el número de acciones requeridas por un usuario para una acción específica. Una característica importante de Keras es su facilidad de uso sin sacrificar la flexibilidad. Keras permite a los usuarios implementar sus modelos como si estuvieran implementados en los marcos base (como TensorFlow o Theano<sup>2</sup>). Proporciona un rendimiento y escalabilidad de nivel industrial, siendo utilizado por organizaciones de renombre como NASA, YouTube y Waymo para una amplia gama de aplicaciones en inteligencia artificial y aprendizaje automático.

Analizando los resultados de [20], podemos observar que usando la CPU, Keras destaca por encima de las demás. No solo logra el mejor accuracy en los tres datasets (MNIST, CIFAR-10, CIFAR-100), sino que además también tiene los tiempos de ejecución más bajos y una de las mejores tasas de convergencia. Estos datos sobresalientes además de su facilidad de uso, accesibilidad y documentación bien estructurada han sido determinantes para acabar por decantarme por Keras. Según Aakash Nain [2]

Keras is that sweet spot where you get flexibility for research and consistency for deployment. Keras is to Deep Learning what Ubuntu is to Operating Systems.

y según Matthew Carrigan [22]

The best thing you can say about any software library is that the abstractions it chooses feel completely natural, such that there is zero friction between thinking about what you want to do and thinking about how you want to code it. That's exactly what you get with Keras.

---

<sup>2</sup>Librería de Python para aprendizaje automático.



## Capítulo 3

# Aplicación en la ciberseguridad

### 3.1. Clasificación de Malware

Hoy en día, uno de los principales retos que enfrenta el software anti-malware es la enorme cantidad de datos y archivos que se requieren evaluar en busca de posibles amenazas maliciosas. Una de las razones principales de este volumen tan elevado de archivos diferentes es que los creadores de malware introducen variaciones en los componentes maliciosos para evadir la detección. Esto implica que los archivos maliciosos pertenecientes a la misma “familia” de malware (con patrones de comportamiento similares), se modifican constantemente utilizando diversas tácticas, lo que hace que parezcan ser múltiples archivos distintos.

Para poder analizar y clasificar eficazmente estas cantidades masivas de archivos, es necesario agruparlos e identificar sus respectivas familias. Además, estos criterios de agrupación pueden aplicarse a nuevos archivos encontrados en computadoras para detectarlos como maliciosos y asociarlos a una familia específica.

#### 3.1.1. Microsoft Malware Classification Challenge

El conjunto de datos utilizado en este estudio proviene del Microsoft Malware Classification Challenge (BIG 2015) [1], una competición dirigida a la comunidad científica con el objetivo de promover el desarrollo de técnicas efectivas para agrupar diferentes variantes de malware. Se decidió escoger este dataset porque el objetivo que tengo en este trabajo es el de aprender y desarrollar diferentes métodos de aprendizaje automático y este dataset nos permite utilizar tanto una CNN como un Autoencoder según [26].

Se puede descargar desde su página web [1]. Tiene un tamaño de 0.5 TB sin comprimir. Para poder manipularla en mi ordenador, tuve que seguir los siguientes pasos. Primero, me descargué la carpeta comprimida (7z) con todo el dataset. Después, la subí al servidor Simba de la facultad de informática y finalmente, usando el comando `7zz x file_name.7z`, la descomprimí.

Este dataset contiene 5 archivos:

- dataSample.7z - Carpeta comprimida(7z) con una muestra de los datos disponibles.
- train.7z - Carpeta comprimida(7z) con los datos para el conjunto de entrenamiento.
- trainLabels.csv - Archivo csv con las etiquetas asociadas a cada archivo de train.

- test.7z - Carpeta comprimida 7z con los datos sin procesar para el conjunto de prueba.
- sampleSubmission.csv - Archivo csv con el formato de envío válido de las soluciones.

