Trabajo Fin de Máster

Detección de actividad sospechosa mediante análisis de comportamiento (UEBA)

Jose Fuentes Rodríguez

Máster en Técnicas Estadísticas

Curso 2021-2022

Propuesta de Trabajo Fin de Máster

Título en galego: Detección de actividade sospeitosa mediante análise de comportamento (UEBA)
Título en español: Detección de actividad sospechosa mediante análisis de comportamiento (UEBA)
English title: Detection of suspicious activity through User and Entity Behaviour Analytics
Modalidad: Modalidad B
Autor/a: Jose Fuentes Rodríguez, Universidad de Santiago de Compostela
Director/a: Marta Sestelo Pérez, Universidad de Vigo
Tutor/a: Nora Martínez Villanueva, Gradiant
Breve resumen del trabajo: En este trabajo se exploraron varios algoritmos para la detección de actividad sospechosa mediante análisis de comportamiento (User and Entities Behavioural Analysis, UEBA), bajo la supervisión de la empresa Gradiant. Para ello se utilizaron técnicas de aprendizaje automático y de profundo.
Recomendaciones:
Otras observaciones:

Doña Marta Sestelo Pérez, Profesora Ayudante Doctora de la Universidad de Vigo y doña Nora Martínez Villanueva, Technical Manager Data Analytics and AI de Gradiant, informan que el Trabajo Fin de Máster titulado

Detección de actividad sospechosa mediante análisis de comportamiento (UEBA)

fue realizado bajo su dirección por don Jose Fuentes Rodríguez para el Máster en Técnicas Estadísticas. Estimando que el trabajo está terminado, dan su conformidad para su presentación y defensa ante un tribunal.

En Santiago de Compostela, a 02 de Febrero de 2022.

La directora:

SESTELO PEREZ MARTA - 76998300G

Firmado digitalmente por SESTELO PEREZ MARTA - 76998300G
Fecha: 2022.02.02 18:46:41 +01'00'

Doña Marta Sestelo Pérez

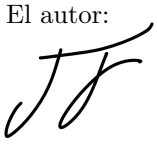
La tutora:

MARTINEZ VILLANUEVA NORA - 53179846M

Firmado digitalmente por MARTINEZ VILLANUEVA NORA - 53179846M
Fecha: 2022.02.02 18:46:57 +01'00'

Doña Nora Martínez Villanueva

El autor:



Don Jose Fuentes Rodríguez

Agradecimientos

En primer lugar quiero agradecer a mi directora Marta Sestelo Pérez el enorme esfuerzo que me ha dedicado para poder llevar este trabajo a término.

A mis compañeros de Gradiant que me han apoyado y ayudado con todo. Especialmente a mi tutora Nora Martínez Villanueva, que con su amabilidad y entusiasmo ha convertido Gradiant en mi segundo hogar.

A mi familia y amigos, que siempre han estado ahí. En especial a mis padres y mi hermano, por inspirarme a sacar lo mejor de mi.

Finalmente a Tamara, por darme siempre toda la fuerza que necesito.

Índice general

Resumen

Resumen en español

El análisis de comportamiento constituye una amplia rama del análisis de datos que trata de construir un perfil de comportamiento de un individuo a partir del cual es posible detectar eventos anómalos. Esta clase de modelos cobra especial peso en el contexto de la ciberseguridad empresarial, donde estas anomalías pueden corresponderse con ciberataques o filtraciones de información confidencial. En este trabajo, que se ha realizado en colaboración y bajo la supervisión de Gradiant, se han aplicado dos técnicas de análisis de comportamiento de los usuarios en una entidad bancaria; utilizando una clase de modelos de aprendizaje profundo, los *autoencoders* y un modelo de aprendizaje automático, el *isolation forest*. Se incluye una revisión de los fundamentos teóricos de estos modelos y una comparación de la capacidad de ambos en el conjunto de datos de la entidad.

English abstract

User and Entity Behaviour Analytics is a broad branch of data analysis that attempts to build a behavioral profile of an individual in order to detect anomalous events. This kind of analysis In this work, which has been carried under the collaboration and supervision of Gradiant, two user behavior analysis techniques have been applied in a banking institution: using a class of deep learning models, the autoencoders, and a machine learning model, the isolation forest. A review of the theoretical foundations of these models and a comparison of the capability of both models on the bank's dataset are included.

Prefacio

Desde la expansión de los sistemas de computación y las TICs a empresas y consumidores, la importancia de la ciberseguridad ha crecido exponencialmente. En las últimas décadas, la digitalización empresarial ha creado enormes cantidades de datos sensibles, y por otro lado, las amenazas en ciberataques se actualizan constantemente. Por esto, es necesario diseñar nuevos modelos y estrategias para poder incrementar la seguridad. En este marco se encuentran la filosofía de *User and Entity Behaviour Analytics* (Leopold, 2016), conocido también por el acrónimo UEBA, donde aprovechando el acceso a estas grandes colecciones de datos, se construyen modelos de comportamiento habitual para detectar patrones o amenazas a la integridad o seguridad de empresas y organizaciones. Muchos de estos modelos utilizan algunas de las técnicas más punteras de aprendizaje estadístico que se incluyen en el llamado *machine learning (ML)* y *deep learning (DL)*.

Este trabajo de fin de máster se ha realizado coordinado con la ejecución de un proyecto en el marco anterior. Este proyecto, que recibe el nombre de FARO, ha sido realizado en colaboración con Gradiant y trata de construir modelos de aprendizaje automático/profundo y aplicar la filosofía UEBA para detectar posibles anomalías en la operación de los sistemas informáticos internos de una sucursal bancaria; donde la seguridad de la información cobra especial importancia. El objetivo del proyecto es desarrollar un modelo capaz de identificar, entre otros, ataques externos, filtraciones de información o escaladas de privilegios. FARO está incluido en el proyecto ÉGIDA, que pretende acercar las tecnologías de seguridad y privacidad a empresas desde centros de investigación.

La estructura de este trabajo es la siguiente: en primer lugar, se ofrece información general sobre Gradiant, ÉGIDA y la filosofía UEBA. En el capítulo 1 se incluye un resumen de la base teórica del aprendizaje profundo, la cual será utilizada en el capítulo 2, donde se introducirán los modelos propuestos para el proyecto. Finalmente, en el capítulo 3, se recoge la experiencia práctica en el proyecto, con los resultados de la aplicación de estos modelos y conclusiones finales.

0.1. Gradiant y ÉGIDA

Gradiant[1](de sus siglas en inglés, Galician Research and Development Center in Advanced Telecommunications) es un centro tecnológico para la investigación y desarrollo de nuevas tecnologías relacionadas con las TICs y su objetivo es construir innovación y mejorar la competitividad de empresas mediante la transferencia tecnológica ofreciendo su experiencia y conocimientos en el marco de la conectividad, inteligencia y seguridad.

[1] http://www.gradiant.org

Se constituye en 2007 como una fundación privada sin animo de lucro y con representación publica y privada en su patronato, que incluye a las tres universidades gallegas (A Coruña, Vigo y Santiago de Compostela), ocho de las compañías importantes de tecnología y telecomunicaciones y la Asociación Empresarial INEO. Gradiant ha experimentado un crecimiento constante, y hoy en día sus más de 100 empleados han participado en más de 300 proyectos y han solicitado 14 patentes. Su compromiso con la industria les ha llevado a trabajar con más de 370 clientes en 30 países.

La actividad de Gradiant se reparte en 4 áreas principales, que se especializan en investigación de tecnologías concretas: *Sistemas Inteligentes*, *Información Multimodal*, *Comunicaciones Avanzadas* y *Seguridad y Privacidad*. La estructura empresarial puede verse en el organigrama de la figura 1.

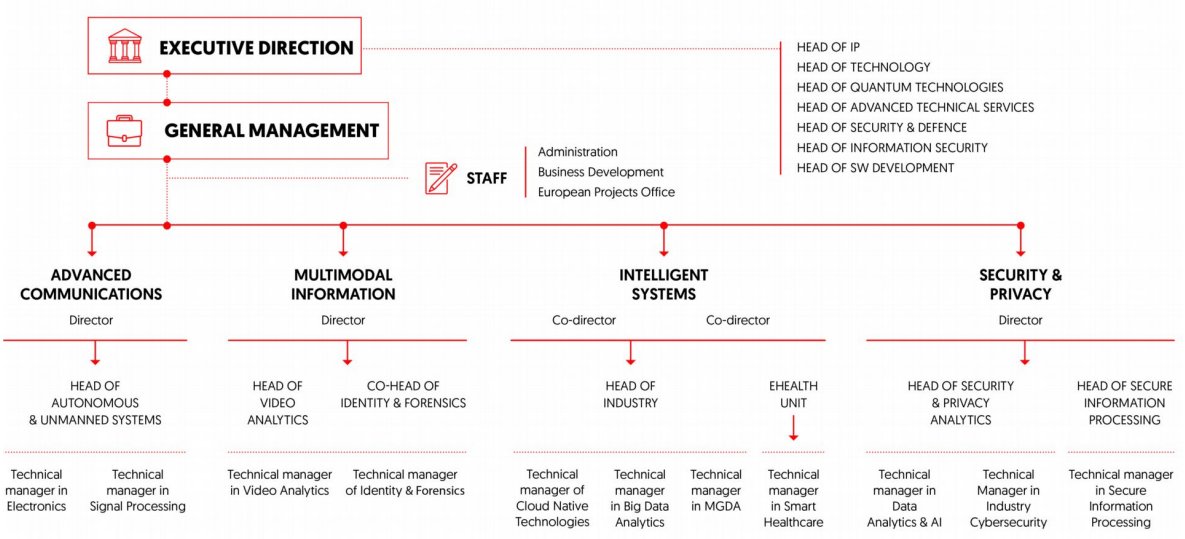


Figura 1: Organigrama de Gradiant.