Para nuestro estudio, nos enfocaremos exclusivamente en el conjunto de datos de entrenamiento, que consta de los archivos “train.7z” y “trainLabels.csv”. Los archivos ‘test.7z’ y ‘sampleSubmission.csv’ están destinados específicamente para la competición. Nosotros no los utilizaremos debido a que son programas de malware sin etiquetar y para este problema de clasificación, es necesario conocerlas. Además, la carpeta ‘dataSample.7z’ proporciona dos programas que se encuentran también en la carpeta train.7z, por lo que tampoco la utilizaremos.

Cada programa malicioso tiene un identificador, un valor hash de 20 caracteres que identifica de forma única el archivo, y una etiqueta de clase, que es un número entero que representa una de las 9 familias de malware al que puede pertenecer. Por ejemplo, el programa *0ACDbR5M3ZhBJajygTuf* tiene como etiqueta el valor 7. Esta información se puede consultar en el archivo “trainLabels.csv”. Cada programa tiene dos archivos, uno `asm`<sup>1</sup> con el código extraído por la herramienta de desensamblado IDA y otro `bytes`<sup>2</sup> con la representación hexadecimal del contenido binario del programa pero sin los encabezados ejecutables (para garantizar esterilidad). Para nuestro estudio vamos a utilizar únicamente este último archivo.

DIRECCIÓN MEMORIA		REPRESENTACIÓN HEXADECIMAL															
28232	0046F470	E0	01	EC	10	4C	01	62	00	EC	00	82	11	06	11	84	01
28233	0046F480	A8	11	00	10	EE	00	AE	01	42	10	20	11	C2	00	A0	10
28234	0046F490	CC	10	4A	01	42	00	EE	01	AA	00	44	00	84	10	0C	01
28235	0046F4A0	24	11	A8	10	AC	01	AE	11	0E	01	80	10	6A	11	6A	10
28236	0046F4B0	4E	01	82	01	00	01	AE	01	0E	11	E2	11	0A	10	2A	01
28237	0046F4C0	60	01	C8	00	E8	10	28	01	04	00	82	00	62	10	E4	01
28238	0046F4D0	EA	00	CE	01	A6	01	46	11	0C	00	00	00	??	??	??	??
28239	0046F4E0	??	??	??	??	??	??	??	??	??	??	??	??	??	??	??	??

Figura 3.1: Explicación del contenido de “0ACDbR5M3ZhBJajygTuf.bytes”.

Como aparece en la figura 3.1, los ocho primeros caracteres son direcciones de memoria, seguido de la representación hexadecimal del contenido binario del programa, que contiene 16 bytes (cada uno dos caracteres). A veces nos podemos encontrar con “??” en el lugar de un byte. Este símbolo se utiliza en estos archivos para representar que se desconoce su información porque su memoria no se puede leer [6].

### Distribución del dataset

Hay un total de 21.741 programas de malware, pero nosotros tan solo usaremos los 10.868 pertenecientes al entrenamiento. Estos programas pertenecen a una de estas 9 familias de malware: Rammit, Lollipop, Kelihos\_ ver3, Vundo, Simda, Tracur, Kelihos\_ ver1, Obfuscator y Gatak. Según [13], podemos deirlas como:

<sup>1</sup>Es una secuencia de bytes que lo podemos visualizar como si fuera un texto plano

<sup>2</sup>Realmente no es un archivo bytes, sino un archivo de texto con letras

1. **Ramnit** es un malware tipo gusano que infecta archivos ejecutables de Windows, archivos de Microsoft Office y archivos HTML. Cuando los ha infectado, el ordenador pasa a formar parte de una red de bots, que son controladas por un nodo central de forma remota. Este malware puede robar información sensible y puede propagarse a través de conexiones de red y unidades extraíbles.

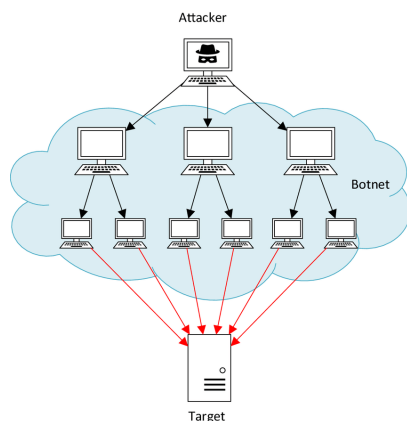


Figura 3.2: Estructura de un botnet. Imagen sacada de [24].

2. **Lollipop** es un tipo de programa adware que muestra anuncios no deseados en los navegadores web. También puede redirigir los resultados de búsqueda a recursos web ilegítimos, descargar aplicaciones maliciosas y robar la información del ordenador monitoreando sus actividades web. Este adware se puede descargar desde el sitio web del programa o empaquetarse con algunos programas de terceros.

3. **Simda** es un troyano backdoor que infecta ordenadores descargando y ejecutando archivos arbitrarios que pueden incluir malware adicional. Los ordenadores infectados pasan a ser parte de una botnet, lo que les permite cometer acciones criminales como robo de contraseñas, credenciales bancarias o descargar otros tipos de malware.

4. **Vundo** es otro troyano conocido por causar publicidad emergente para programas de antivirus falsos. A menudo se distribuye como un archivo DLL (Dynamic Link Library) <sup>3</sup> y se instala en el ordenador como un Objeto Auxiliar del Navegador (BHO) sin su consentimiento. Además, utiliza técnicas

avanzadas para evitar su detección y eliminación.

5. **Kelihos\_ ver3** es un troyano tipo backdoor <sup>4</sup> que distribuye correos electrónicos que pueden contener enlaces falsos a instaladores de malware. Consta de tres tipos de bots [15]: controladores (operados por los dueños y donde se crean las instrucciones), enrutadores (redistribuyen las instrucciones a otros bots) y trabajadores (ejecutan las instrucciones).
6. **Tracur** es un descargador troyano que agrega el proceso 'explorer.exe' a la lista de excepciones del Firewall de Windows para disminuir deliberadamente la seguridad del sistema y permitir la comunicación no autorizada a través del firewall. Además, esta familia también te puede redirigir a enlaces maliciosos para descargar e instalar otros tipos de malware.
7. **Kelihos\_ ver1** es una versión más antigua del troyano Kelihos\_ ver3 pero tiene el mismo funcionamiento y las mismas partes.
8. **Obfuscator.ACY** es un tipo de malware sofisticado que oculta su propósito y podría sobrepasar las capas de seguridad del software. Se puede propagar mediante archivos adjuntos de correo electrónico, anuncios web y descargas de archivos.
9. **Gatak** es un troyano que abre una puerta trasera en el ordenador. Se propaga a través de sitios web falsos que ofrecen claves de licencias de productos. Una vez infectado el sistema, Gatak recopila información del ordenador.

Como ya mencionamos antes, vamos a entrenar nuestras redes neuronales con 10.868 archivos bytes. De estos archivos, solo nos son válidos 10.860 porque en los 8 restantes (pertenecientes a

<sup>3</sup>Una parte del programa que se ejecuta cuando una aplicación se lo pide. Se suele guardar en un directorio del sistema.

<sup>4</sup>Un backdoor permite que una entidad no autorizada tome el control completo del sistema de una víctima sin su consentimiento.

la familia Ramnit), todo su contenido son “?”. Con estos programas finales, vamos a ver sus datos en el gráfico 3.3.

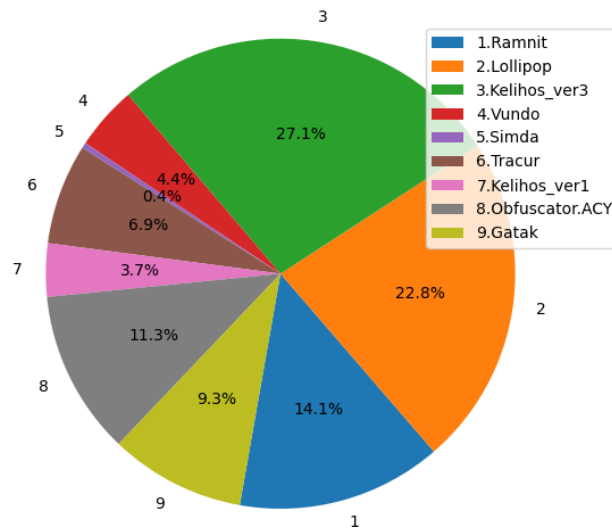


Figura 3.3: Distribución del BIG 2015 training dataset.