Este trabajo se realiza dentro del área de Seguridad y Privacidad, en particular en la línea de *Security and Privacy Analytics*. Entre otros, esta área tiene como líneas de trabajo la seguridad biométrica y multimedia, análisis en datos de vídeo y análisis, almacenamiento y procesamiento de datos sensibles en entornos no confiables. En particular, el área está implicada en la ejecución del proyecto ÉGIDA (CER-20191012), en el cual se enmarca la colaboración de este trabajo.

ÉGIDA[2] es la primera red de excelencia estatal en tecnologías para la protección de la privacidad de la información, cuya financiación procede de la Red Cervera. Este proyecto nace del consorcio de los centros tecnológicos Gradiant, Fidesol y los centros de BRTA, Basque Research and Technology Alliance, Ikerlan y Vicomtech, y tiene como objetivo último acercar las tecnologías de seguridad y privacidad a las empresas y fortalecer el sistema nacional de innovación.

El proyecto ÉGIDA establece nuevos caminos para la colaboración e interacción de los centros tecnológicos, entre ellos y con empresas. Este trabajo recoge una de estas colaboraciones, donde una entidad bancaria ha solicitado la elaboración de modelos para ayudar a mejorar la seguridad interna de sus oficinas. Dentro de ÉGIDA, esta colaboración recibe el nombre de FARO. En concreto, el objetivo es construir un sistema que trate problemas de ciberseguridad, como intrusiones (entrada de un usuario externo en el sistema), escalada de privilegios (modificación de un usuario sin permisos para realizar

[2] https://egidacybersecurity.com

acciones restringidas), *bots* (programas que utilizan el equipo para realizar otra actividad como, por ejemplo, lanzar otros ataques) u otros; y que también intente identificar fugas de información ya sea debido a algún ciberataque o a usuarios legítimos (*rogue users*).

Para esta tarea se ha propuesto la realización de modelos, principalmente de aprendizaje profundo, que siguen la filosofía UEBA (*User and Entity Behaviour Analytics*).

0.2. Modelizado por UEBA

El concepto de *User and Entity Behaviour Analytics* o UEBA fue presentado originalmente por la consultora Gartner[3], pero se engloba en un conjunto de sistemas similares que se pueden denominar como *Behaviour Analytics* (BA), y que incluye *Security User Behaviour Analytics* (SUBA) (Cser, 2016),y *User Behaviour Analytics* (UBA) [4]. Todos estos sistemas de análisis de comportamiento se fundamentan en elaborar modelos "base" de actividad ordinaria en una organización y sobre los que se contrastan datos nuevos para detectar actividad sospechosa. Generalmente, los sistemas de BA se basan en la elaboración de modelos estadísticos o de aprendizaje que intentan resumir y predecir el comportamiento habitual de los usuarios, empleados o clientes de un servicio. La característica principal es que para cada uno de estos elementos se elabora un modelo distinto, ya que lo que puede ser un comportamiento típico de un usuario puede ser totalmente anómalo para otro. Por ejemplo, para un comercial resulta normal enviar y recibir gran cantidad de correos externos a su empresa; sin embargo, para un ingeniero de sistemas es posible que la mayoría de su correspondencia sea dentro de la organización, y un aumento en los correos externos podría deberse, por ejemplo, a que su cuenta está bajo control de un atacante. La principal diferencia entre UEBA y otros tipos de BA es que se permite considerar entidades distintas a usuarios. UEBA abarca modelizar el comportamiento habitual de máquinas, como servidores, *switches* o sistemas personales; usuarios máquina (`system` en Windows o `root` en Unix), servicios, sensores y otros además de usuarios reales.

El objetivo de estos modelos es elaborar algo similar a una firma digital de comportamiento, que permite identificar al usuario y comprobar que no se está suplantando su identidad. La principal aplicación de los modelos de UEBA es, por tanto, la detección de anomalías y ataques de ciberseguridad (Maher, 2017): intrusiones, suplantación de identidad, usuarios con objetivos maliciosos (*rogue users*, por ejemplo, filtrado de información a otras empresas) o usuarios negligentes (uso de correos personales, almacenamiento de datos confidenciales en plataformas desprotegidas. Otra característica principal de estos modelos es la cantidad y variedad de datos que se recogen y aglutinan para modelizar cada usuario.

Como algunos ejemplos en el uso de UEBA, en Shashanka, Shen y Wang (2016) se implementan modelos UEBA basados en la distancia de Mahalanobis y *Singular Value Decomposition* para identificar comportamiento anómalo en usuarios que acceden a un servidor, construyendo tanto modelos históricos por usuario como modelos por usuarios similares (*peers*). También en Voris et al. (2019), además de configurar una serie de archivos trampa para atraer e identificar atacantes, se construyen modelos de mixturas gaussianas por cada usuario de computadora a través de datos del sistema de ficheros, lanzamiento de procesos y comportamiento en la red. En este mismo trabajo, también se puede ver otro caso de uso de UEBA: la autenticación activa, es decir, identificar continuamente al usuario, no sólo en los momentos de conexión o entrada, sino durante todo un procedimiento mediante la monitorización de su actividad. Otro ejemplo de este caso de uso puede verse en Pusara y Brodley (2004), donde se

[3] https://www.gartner.com

[4] https://www.ibm.com/es-es/products/qradar-user-behavior-analytics

aplican árboles de decisión a datos de movimiento del ratón para identificar al usuario. Similarmente en Slipenchuk y Epishkina (2019), se comparan resultados de aplicar UEBA con datos de movimientos de ratón, dinámicas en el modo de escribir con el teclado y secuencias de eventos en el contexto de operaciones de banca en línea. Finalmente, en Martin et al. (2021) puede verse una combinación de datos estáticos (*logins*, *cookies*, tipo de sistema, etc.) y dinámicos (ratón, teclado, micrófono, uso de red, etc.) para elaborar modelos de usuario.

Finalmente es importante prestar atención a las posibles limitaciones y sesgos que se asumen cuando se implementan soluciones UEBA. Un resumen de buenas prácticas y problemas a tener en cuenta puede verse en Gates y Taylor (2006), principalmente se destaca tener especial cuidado en asumir lo siguiente:

- Los ataques son anómalos y pueden diferenciarse del comportamiento real.
- Los ataques ocurren pocas veces.
- Los datos anómalos se corresponden con actividad maliciosa.
- Un ratio de falsos positivos mayor que 1 % es aceptable.

En este trabajo se han valorado estos puntos encontrando como solución a varios de estos inconvenientes la elección correcta de variables descriptivas, el filtrado previo mediante reglas fijas por parte de la entidad bancaria y asumir una parte de ruido sobre la muestra de entrenamiento. El modelo que proponemos es similar a los citados anteriormente, obteniendo datos de un amplio abanico de servicios y sistemas entre los que se incluye uso de red y de correo electrónico, acceso a máquinas, lanzamiento de procesos, entre otros. Estos datos se utilizan para elaborar modelos individualizados con redes neuronales de aprendizaje profundo y con otro tipo de técnicas englobadas dentro del aprendizaje automático. En el capítulo 1 y 2 puede encontrarse una exposición más completa de los modelos utilizados. De igual forma en el capítulo 3 pueden verse los resultados obtenidos y algunos comentarios relacionados con problemas presentados anteriormente.

Capítulo 1

Fundamentos Teóricos del aprendizaje profundo

Denominamos Redes Neuronales a un conjunto bastante amplio de modelos de aprendizaje automático que se fundamentan en la idea de aproximar funciones de un conjunto de datos mediante el intercalado de funciones afines y una clase de funciones no lineales, llamadas funciones de activación.

Las redes neuronales aparecen históricamente en el contexto de la neurología, donde el objetivo original era el diseñar modelos matemáticos para explicar el funcionamiento de las neuronas en el sistema nervioso. Se puede considerar que la base de los modelos de redes neuronales está en el trabajo de McCulloch y Pitts (1943), en el que construyen un modelo de una red de neuronas biológicas cuya activación depende de las señales que recibe de las conexiones sinápticas de otras neuronas con ella.