Analizando 3.3, podemos observar como la distribución entre las clases no es uniforme. Mientras que de la clase Simbda hay 42 muestras, de la clase Kelihos\_ver3 hay 2942, es decir, un 98 % más. En [14] deciden prescindir de esta clase, pero nosotros hemos decidido hacer el análisis con las 9 clases. A la hora de crear nuestros modelos, hemos dividido el conjunto de datos aleatoriamente en grupos del 75 %, 15 % y 10 % para entrenamiento, test y validación respectivamente. La imagen 3.4 muestra como quedarían distribuidas las clases en los diferentes grupos.

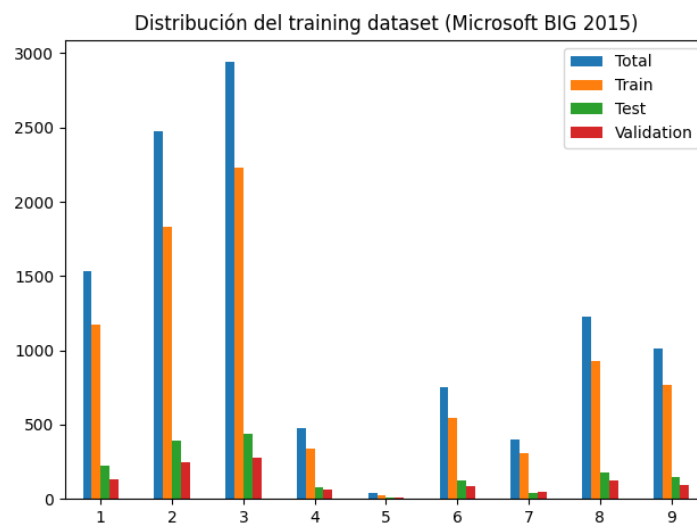


Figura 3.4: Distribución de las clases en cada grupo.

Como podemos observar, todos los programas proporcionados son malware, lo que nos indica que con este dataset no podemos crear un modelo que nos prediga si un programa es malware o no, sino que tan solo podemos hacer un modelo de clasificación. Para ello vamos a abordar este experimento de dos formas diferentes. Una de ellas es utilizando una CNN y la otra es usando un autoencoder. utilizar dos redes neuronales



### 3.1.2. Autoencoder

### 3.1.3. Red Neuronal Convolutacional

### 3.1.4. Resultados

El entorno de hardware en el que he realizado todos los experimentos es un sistema operativo Debian 12.2 con Linux version 6.1.0-17-amd64. La CPU utilizada es un Intel(R) Xeon(R) W-2235 CPU con 3.8 GHz ( $3,80 * 10^9$ Hz) con 6 núcleos. La memoria RAM disponible es de 128 GB.

## 3.2. Detección de intrusiones

### 3.2.1. KDD Cup 1999

### 3.2.2. Autoencoder

Para la clasificación binaria usar autoencoder con el entrenamiento de las imágenes (buenas o malas) y según el error que den, se clasifica. Para la multclasificación, tenemos dos opciones:

- Usamos autoencoder para comprimir la información de entrada y después esa información la usamos para clasificarla usando una DNN [18]
- Usamos una cadena de autoencoders en el cual la salida de  $h$  es la entrada del autoencoder  $h+1$ . Utilizo el artículo [9] donde se desarrolla todo el modelo y explicación y además se hace referencia al artículo [5] porque se basa en él (lo de salida de  $h$  es la entrada de  $h+1$ ). Ver también:
  - Asymmetric Stacked Autoencoder
  - Constrained Nonlinear Control Allocation based on Deep Auto-Encoder Neural Networks.

El algoritmo consiste en entrenar las capas por separado en la que el input del autoencoder es la salida del autoencoder anterior. Lo que de verdad nos interesa es la capa oculta, que tiene una representación comprimida de los datos de entrada y sus pesos. Estos pesos son con los que se inicializa el entrenamiento de la stacked autoencoder acabando en softmax. He usado el url para enterlo <https://amiralavi.com/tied-autoencoders/>. Además en [4] explica bastante bien la diferencia entre capa autoencoder y un autoencoder.

### 3.2.3. Red Neuronal Convolutacional

Para clasificar los datos del dataset KDD 1999 usando las Convolutional Neural Network (CNN) vamos a seguir los siguientes artículos [16, 31, 25, 17]. Prácticamente todo el cuerpo del experimento se encuentra en el artículo [16], pero en el artículo [17] aparece la parte de normalización de los datos y algunos hiperparámetros de inicio.

### 3.2.4. Red Neuronal Profunda

Por otro lado, el método Deep Neural Network (DNN) utiliza una arquitectura muy parecida a una CNN. Podemos ver todo el procesamiento de los datos y el modelo en el artículo [21]. Además, hay buena explicación del experimento en [30]. Por último, en el artículo [7] están los experimentos con DNN, RNN, RBM que puedo tomar también como referencia porque está muy bien explicado las capas e hiperparametros que utiliza.

### 3.2.5. Red Neuronal Recurrente

En el artículo [7] están los experimentos con DNN, RNN, RBM que puedo tomar también como referencia porque está muy bien explicado las capas e hiperparametros que utiliza.

### 3.2.6. Restricted Boltzmann Machine

En el artículo [7] están los experimentos con DNN, RNN, RBM que puedo tomar también como referencia porque está muy bien explicado las capas e hiperparametros que utiliza.

### 3.2.7. Resultados

El entorno de hardware en el que he realizado todos los experimentos es un sistema operativo Debian 12.2 con Linux version 6.1.0-17-amd64. La CPU utilizada es un Intel(R) Xeon(R) W-2235 CPU con 3.8 GHz ( $3,80 * 10^9$ Hz) con 6 núcleos. La memoria RAM disponible es de 128 GB.

## Capítulo 4

# Conclusiones y Trabajo Futuro

### 4.1. Conclusiones

### 4.2. Trabajo futuro



# Bibliografía

- [1] Microsoft malware classification challenge (big 2015), 2015.
- [2] Aakash Nain. Keras Documentation. <https://keras.io/>. Consultado el 06-05-2024.
- [3] Apache Software Foundation. Apache mxnet, 2015.
- [4] Wei Bao, Jun Yue, and Yulei Rao. A deep learning framework for financial time series using stacked autoencoders and long-short term memory. *PloS one*, 12(7):e0180944, 2017.
- [5] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. *Advances in neural information processing systems*, 19, 2006.
- [6] Niken Dwi Wahyu Cahyani, Erwid M Jadied, Nurul Hidayah Ab Rahman, and Endro Ariyanto. The influence of virtual secure mode (vsm) on memory acquisition. *International Journal of Advanced Computer Science and Applications*, 13(11), 2022.
- [7] Wisam Elmasry, Akhan Akbulut, and Abdul Halim Zaim. Empirical study on multiclass classification-based network intrusion detection. *Computational Intelligence*, 35(4):919–954, 2019.
- [8] Facebook AI Research. Pytorch, 2017.
- [9] Fahimeh Farahnakian and Jukka Heikkonen. A deep auto-encoder based approach for intrusion detection system. In *2018 20th International Conference on Advanced Communication Technology (ICACT)*, pages 178–183. IEEE, 2018.
- [10] Google AI Team. Keras, 2015.
- [11] Google Brain Team. Tensorflow, 2015.
- [12] Sanchit Gupta, Harshit Sharma, and Sarvjeet Kaur. Malware characterization using windows api call sequences. In *Security, Privacy, and Applied Cryptography Engineering: 6th International Conference, SPACE 2016, Hyderabad, India, December 14-18, 2016, Proceedings 6*, pages 271–280. Springer, 2016.
- [13] Yen-Hung Frank Hu, Abdinur Ali, Chung-Chu George Hsieh, and Aurelia Williams. Machine learning techniques for classifying malicious api calls and n-grams in kaggle data-set. In *2019 SoutheastCon*, pages 1–8. IEEE, 2019.
- [14] Temesguen Messay Kebede, Ouboti Djaneye-Boundjou, Barath Narayanan Narayanan, Anca Ralescu, and David Kapp. Classification of malware programs using autoencoders based deep learning architecture and its application to the microsoft malware classification challenge (big 2015) dataset. In *2017 IEEE National Aerospace and Electronics Conference (NAECON)*, pages 70–75. IEEE, 2017.