Por otra parte, en 1949, Hebb (1949) introduce en neurología las ideas de aprendizaje Hebbiano que establece mecanismos por los que las neuronas experimentan cambios dependiendo de las conexiones que tienen con otras neuronas y de sus activaciones. Esta idea, trasladada a los modelos abstractos, implica la posibilidad de que neuronas artificiales cambien y se adapten para resolver un problema.

Estas dos ideas convergen en el *perceptrón* de Rosenblatt (1958), que se considera el primer algoritmo que presentaba una red neuronal simple y en la que se basan los desarrollos posteriores. Por primera vez, se propone como un modelo de aprendizaje, es decir, con un algoritmo según el cual las neuronas del perceptrón modifican su efecto usando unos pesos que moderan su activación con el objetivo de reducir el error en tareas de aprendizaje supervisado.

En la década de 1980, aparece por primera vez la idea de aprendizaje profundo (*deep learning*) bajo el contexto del conexionismo en psicología cognitiva, donde se postulaba que se podía obtener un comportamiento inteligente mediante la conexión de elementos más sencillos (véase Goodfellow, Bengio y Courville, 2016:p.17). A nivel computacional, esta filosofía se traducía a procesos de representación distribuida (Rumelhart, Hinton y Mcclelland, 1986), donde varias neuronas trabajaban en paralelo para aprender un modelo, y a la concatenación de las redes anteriores para obtener redes neuronales multicapa.

Durante esta década, aparecen numerosos avances en tipos de redes neuronales en técnicas para enfrentarse a problemas de aprendizaje cada vez más complejos. Entre ellos cabe destacar el algoritmo de *backpropagation* (Rumelhart, Hinton y Williams, 1986), que es hoy en día el algoritmo más utilizado para actualizar los pesos de las neuronas; y las redes neuronales para datos correlacionados, como secuencias y series de tiempo (Hochreiter y Schmidhuber, 1997) o imágenes (Fukushima, 1980) (Lecun et al., 1998).

Además, en las dos últimas décadas del siglo XX, aunque ciertos modelos siguen inspirándose en neurología y psicología, las redes neuronales comienzan a dejar de intentar representar estructuras biológicas y pasan a ser vistas más como modelos matemáticos o computacionales; englobadas en el contexto del aprendizaje estadístico y con objetivos de realizar tareas de regresión o clasificación.

Finalmente, en los últimos años, las redes neuronales han alcanzado un máximo de popularidad gracias a los avances en capacidad de cómputo y de disponibilidad de grandes cantidades de datos, permitiendo modelos cada vez más complejos.

1.1. Estructura de una red neuronal

Como se mencionó anteriormente, las redes neuronales son modelos que intentan aprender o aproximar una función en base a unos datos.

A diferencia de otros modelos estadísticos que se centran en el problema de regresión, las redes neuronales tienen como objetivo aproximar una función determinada, lo cual permite que puedan utilizarse, por ejemplo, para problemas de aprendizaje no supervisado, donde no se cuenta con una variable respuesta, o problemas de aprendizaje por refuerzo.

En particular, las redes de aprendizaje profundo presentan un diseño que podría decirse modular en su arquitectura. Como red neuronal, intentan obtener una función $\hat{f} \approx f : \mathbb{R}^n \longrightarrow \mathbb{R}^m$ con la particularidad de que f sea una composición de funciones más sencillas, es decir

$$\hat{f} = \bigcirc_{l=d}^{1} \hat{f}^{(l)} = \hat{f}^{(d)} \circ \ldots \circ \hat{f}^{(1)}$$

Donde $\hat{f}^{(l)} : \mathbb{R}^{n(l-1)} \longrightarrow \mathbb{R}^{n^{(l)}}$ con $n^{(0)} = n$ y $n^{(d)} = m$, y donde el valor $d \in \mathbb{N}$ se denomina *profundidad* del modelo.

De esta forma una red neuronal profunda se puede entender como un proceso por etapas, en las que se van aplicando las $\hat{f}^{(l)}$ de forma secuencial. A cada etapa del proceso suele denominarse capa, incluyendo las entradas $\mathbf{x}$ como primera etapa o capa de entradas. Las etapas siguientes, $\mathbf{h}^{(1)} = \hat{f}^{(1)}(\mathbf{x}), \mathbf{h}^{(2)} = \hat{f}^{(2)}(\hat{f}^{(2)}(\mathbf{x}))$... se denominan capas ocultas, ya que sus valores no son fácilmente accesibles. Finalmente, la última capa se denomina capa de salida. Además, se denomina anchura de la capa $\mathbf{h}^{(l)}$ a su dimensión, es decir $n^{(l)}$.

Para obtener las estimaciones de $\hat{f}$, es necesario definir las funciones $\hat{f}^{(l)}$ de forma que en el proceso de entrenamiento o ajuste se obtengan las funciones óptimas para el problema propuesto. La forma más simple de hacer esto es si restringimos la elección de funciones a aplicaciones lineales, ya que el problema se reduce a optimizar los coeficientes de una matriz de pesos $W = \prod_{l=d}^{1} W^{(l)}$. Sin embargo, en este caso, la red neuronal no es más que otra forma de representar un modelo lineal. Es habitual considerar el uso de funciones afines, ya que la traslación que incluyen tiene una interpretación

similar al intercepto de los modelos de regresión. Los pesos asociados a estas translaciones suelen denominarse sesgos de la red neuronal. Sin embargo, al igual que en los modelos lineales generalizados, podemos realizar transformaciones afines como transformaciones lineales transformando previamente la matriz de entradas en cada capa añadiendo una columna de unos, es decir, presentándola como una matriz de diseño. De esta forma, resulta equivalente tratar de redes lineales o afines y usaremos esta notación indistintamente; dependiendo de si se quiere marcar atención a la sencillez de representación de funciones lineales mediante matrices, o al efecto de las traslaciones o sesgos.

Para poder modelizar funciones más complejas que las afines y lineales, se añade la no linealidad de una forma sencilla. Para esto, cada función $\hat{f}_l$ se descompone en una transformación afín y una función no lineal perteneciente a una clase de funciones sencillas que se denominan funciones de activación. Las funciones de activación se originan de la idea del umbral de acción de las neuronas biológicas, que envían una señal cuando su potencial de membrana alcanza un cierto valor. Las funciones de activación son más amplias que la activación biológica y permiten que la "señal"que envía cada neurona sea continua. No tienen una definición formal expresa más que el hecho de ser funciones no lineales, aunque sí que existen características que hacen que algunas funciones de activación sean preferibles a otras, ya sea por su mayor facilidad de cómputo, por sus buenas propiedades de optimización, por requerimientos de suavidad o del rango de la función a aproximar.

Con estas ideas, como puede verse en Goodfellow, Bengio y Courville (2016) y en Mhaskar y Poggio (2016) se puede intentar formalizar la idea de una red neuronal profunda de la siguiente forma:

Definición 1.1.1. Sea $\mathcal{F} \subset \{\varphi : \mathbb{R} \longrightarrow \mathbb{R}\}$ un conjunto de funciones de activación. Dado $d \geq 2$, $d \in \mathbb{N}$ y $n^{(0)} = n$, para cada $l = 1, \ldots, d$, sean $n^{(l)} \in \mathbb{Z}^+$ con $n^{(d)} = m$; $A^{(l)} : \mathbb{R}^{n^{(l-1)}} \longrightarrow \mathbb{R}^{n^{(l)}}$ funciones afines y sea $\Phi^{(l)} : \mathbb{R}^{n^{(l)}} \longrightarrow \mathbb{R}^{n^{(l)}}$ tal que $\Phi^{(l)}(\mathbf{x}) = (\varphi_1(x_1), \ldots, \varphi_{n^{(l)}}(x_{n^{(l)}}))$ con $\varphi_j \in \mathcal{F} \quad j = 1 \ldots n^{(l)}$.

Una **red neuronal profunda** con $d-1$ capas ocultas es una función $\hat{f} : \mathbb{R}^n \longrightarrow \mathbb{R}^m$ que cumple:

$$\hat{f}(\mathbf{x}) = \Phi^{(d)}(A^{(d)}(\ldots \Phi^{(l)}(A^{(l)}(\ldots \Phi^{(1)}(A^{(1)}(\mathbf{x}))\ldots). \tag{1.1}$$

Es decir

$$\hat{f} = \Phi^{(d)} \circ A^{(d)} \circ \Phi^{(d-1)} \circ A^{(d-1)} \circ \ldots \circ \Phi^{(1)} \circ A^{(1)} : \mathbb{R}^n \longrightarrow \mathbb{R}^m. \tag{1.2}$$

Siguiendo esta definición, las redes neuronales no profundas pueden verse como una red neuronal con una capa oculta ($d = 2$), esto es, $\hat{f} = \Phi^{(2)} \circ A^{(2)} \circ \Phi^{(1)} \circ A^{(1)}$.