- [15] Max Kerkers, José Jair Santanna, and Anna Sperotto. Characterisation of the keliho. b botnet. In *Monitoring and Securing Virtualized Networks and Services: 8th IFIP WG 6.6 International Conference on Autonomous Infrastructure, Management, and Security, AIMS 2014, Brno, Czech Republic, June 30–July 3, 2014. Proceedings 8*, pages 79–91. Springer, 2014.
- [16] Jiyeon Kim, Jiwon Kim, Hyunjung Kim, Minsun Shim, and Eunjung Choi. Cnn-based network intrusion detection against denial-of-service attacks. *Electronics*, 9(6):916, 2020.
- [17] Taejoon Kim, Sang C Suh, Hyunjoo Kim, Jonghyun Kim, and Jinoh Kim. An encoding technique for cnn-based network anomaly detection. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 2960–2965. IEEE, 2018.
- [18] Ivandro O Lopes, Deqing Zou, Ihsan H Abdulqadder, Francis A Ruambo, Bin Yuan, and Hai Jin. Effective network intrusion detection via representation learning: A denoising autoencoder approach. *Computer Communications*, 194:55–65, 2022.
- [19] Mika Luoma-aho. Analysis of modern malware: obfuscation techniques. 2023.
- [20] Nesma Mahmoud, Youssef Essam, Radwa Elshawi, and Sherif Sakr. Dlbenc: an experimental evaluation of deep learning frameworks. In *2019 IEEE International Congress on Big Data (BigDataCongress)*, pages 149–156. IEEE, 2019.
- [21] Mohammed Maithem and Ghadaa A Al-Sultany. Network intrusion detection system using deep neural networks. In *Journal of Physics: Conference Series*, volume 1804, page 012138. IOP Publishing, 2021.
- [22] Matthew Carrigan. Keras Documentation. <https://keras.io/>. Consultado el 06-05-2024.
- [23] Montreal University. Theano, 2010.
- [24] Mohammad Najafimehr, Sajjad Zarifzadeh, and Seyedakbar Mostafavi. A hybrid machine learning approach for detecting unprecedented ddos attacks. *The Journal of Supercomputing*, 78, 04 2022.
- [25] Sinh-Ngoc Nguyen, Van-Quyet Nguyen, Jintae Choi, and Kyungbaek Kim. Design and implementation of intrusion detection system using convolutional neural network for dos detection. In *Proceedings of the 2nd international conference on machine learning and soft computing*, pages 34–38, 2018.
- [26] Prajoy Podder, Subrato Bharati, M Mondal, Pinto Kumar Paul, and Utku Kose. Artificial neural network for cybersecurity: A comprehensive review. *arXiv preprint arXiv:2107.01185*, 2021.
- [27] Sajedul Talukder. Tools and techniques for malware detection and analysis. *arXiv preprint arXiv:2002.06819*, 2020.
- [28] Mingdong Tang and Quan Qian. Dynamic api call sequence visualisation for malware classification. *IET Information Security*, 13(4):367–377, 2019.
- [29] Tokyo University. Chainer, 2015.
- [30] Rahul K Vigneswaran, R Vinayakumar, KP Soman, and Prabakaran Poornachandran. Evaluating shallow and deep neural networks for network intrusion detection systems in cyber security. In *2018 9th International conference on computing, communication and networking technologies (ICCCNT)*, pages 1–6. IEEE, 2018.