Esta definición formal, que resulta la más sencilla e intuitiva, no carece de defectos. Como se marca Mhaskar y Poggio (2019), la definición 1.1.1 no representa de forma única la función $\hat{f}$, ya que se podría obtener el mismo resultado con otras funciones afines o en otro orden, además esta definición es demasiado secuencial y puede hacer difícil definir ciertos tipos de redes neuronales como redes convoluciones o recursivas, donde la estructura se asemeja más a un árbol. Aunque no sea necesario para el tipo de redes neuronales que se utilizarán en este trabajo, consideramos interesante dar otra definición algo más amplia basándonos en Mhaskar y Poggio (2019) y Cano Córdoba (2018), que además permite definir de una manera más sencilla los conceptos de *neurona*, *capa*, y otros.

Este enfoque es más cercano a la perspectiva histórica cercana a la neurología, por lo que se comienza definiendo la neurona como elemento fundamental del modelo.

Definición 1.1.2. Sea $\varphi : \mathbb{R} \longrightarrow \mathbb{R}$ una función de activación, $n_\eta \in \mathbb{N}$, $\mathbf{w} \in \mathbb{R}^{n_\eta}$ y $b \in \mathbb{R}$.

Se denomina neurona a una función

$$\begin{aligned} \eta : \mathbb{R}^{n_\eta} &\longrightarrow \mathbb{R} \\ \eta(\mathbf{x}) &\mapsto \varphi(\mathbf{w} \cdot \mathbf{x} + b). \end{aligned} \tag{1.3}$$

Las neuronas constituyen la base para definir las redes desde este punto de vista. En comparación con el enfoque anterior de la definición 1.1.1, para una red neuronal $\hat{f} = \Phi^{(d)} \circ A^{(d)} \circ \Phi^{(d-1)} \circ A^{(d-1)} \circ \ldots \circ \Phi^{(1)} \circ A^{(1)}$, cada $\pi_k \circ \Phi^{(l)} \circ A^{(l)}$, donde π_k es la función proyección, se corresponde con una neurona.

Las redes neuronales profundas pasan a definirse en base a estas neuronas como nodos de un grafo en el que se conectan según las entradas de cada neurona. Comenzamos con algunas definiciones relativas a estos grafos.

Definición 1.1.3. Sea $\mathfrak{G} = (V, E)$ un grafo dirigido conexo y acíclico. Para cada $v \in V$ definimos los conjuntos $I_v^- = \{(u, v) \in E \mid u \in V\}$ y $I_v^+\{(v, u) \in E \mid w \in V\}$ como las **aristas salientes y entrantes** de v.

Definimos el **conjunto de nodos origen** del grafo como $V^{(0)} = \{v \in V \mid I_v^- = \emptyset\}$. De forma análoga podemos definir para cada $k > 1$, $k \in \mathbb{N}$ los conjuntos $V^{(k)} \in V$ el conjunto de nodos que tienen una arista desde algún nodo de $V^{(k-1)}$, es decir $V^{(k)} = \{v \in V \mid \exists\, u \in V^{(k-1)} \text{ con } I_v^- \cap I_u^+ \neq \emptyset\}$. Ya que el grafo es acíclico, existe $d \in \mathbb{N}$ tal que $V^{(d)} \neq \emptyset$ y $V^{(k)} = \emptyset \quad \forall k > d$. A $V^{(d)}$ se denomina **conjunto de nodos terminales**.

Definición 1.1.4. Se dice que un grafo dirigido conexo y acíclico $\mathfrak{G}$ es un **grafo a capas** si para cada $i, j \in \mathbb{N}$ con $i \neq j$, $V^{(i)} \cap V^{(j)} = \emptyset$. Si $\mathfrak{G}$ es a capas, cada $V^{(k)}$ se denomina capa k-ésima del grafo, y d recibe el nombre de profundidad de $\mathfrak{G}$.

Ahora definiremos una clase de funciones basadas en un grafo $\mathfrak{G}$, las $\mathfrak{G}$-funciones, que engloban a las redes neuronales profundas.

Definición 1.1.5. Sea $\mathfrak{G} = (V, E)$ un grafo dirigido conexo y acíclico con $|V^{(0)}| = n$ y $|V^{(d)}| = m$. Para cada $v \in V \setminus V^{(0)}$ sea $n_v = |I_v^-|$ el numero de aristas que terminan en v. Para cada v se define una función $f_v : \mathbb{R}^{n_v} \longrightarrow \mathbb{R}$ que denominaremos **función constituyente** en v. Una **$\mathfrak{G}$-función** es una función $g : \mathbb{R}^n \longrightarrow \mathbb{R}^m$ que se computa de la siguiente forma.

1. Para un $\mathbf{x} \in \mathbb{R}^n$, se asocia cada $\mathbf{x}_i \quad i = 1 \ldots n$ con un único nodo de $V^{(0)}$. De esta forma, cada nodo de $V^{(0)}$ representa una entrada real.
2. Para cada $v \in V \setminus V_0$, se calcula su función constituyente f_v tomando como entrada el vector donde sus componentes se corresponden a los valores asociados a los n_v nodos con aristas dirigidas a v. Se asocia el resultado de este cómputo a v.
3. El resultado de $g(\mathbf{x})$ se corresponde al vector cuyas componentes coinciden con los valores asociados a los nodos de $V^{(d)}$

Cabe destacar que g puede obtenerse con distintas elecciones de las funciones constituyentes.

Finalmente presentamos la definición de red neuronal.

Definición 1.1.6. Dado un grafo a capas $\mathfrak{G}$, una red neuronal profunda asociada a $\mathfrak{G}$ es una $\mathfrak{G}$-función donde cada función constituyente es una neurona.

Finalmente vemos la equivalencia entre ambas definiciones:

Proposición 1.1.1. *Sea $\mathfrak{G}$ un grafo dirigido conexo y acíclico. Si $\mathfrak{G}$ es a capas, entonces, las definiciones 1.1.1 y 1.1.6 son equivalentes.*

Demostración. 1.1.1 $\implies$ 1.1.6

Sea $\hat{f} : \mathbb{R}^n \longrightarrow \mathbb{R}^m \quad \hat{f} = \Phi^{(d)} \circ A^{(d)} \circ \ldots \circ \Phi^{(l)} \circ A^{(l)} \circ \ldots \Phi^{(1)} \circ A^{(1)}$, sea $\hat{f}^{(i)} = \Phi^{(i)} \circ A^{(i)} : \mathbb{R}^{n^{(i-1)}} \longrightarrow \mathbb{R}^{n^{(i)}}$ con $n^{(i)} \in \mathbb{N}; \quad n^{(0)} = n, \quad n^{(d)} = m$.

Definimos $\mathbf{h}^{(i)} = \bigcirc_{l=i}^{1} \hat{f}^{(l)}(\mathbf{x})$ y $\mathbf{h}^{(0)} = \mathbf{x}$.

Como para cada $\hat{f}^{(i)}$, $\pi_j(\hat{f}^{(i)}(\mathbf{h}^{(i-1)})) = \varphi_j(W_j^{(i)} \cdot \mathbf{h}^{(i)} + b^{(i)})$ con $W^{(i)} \in \mathcal{M}_{n^{(i)} \times n^{(i-1)}}$, $W_j^{(i)}$ la fila j-ésima de $W^{(i)}$, $b^{(i)} \in \mathbb{R}$ y φ_j una función de activación; se tiene que $\pi_j \circ \hat{f}^{(i)}$ es una neurona.

Construimos un grafo $\mathfrak{G}$ de la siguiente forma:

1. Se define $V^{(0)}$ con n nodos a los que se asocian los valores $\mathbf{x}$.
2. Para cada $i = 1 \ldots d$ se define $V^{(i)}$ como un conjunto de $n^{(i)}$ nodos con aristas tal que $\forall v \in V^{(i)}$, $I_v^- = \{(u, v) \mid u \in V^{(i-1)}\}$. Para cada nodo $v_j \in V^{(i)}$ se le asocia el valor $\pi_j(\hat{f}^{(i)}(\mathbf{h}^{(i-1)})), \quad j = 1, \ldots, n^{(i)}$ y se define su función constituyente como $\pi_j \circ \hat{f}^{(i)}$.
3. Se define $V = \bigcup_{i=1}^{d} V^{(i)}$, $E = \bigcup_{i=1}^{d} \bigcup_{v \in V^{(i)}} I_v^-$ y $\mathfrak{G} = (V, E)$.

Se tiene que $\hat{f}$ es una $\mathfrak{G}$-función y una red neuronal profunda con grafo asociado $\mathfrak{G}$

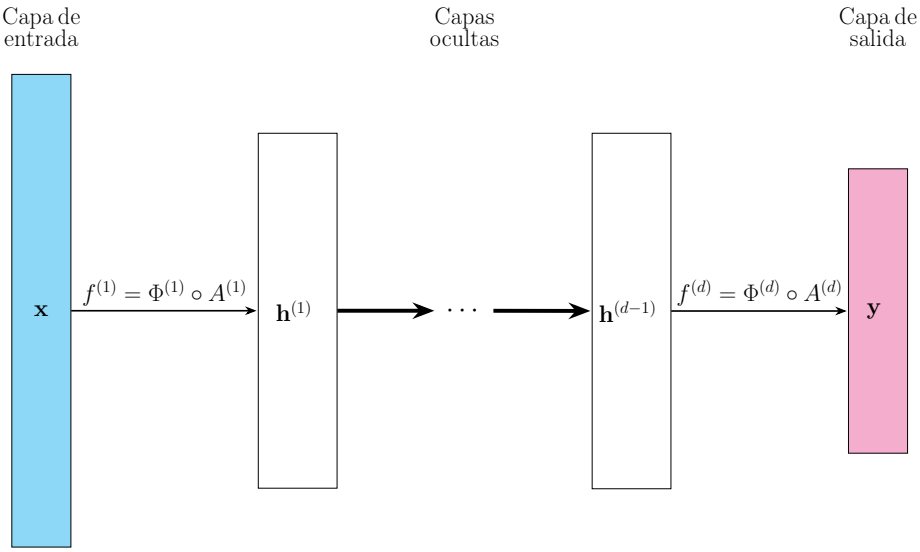
1.1.6 $\implies$ 1.1.1

Sea $\mathfrak{G}$ un grafo a capas y g una red neuronal asociada a $\mathfrak{G}$. Para cada $i = 1, \ldots, n^{(i)}$, $n^{(i)} = |V^{(i)}|$, da un orden a los nodos de $V^{(i)} = \{v_1, ..., v_{n_i}\}$. Se define $\mathbf{h}^{(i)} = (h_1^{(i)}, ..., h_{n^{(i)}}^{(i)}) \in \mathbb{R}^{n^{(i)}}$ tal que cada $h_j^{(i)}$ se corresponde al valor asociado a $v_j \in V^{(i)}$.

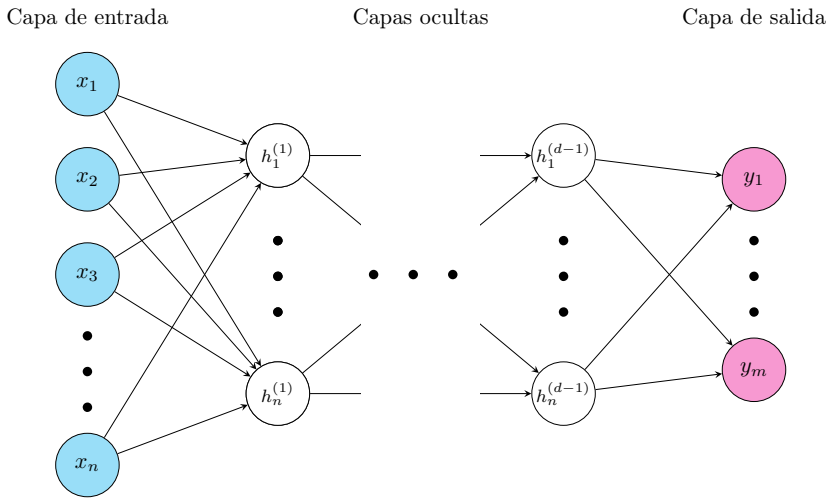
Para cada $v \in V_i$ definimos $\mathbf{z}_v$ como el vector tal que cada componente se corresponde con uno de los valores asociados a los nodos $u \mid (u, v) \in I_v^-$ con el orden correspondiente en $V^{(i-1)}$. Definimos $\hat{f}^{(i)} : \mathbb{R}^{n^{(i-1)}} \longrightarrow \mathbb{R}^{n^{(i)}}$ tal que $\hat{f}^{(i)}(\mathbf{h}^{(i-1)}) = (f_{v_1}(\mathbf{z}_{v_1}), ..., f_{v_{n^{(i)}}}(\mathbf{z}_{v_{n^{(i)}}}))$ donde f_v es la función constituyente en v.

Entonces $\hat{f} = \hat{f}^{(d)} \circ \ldots \circ \hat{f}^{(1)} = g$ □

La definición 1.1.6 permite referirse a una cantidad más amplia de funciones que la definición 1.1.1 y todas las propiedades de las redes funcionales que veremos en las secciones posteriores se cumplen también en estos casos. Sin embargo, en este trabajo nos referiremos siempre a las redes neuronales que cumplen la proposición 1.1.1. Otra ventaja de la definición 1.1.6 es que permite referirnos a las neuronas de forma individual, lo cual reduce la cantidad de notación requerida para explicar algunos algoritmos. Por su parte la definición 1.1.1 es más simple y permite expresar la red neuronal de forma más sencilla con una notación vectorial. Las dos definiciones y sus equivalencias pueden verse en la figura 1.1.



(a) Representación según 1.1.1



(b) Representación según 1.1.6

Figura 1.1: Esquema de una red neuronal profunda

1.2. Funciones de activación

En las dos definiciones de redes neuronales que se muestran en la sección anterior, uno de los conceptos que quedan abiertos es el de función de activación. Como ya se avanzó anteriormente, las funciones de activación tienen una gran importancia en las redes neuronales, ya que sin ellas estos modelos son esencialmente una reformulación de un modelo lineal. Las funciones de activación son las que dotan al modelo de no linealidad y lo que permite ajustar funciones más complejas. Como se mostrará en la sección 1.3, a través del número de funciones de activación a aplicar, ya sea aumentando la anchura de las capas o la profundidad de la red, las redes neuronales son capaces de aproximar un amplio abanico de funciones con una precisión arbitraria.

En ambas definiciones 1.1.1 y 1.1.6, lo único que se exige para las funciones de activación es que sean funciones reales de variable real. Esto es para permitir escoger funciones en cada capa a conveniencia según la necesidad del diseño de la red. Por ejemplo, podríamos querer que el valor de la última capa de la red fuera el promedio de las neuronas en la penúltima capa. En este caso, calcular ese promedio es una función lineal, por lo que la última función de la red sería la función identidad.

Sin embargo, aunque podamos escoger cualquier función para las neuronas, para que la red neuronal sea útil y que pueda ajustarse para unos datos, es necesario que se impongan restricciones sobre al menos algunas de las capas de la red. Por tanto, la mayoría de funciones de activación en una red suelen escogerse de un repertorio de funciones que cumplen con las hipótesis necesarias para garantizar la universalidad (ver sección 1.3) de la red y con algunas propiedades de interés. Estas propiedades no son todas necesarias y muchas de estas funciones solo presentan alguna de ellas. Normalmente la elección entre función de activación u otra está motivada por resultados en problemas similares en la bibliografía o mediante ensayo y error. Entre las "propiedades preferibles" de una red neuronal se encuentran:

- **Diferenciabilidad**: Es preferible que las funciones de activación sean $\mathcal{C}^\infty$ o $\mathcal{C}^1$ para poder calcular el gradiente de la red neuronal sin problemas. Sin embargo, ya que en general el entrenamiento se realiza por métodos numéricos, suele ser suficiente que existan las derivadas laterales en todo el dominio y que el conjunto de puntos de no-derivabilidad sea de medida cero (Berner et al. (2021); Nwankpa et al. (2018)). La falta de derivabilidad puede estar vinculada a problemas de la capacidad de la red neuronal, como se discute en Berner et al. (2021), y debe evaluarse si las ventajas y desventajas de las funciones de activación no diferenciables en cada problema en particular.

- ***Squashing***: El término *squashing function* se refiere a funciones con rango en un intervalo real acotado superior e inferiormente. Esta característica asegura que ciertas neuronas no se vuelven demasiado dominantes.

- **Centradas en cero**: En general es preferible que las neuronas tengan en promedio una activación centrada en 0 pues permite una convergencia más rápida de los métodos de optimización (Gilon et al., 2021).

- **Saturación reducida**: Se dice que una función de activación se satura en $x_0 \in \mathbb{R} \cup \{-\infty, \infty\}$ si $\lim_{x \to x_0} \phi'(x) = 0$. De forma más general, se dice que una función de activación se satura en una región si su derivada es muy próxima a cero. Como veremos en la sección 1.4.2, como el gradiente de la red se obtiene mediante la regla de la cadena, la saturación de las funciones de activación puede resultar en múltiples factores cercanos a cero anulando el gradiente. Por esta razón, es preferible que las funciones de activación se saturen en la mínima cantidad de puntos, o al menos que no se saturen en un entorno de los valores que se encuentra la neurona como entrada. Esta característica se encuentra reñida con la idea del *squashing*, ya que es esos casos es habitual que $-\infty, \infty$ sean puntos de saturación.

- **Baja complejidad computacional**: Ya que normalmente las redes neuronales se ajustan mediante métodos y algoritmos numéricos, es preferible que las funciones de activación y sus derivadas sean fáciles de computar y que requieran poca memoria para mejorar el tiempo de cálculo. Por esto, funciones sin divisiones ni exponenciales suelen ser preferibles.