- [31] Zhongxue Yang and Adem Karahoca. An anomaly intrusion detection approach using cellular neural networks. In *Computer and Information Sciences–ISCIS 2006: 21th International Symposium, Istanbul, Turkey, November 1-3, 2006. Proceedings 21*, pages 908–917. Springer, 2006.
- [32] Ilsun You and Kangbin Yim. Malware obfuscation techniques: A brief survey. In *2010 International conference on broadband, wireless computing, communication and applications*, pages 297–300. IEEE, 2010.
- [33] Mohamad Fadli Zolkipli and Aman Jantan. Malware behavior analysis: Learning and understanding current malware threats. In *2010 Second International Conference on Network Applications, Protocols and Services*, pages 218–221. IEEE, 2010.





Hex	Dec	Hex	Dec	Hex	Dec	Hex	Dec	Hex	Dec	Hex	Dec
00	0	2E	46	5C	92	8A	138	B8	184	E6	230
01	1	2F	47	5D	93	8B	139	B9	185	E7	231
02	2	30	48	5E	94	8C	140	BA	186	E8	232
03	3	31	49	5F	95	8D	141	BB	187	E9	233
04	4	32	50	60	96	8E	142	BC	188	EA	234
05	5	33	51	61	97	8F	143	BD	189	EB	235
06	6	34	52	62	98	90	144	BE	190	EC	236
07	7	35	53	63	99	91	145	BF	191	ED	237
08	8	36	54	64	100	92	146	C0	192	EE	238
09	9	37	55	65	101	93	147	C1	193	EF	239
0A	10	38	56	66	102	94	148	C2	194	F0	240
0B	11	39	57	67	103	95	149	C3	195	F1	241
0C	12	3A	58	68	104	96	150	C4	196	F2	242
0D	13	3B	59	69	105	97	151	C5	197	F3	243
0E	14	3C	60	6A	106	98	152	C6	198	F4	244
0F	15	3D	61	6B	107	99	153	C7	199	F5	245
10	16	3E	62	6C	108	9A	154	C8	200	F6	246
11	17	3F	63	6D	109	9B	155	C9	201	F7	247
12	18	40	64	6E	110	9C	156	CA	202	F8	248
13	19	41	65	6F	111	9D	157	CB	203	F9	249
14	20	42	66	70	112	9E	158	CC	204	FA	250
15	21	43	67	71	113	9F	159	CD	205	FB	251
16	22	44	68	72	114	A0	160	CE	206	FC	252
17	23	45	69	73	115	A1	161	CF	207	FD	253
18	24	46	70	74	116	A2	162	D0	208	FE	254
19	25	47	71	75	117	A3	163	D1	209	FF	255
1A	26	48	72	76	118	A4	164	D2	210		
1B	27	49	73	77	119	A5	165	D3	211		
1C	28	4A	74	78	120	A6	166	D4	212		
1D	29	4B	75	79	121	A7	167	D5	213		
1E	30	4C	76	7A	122	A8	168	D6	214		
1F	31	4D	77	7B	123	A9	169	D7	215		
20	32	4E	78	7C	124	AA	170	D8	216		
21	33	4F	79	7D	125	AB	171	D9	217		
22	34	50	80	7E	126	AC	172	DA	218		
23	35	51	81	7F	127	AD	173	DB	219		
24	36	52	82	80	128	AE	174	DC	220		
25	37	53	83	81	129	AF	175	DD	221		
26	38	54	84	82	130	B0	176	DE	222		
27	39	55	85	83	131	B1	177	DF	223		
28	40	56	86	84	132	B2	178	E0	224		
29	41	57	87	85	133	B3	179	E1	225		
2A	42	58	88	86	134	B4	180	E2	226		
2B	43	59	89	87	135	B5	181	E3	227		
2C	44	5A	90	88	136	B6	182	E4	228		
2D	45	5B	91	89	137	B7	183	E5	229		

Cuadro 1: Tabla de códigos hexadecimales y sus equivalentes decimales