Hoy en día existen una gran cantidad de funciones de activación válidas que pueden encontrarse en la literatura (Nwankpa et al., 2018; Lederer, 2021; Szandała, 2021). De entre ellas, a continuación se muestra una pequeña lista con algunas de las más utilizadas:

- ***Step***: Se trata de $\varphi : \mathbb{R} \to \{0, 1\}$ tal que

$$\varphi(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0. \end{cases} \tag{1.4}$$

  No puede ser usada fácilmente en redes neuronales ya que no es diferenciable y además su derivada es 0 en todo su dominio. Principalmente tiene valor en el contexto de la neurlogía y para redes que modelan funciones lógicas.

- **Lineal**: $\varphi : \mathbb{R} \to \mathbb{R}$ tal que $\varphi(x) = ax \quad a \in \mathbb{R}$. Como ya se afirmó, si la red neuronal solo tiene activaciones lineales, es equivalente a un modelo lineal. Sin embrago, las activaciones lineales pueden tener sentido en la última capa. Un ejemplo sencillo es tomar la función de la última capa como la función identidad, que es un caso de esta activación.

- **Sigmoide** o **logit**: $\varphi : \mathbb{R} \to (0, 1)$ tal que $\varphi(x) = \frac{1}{1+e^{-x}}$. Es una de las primeras funciones de activación, es diferenciable, *squashing*, no centrada en 0 y computacionalemte complicada. Su derivada se acerca a cero en $-\infty$ y ∞ muy rápidamete por lo que se satura pronto. Se encuentra en desuso para capas ocultas pero aún se utiliza para clasificación binaria, donde la respuesta se tiene como probabilidad de una distribución Bernoulli.

- **Tangente hiperbólica**: $\varphi : \mathbb{R} \to (-1, 1)$ tal que $\varphi(x) = \tanh(x)$. Se trata de una función similar a la sigmoide ya que es diferenciable y *squashing* y relativamente costosa de calcular numéricamente. Sin embargo, es centrada en cero, y aunque presenta los mismos problemas de saturación, cerca de cero, la derivada es mayor que la sigmoide. Estas dos características hacen que el entrenamiento de redes con esta activación sea más rápido que con la anterior.

- **Hardtanh** $\varphi : \mathbb{R} \to [0, 1]$ tal que

$$\varphi(x) = \begin{cases} -1 & \text{if } x < -1 \\ x & \text{if } -1 \leq x \leq 1 \\ 1 & \text{if } x > 1. \end{cases} \tag{1.5}$$

  Se trata de una función *squashing* lineal a trozos que no es diferenciable en $\{-1, 1\}$ y se satura en $\mathbb{R} \setminus [0, 1]$. Es una simplificación de tanh con mucho menos coste de computación numérica.

- **ReLU** $\varphi : \mathbb{R} \to [0, \infty)$ tal que

$$\varphi(x) = \max 0, x = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x \geq 0. \end{cases} \tag{1.6}$$

  La función ReLU (*Rectified Linea Unit*) es también una función lineal a trozos que no es diferenciable en 0. Es muy popular ya que es muy poco costosa computacionalmente en la mayoría de los problemas. Al no estar acotada superiormente no presenta los mismos problemas de saturación

que las sigmoidales. Sin embargo, como tiene derivada 0 para $x < 0$, para entradas negativas es posible que en algunos algoritmos, las neuronas con ReLU dejen de actualizarse. Este fenómeno se denomina *Dying ReLU* o muerte de la ReLU. También se ha marcado que las redes neuronales con ReLU presentan un sesgo hacia los valores positivos debido a que RELU no es centrada en 0, lo que empeora su rendimiento (Clevert, Unterthiner y Hochreiter, 2016).

- **LReLU** $\varphi : \mathbb{R} \to \mathbb{R}$ tal que

$$\varphi(x) = \begin{cases} a \cdot x & \text{if } x < 0,\ a \in \mathbb{R} \\ x & \text{if } x \geq 0. \end{cases} \tag{1.7}$$

  La función *Leaky ReLU* es una versión de ReLU modificada para intentar evitar el problema de muerte de ReLU y el sesgo positivo haciendo que la derivada para valores negativos no sea 0.

- **ELU** $\varphi : \mathbb{R} \to \mathbb{R}$ tal que

$$\varphi(x) = \begin{cases} a \cdot (e^x - 1) & \text{if } x < 0,\ a \in \mathbb{R} \\ x & \text{if } x \geq 0. \end{cases} \tag{1.8}$$

  ELU es una activación similar a ReLU y LReLU que también intenta reducir la muerte de ReLU. Da mejores resultados para algunas redes neuronales, pero es más computacionalmente compleja debido a no ser lineal a trozos.

1.3. Aproximación universal de funciones

Como avanzamos en la sección anterior, las redes neuronales son concebidas como modelos para aproximar funciones. En el caso más general, se parte de un conjunto de covariables X con una variable respuesta Y, y el objetivo es encontrar una función que las relacione $f(\mathbf{x}) = \mathbf{y}$. Esta relación cuando se trata de variables aleatorias pasa a ser la función de regresión m tal que $y = m(\mathbf{x}) + \varepsilon, \quad \mathbb{E}(\varepsilon) = 0$. En un problema de clasificación se trata de aproximar la misma función pero siendo la respuesta, por ejemplo, una variable con distribución binomial. Sin embargo, las redes neuronales no se limitan a los modelos estadísticos, si no que se convierten en un marco genérico para aproximar funciones a partir de unos puntos que suponemos evaluados en la función objetivo. Ejemplos de esto se encuentran en redes neuronales que tratan de construir circuitos lógicos o verificar leyes físicas.

La idea de que las redes neuronales, una vez entrenadas, nos ofrecen una aproximación de la función objetivo nos lleva a pensar en cual puede ser el rango de aplicación de éstas. Ya solo en estadística, la relación entre variables puede ser extremadamente complicada, lo que ha llevado al auge de los métodos estadísticos no paramétricos, como los modelos basados en *kernels* (Nadaraya, 1964; Watson, 1964) o en *splines* como los GAM (Hastie, 2017). Lo ideal sería que las redes neuronales tuvieran capacidad de representar cualquier función que se quisiera exigiendo una precisión arbitraria, aunque fuera necesario incrementar la profundidad de la red o la cantidad de neuronas por capa. Esto sería muy similar a lo que ocurre con los modelos de mixturas gaussianas donde, dada una cantidad suficiente de componentes, el modelo puede aproximar cualquier función de densidad (véase Titterington, Makov y Smith, 1995:p.50).

Este concepto recibe el nombre de aproximación universal. Sin embargo, es importante definir correctamente el significado de "aproximar" y "universal". Es decir, en que punto consideramos que una función aproxima a otra y qué conjuntos de funciones podemos aproximar. Aproximar funciones requiere que estos conjuntos que tomemos sean espacios métricos, en general espacios de Banach. Así podemos dar una definición de capacidad de aproximación:

Definición 1.3.1. Sea $\mathcal{G}$ un espacio de funciones con una topología, sea $\mathcal{A} \subset \mathcal{G}$. Se dice que $\mathcal{A}$ tiene capacidad de aproximar $\mathcal{G}$ si $\mathcal{A}$ es denso en $\mathcal{G}$, es decir si $cl_{\mathcal{G}}(\mathcal{A}) = \mathcal{G}$.

Si $\mathcal{G}$ es métrico como ocurrirá habitualmente, tenemos que cuando $\mathcal{A}$ aproxima $\mathcal{G}$, para cada $f \in \mathcal{G}$ podemos encontrar $g \in \mathcal{A}$ arbitrariamente cerca.

Para establecer una notación nos referiremos como $\mathcal{N}_{(A,B;\mathfrak{G},\mathcal{F})}$ al conjunto de redes neuronales con estructura dada por $\mathfrak{G} = (V, E)$ y activaciones en $\mathcal{F}$. Por otro lado definimos $\mathcal{N}_{A,B;d,\mathcal{F}}$ como el conjunto de redes neuronales con profundidad fijada d y $\mathcal{N}_{A,B;\mathcal{F}}$ a el conjunto de redes neuronales con grafo arbitrario. Se trata de ver la densidad de estos conjuntos de redes neuronales en espacios de funciones de interés. Algunos de estos espacios que se tratan en los teoremas incluyen (en lo siguiente A es un abierto de $\mathbb{R}^n$):

- $\mathcal{C}(A)$ es el espacio de funciones continuas en su dominio A, dotado de la norma $\|\cdot\|_\infty$ con $\|f\|_\infty = \sup_{\mathbf{x}\in(A)} |f(x)|$
- $\mathcal{B}(A)$ es el espacio de funciones Borel medibles con la norma anterior.
- $L^p_\mu(A),\ 1 \leq p < \infty$ es el espacio de funciones Lebesgue medibles tales que $\|f\|_{L^p_\mu} = (\int_A |f|^p \, d\mu)^{1/p}$ cocientadas por el kernel de $\|\cdot\|_{L^p_\mu}$ y con norma $\|\cdot\|_{L^p_\mu}$.
- $L^\infty_\mu(A)$ es el espacio de funciones Lebesgue μ-medibles acotadadas en casi todo punto cocientadas por el kernel de su norma $\|\cdot\|_{L^\infty_\mu}$, $\quad \|f\|_{L^\infty_\mu} = \text{ínf}\{s|\mu(\{\mathbf{x} : |f(\mathbf{x}) \geq s|\}) = 0\}$.

Durante las últimas décadas ha habido un gran esfuerzo en estudiar qué condiciones es necesario imponer a la red neuronal para conseguir la densidad en distintos espacios. La primera versión del teorema de aproximación universal de redes neuronales se encuentra en Cybenko (1989), donde se afirma que las redes neuronales de una capa oculta definidas en un compacto $K \subset \mathbb{R}^n$ y con función de activación continua φ con $\lim_{x\to-\infty} \varphi(x) = 0, \quad \lim_{x\to\infty} \varphi(x) = 1$; son densas en $\mathcal{C}(K)$. Además, en el trabajo de Hornik, Stinchcombe y White (1989) se establece un resultado similar pero donde se relajan los requerimientos para las funciones de activación, siendo solamente necesario que φ sea monótona creciente, *squashing* y Borel-medible, permitiendo así funciones con una cantidad finita de discontinuidades. Años después Hornik (1991) extiende estos resultados a redes neuronales con cualquier cantidad fijada de capas ocultas, donde con suficiente anchura de las capas, puede asegurarse una buena aproximación. Además, reduce las condiciones necesarias de φ de nuevo, estableciendo los siguientes dos teoremas:

Teorema 1.3.1. *Si $\varphi : \mathbb{R} \to \mathbb{R}$ es acotada y no constante, entonces, para toda medida finita μ de $\mathbb{R}^n$ el conjunto $\mathcal{N}_{\mathcal{R}^n,\mathcal{R}^m;d,\{\varphi\}}$ es denso en $L^p_\mu(\mathbb{R}^n) = \{f : \mathbb{R}^n \to \mathbb{R}^m \mid \int_{\mathbb{R}^n} |f(x)|^p \, d\mu(x) < \infty\}$ según la norma $\|\cdot\|_\infty$.*

Teorema 1.3.2. *Si $\varphi : \mathbb{R} \to \mathbb{R}$ es continua, acotada y no constante, entonces, para todo compacto $K \subset \mathbb{R}^n$ el conjunto $\mathcal{N}_{K,\mathcal{R}^m;d,\{\varphi\}}$ es denso en $\mathcal{C}(K)$ según la norma $\|\cdot\|_\infty$.*

Nota. Nótese que por ser $\varphi : \mathbb{R} \to \mathbb{R}$ acotada y no constante, en particular, φ no puede ser un polinomio.

Estos teoremas permiten utilizar un amplio abanico de funciones de activación, sin embargo muchas de las funciones de activación que se utilizan con buenos resultados no son acotadas. Finalmente, con

el trabajo de Leshno et al. (1993) se consigue la versión moderna del teorema de universalidad para un número fijo de capas. Ésta se presenta a continuación.

Definición 1.3.2. Dada una medida de Lebesgue μ, sea $A \subset \mathbb{R}^n$ un abierto medible, sea f definida para casi todo punto de A. Se dice que f está **esencialmente acotada localmente** en A, equivalentemente $f \in L^\infty_{loc}(A)$ si para cada $K \subset A$ compacto, f está acotada en K menos a lo sumo un conjunto de medida cero.

Teorema 1.3.3. *Sea $\varphi \in L^\infty_{loc}(\mathbb{R})$ y, además el conjunto de puntos de discontinuidad de φ es de medida cero. Entonces $\mathcal{N}_{\mathbb{R}^n,\mathbb{R}^m;d,\{\varphi\}}$ es denso en $\mathcal{C}(\mathbb{R}^n)$ con la topología inducida por $\|\cdot\|_{L^\infty_\mu}$ si y solo si φ no es un polinomio para casi todo $x \in \mathbb{R}$*

Además, si μ es una medida absolutamente continua con respecto a una medida de Lebesgue, entonces $\mathcal{N}_{\mathbb{R}^n,\mathbb{R}^m;d,\{\varphi\}}$ es denso en $L^p_\mu(A),\ 1 \leq p < \infty$

Estos teoremas no imponen ninguna limitación en la anchura de las redes, esto quiere decir que aunque sea posible encontrar una red neuronal para aproximar cualquier función, ésta puede ser inviable en la práctica. En Barron (1993) se puede encontrar una cota para la anchura de una red con una capa oculta y activaciones sigmoides para aproximar un amplio abanico de funciones, incluyendo funciones $\mathcal{C}^\infty(K)$ para $K \subset \mathbb{R}^n$ compactos. Sin embargo es posible que para ciertos casos se requiera una anchura exponencial con respecto a n (Goodfellow, Bengio y Courville, 2016:sec. 6.4.1). Para solventar este problema, una gran parte del estudio en la capacidad de redes neuronales se enfoca en analizar el efecto de la profundidad y la anchura, es decir, si el aumento de la cantidad de capas reduce la cantidad de neuronas totales para aproximar funciones y si para una anchura fija puede asegurarse la aproximación universal con suficiente profundidad.

Entre los trabajos donde se investigan estas propiedades podemos encontrar Hanin (2019), donde se muestra una primera versión para la universalidad de redes profundas con anchura fija y función de activación ReLU. En particular, muestran que las redes neuronales $\mathcal{N}_{[0,1]^n,\mathbb{R}^+,\{ReLU\}}$ con anchura en capas ocultas $n^{(i)} = n+1,\ i = 1,..,d-1$ son densas en $\{f : [0,1]^n \to \mathbb{R}^+ : f \text{ es continua y convexa}\}$ para la norma $\|\cdot\|_\infty$. En este mismo documento puede verse un resumen de otros trabajos anteriores sobre universalidad de redes con anchura fija. Asimismo, en Hanin y Sellke (2018) se amplían los resultados anteriores de redes ReLU a $\{f : [0,1]^n \to \mathbb{R}^m : f\text{es continua}\}$. Finalmente en Kidger y Lyons (2020) se da un teorema más general para redes neuronales de anchura fija:

Teorema 1.3.4. *Si φ es una función continua, no afín, $\mathcal{C}^1$ en al menos un punto y con derivada distinta de 0 en ese punto; entonces para todo $K \subset \mathbb{R}^n$ el conjunto $\mathcal{N}_{\mathbb{R}^n,\mathbb{R}^m,\{\varphi\}}$ con anchura en capas ocultas $n^{(i)} = n+m+2,\ i = 1,..,d-1$ es denso en $\mathcal{C}(K)$ con la norma $\|\cdot\|_\infty$.*

Nota. Este teorema permite que φ sea un polinomio, aunque le impone otras restricciones.

Los resultados anteriores fundamentan la base teórica para justificar el uso de redes neuronales para problemas como el que atañe a este trabajo. Sin embargo la investigación en este ámbito aún presenta muchos frentes abiertos, de los que sólo tenemos resultados de la experiencia como la elección de función de activación, el tamaño que se requiere tanto en ancho como en profundidad para obtener buenas aproximaciones y el papel que juegan estas dos dimensiones de la red en la velocidad de aproximación. Con respecto a esto último, en los últimos años han aparecido nuevas publicaciones con resultados empíricos como el de Zhou y Paffenroth (2017) y Szegedy et al. (2014), y en algunos avances teóricos como en Mhaskar y Poggio (2019) parece que la profundidad tienen un mayor efecto en la capacidad

de las redes neuronales para al menos algunas aplicaciones. Esto justifica también querer utilizar en este proyecto redes neuronales profundas.

1.4. Optimización de redes neuronales

La sección anterior da una justificación para poder utilizar las redes neuronales para enfrentarnos a multitud de problemas prácticos en los que suponemos que los datos tienen algún tipo de relación funcional entre ellos. Todos los resultados hasta el momento están pensados sobre la aproximación de las redes neuronales a estos funcionales. Sin embargo, en realidad prácticamente nunca tenemos acceso a la función que queremos aproximar, sino unos datos que son teóricamente una realización de esta función en un conjunto de puntos en las variables disponibles. Más aún, normalmente los datos que disponemos suelen ser variables aleatorias, incluyendo las respuestas, de forma que realmente se trata de aproximar una función estocástica. Este problema de estimar una función únicamente con un conjunto de datos que ellos mismos puedan presentar variabilidad se denomina **problema de aprendizaje**. En Berner et al. (2021) se puede encontrar un desarrollo formal de teoría del aprendizaje para redes neuronales que es compatible con lo desarrollado en este trabajo hasta el momento. Aquí mostramos únicamente un esbozo general basándonos en Berner et al. (2021) y en el capítulo 8 de Goodfellow, Bengio y Courville (2016), para justificar el proceso de estimación. La idea principal se muestra a continuación; sea $X = \mathbf{x}_1, \dots, \mathbf{x}_r,\ r \in \mathbb{Z}^+$ una muestra aleatoria y $\mathbf{x}$ una variable aleatoria con una distribución $\mathcal{X}$ y f una función a estimar. Supongamos que en nuestra muestra de datos tenemos una serie de variables respuesta $Y = \mathbf{y}_1, \dots \mathbf{y}_r$, que suponemos que son realizaciones de la función que estamos aproximando, el problema de aprendizaje consiste en lo siguiente:

- Se considera una función de pérdida $\mathcal{L}(g, \mathbf{x}, \mathbf{y})$ que se minimiza cuando $g(\mathbf{x})$ se acerca a $\mathbf{y} = f(\mathbf{x})$, por ejemplo, una distancia.
- Se considera una función de error, coste o riesgo, dado por $\mathcal{R}(g) = \mathbb{E}_{\mathcal{X}}(\mathcal{L}(g, \mathbf{x}, \mathbf{y}))$ y un estimador de este, que se denomina riesgo o coste empírico,

$$\hat{\mathcal{R}}(g, X, Y) = \frac{1}{r} \sum_{i=1}^{r} \mathcal{L}(g, \mathbf{x}_i \mathbf{y}_i). \tag{1.9}$$

- Se busca la función $\hat{f}$ que minimiza $\hat{\mathcal{R}}$.

Normalmente las funciones de pérdida que se toman dependen del problema a tratar. Los riesgos empíricos más habituales son:

- Para $\mathbf{y_i} = (y_{i;1}, \dots, y_{i,m}) \in \mathbb{R}^m$ o cuando la función objetivo tiene imagen en $\mathbb{R}^m$, la función de coste más utilizada es el **error cuadrático medio** (**ECM**).

$$ECM = \frac{1}{r} \sum_{i=1}^{r} \mathcal{L}_2(g, \mathbf{x}_i) = \frac{1}{r} \sum_{i=1}^{r} \sum_{j=1}^{m} (y_{i;j} - g_j(x_i))^2 \tag{1.10}$$

- En el mismo caso, otra función utilizada es el **error absoluto medio** (**EAM**).

$$EAM = \frac{1}{r} \sum_{i=1}^{r} \mathcal{L}_1(g, \mathbf{x}_i) = \frac{1}{r} \sum_{i=1}^{r} \sum_{j=1}^{m} |y_{i;j} - g_j(x_i)| \tag{1.11}$$

- Para problemas de clasificación, es decir $\mathbf{y_i} \in \mathbb{Z}^m$, se suele utilizar la **entropía cruzada**.

$$H = \frac{1}{r}\sum_{i=1}^{r}\mathcal{L}_H(g, \mathbf{x}_i) = \frac{1}{r}\sum_{i=1}^{r}\sum_{j=1}^{m}(-1)y_{i;j}\log(g_j(x_i)) \tag{1.12}$$

1.4.1. Balance sesgo-varianza

Es importante remarcar que optimizar estos estimadores del riesgo con nuestros modelos no nos asegura la optimización del riesgo. Centrarnos únicamente en optimizar el riesgo empírico puede llevar a *overfitting*, es decir, un modelo que únicamente responde bien a la muestra y que en vez de estimar la función objetivo, interpole los datos. En Berner et al. (2021) puede verse una descomposición de $\mathcal{R}$ que aporta más información sobre esto. Si denominamos por $\mathcal{N}$ el conjunto de modelos disponibles: en nuestro caso, redes neuronales de anchura, profundidad y activación fijados, llamamos f^* a la mejor aproximación de la función objetivo en $\mathcal{N}$ es decir $f^* \in argmin_{f\in\mathcal{N}}\mathcal{R}(f^*)$. De la sección 1.3 se sabe que f^* puede aproximarse arbitrariamente a f, dependiendo únicamente del tamaño y características de las redes de $\mathcal{N}$. Se tiene

$$\mathcal{R}(g) \leq \underbrace{\hat{\mathcal{R}}(g) - \hat{\mathcal{R}}(\hat{f})}_{\varepsilon^{opt}} + \underbrace{2\cdot \underset{\gamma\in\mathcal{N}}{sup}\left|\mathcal{R}(\gamma) - \hat{\mathcal{R}}(\gamma)\right|}_{\varepsilon^{gen}} + \underbrace{\mathcal{R}(f^*)}_{\varepsilon^{\mathcal{N}}} = \varepsilon^{opt} + 2\varepsilon^{gen} + \varepsilon^{\mathcal{N}}, \tag{1.13}$$

siendo

- ε^{opt} el error de optimización, que representa el error debido a no tener riesgo empírico óptimo. Es el que se puede reducir mediante el entrenamiento.
- ε^{opt} el error de generalización que está vinculado a la varianza del estimador del riesgo. Conforme $\mathcal{N}$ aumenta (es decir, se permiten redes con más neuronas), este error también aumenta.
- $\varepsilon^{\mathcal{N}}$ el error de aproximación que, tal y como se comentó anteriormente, representa el error de aproximación entre f la mejor aproximación de f en $\mathcal{N}$, por lo que puede considerarse como el sesgo del error. Además, también se pudo ver que para reducir este error pueden tomarse redes más complejas, es decir, ampliar $\mathcal{N}$.

Esta relación entre los tipos de errores se conoce habitualmente como balance sesgo-varianza. De esta relación se ve que los modelos que minimizan ε^{opt} no garantizan ser el mejor posible y que aunque aumentar el número de neuronas generalmente da menos error, es responsable de *overfitting*. Para solucionar esto, en *machine learning* es habitual dividir el conjunto de datos en tres grupos. En primer lugar, el conjunto de datos de entrenamiento, en los que se entrena el modelo. Al evaluar el rendimiento del modelo sobre estos datos se obtiene el riesgo empírico. Por otro lado en los conjunto de datos de test y evaluación se evalúa el modelo una vez optimizado. Cuando se mide el coste empírico en validación y test, se obtiene un mayor error ya que no forman parte de la muestra sobre la que se ha ajustado el modelo. Sin embargo, como se supone que los datos son independientes y con la misma distribución, el coste sobre los datos de test o validación da una idea del error de generalización del modelo. La diferencia entre ambos conjuntos es que los datos de validación se utilizan para seleccionar entre un conjunto de modelos. Una vez hecho esto el conjunto de test da una medida independiente y global del mejor modelo escogido.

Además de utilizar esta separación de los datos, existen métodos para modificar la función de coste empírico o restricciones sobre $\mathcal{N}$ para reducir el sobreajuste en la optimización. Estas modificaciones

se denominan en general como regularización y se tratan en más detalle en 1.4.4. Si la regularización se realiza modificando el cálculo del coste empírico, este nuevo valor se denomina coste regularizado $\hat{\mathcal{R}}^{reg}$. Como la regularización depende del problema, por sencillez de notación usaremos $\mathcal{J}$ para referirnos a la cantidad a minimizar, ya sea $\hat{\mathcal{R}}^{reg}$ o $\hat{\mathcal{R}}$.

1.4.2. *Backpropagation*

Según lo que hemos desarrollado, lo que comúnmente se llama aprendizaje o entrenamiento de una red neuronal es en realidad el proceso de optimizar el coste empírico o regularizado. Es decir, se trata de encontrar la red neuronal $\hat{f}$ en $\mathcal{N}$ que minimice $\mathcal{J}$. Esto parece muy complejo a simple vista pero proviene de la estructura simple de las redes neuronales.

Recordemos que $\mathcal{N}$ es el conjunto de redes neuronales con profundidad d, vinculadas al mismo grafo computacional $\mathfrak{G}$, y con la misma función de activación φ (los resultados que se presentan aquí son válidos si las funciones de activación son distintas en cada capa o neurona, pero se opta por fijarlos iguales para simplificar las expresiones). Según la definicion 1.1.1 todas las funciones $g \in \mathcal{N}$ son composición de funciones afines $A^{(i)}$ y la función de activación. Cada una de estas aplicaciones afines pueden descomponerse en una transformación lineal, representada como una matriz de pesos $W \in \mathcal{M}_{n^{(i)} \times n^{(i-1)}}$ y una traslación $\mathbf{b} \in \mathbb{R}^{n^{(i)}}$. Esta descomposición puede observarse también en las neuronas de la definición 1.1.6 y en la demostración de la proposición 1.1.1. De esta forma podemos ver que cada red neuronal viene parametrizada por $\theta = \left\{ w_{j,k}^{(i)}, b_k^{(i)}, \quad j = 1, \dots n^{(i-1)},\ k = 1, \dots n^{(i)},\ i = 1, \dots, d \right\}$ y por tanto, podemos notarla por $g_\theta \in \mathcal{N}$. Poder parametrizar las redes neuronales nos abre a su vez el camino para la optimización, ahora el problema de aprendizaje se reduce a encontrar los valores de θ para los cuales g_θ minimice $\mathcal{J}$ con los datos de entrenamiento. Para encontrar este mínimo, como en cualquier otro problema de optimización, la forma más sencilla de actuar seria obtener el gradiente de $\mathcal{J}$ con respecto a los parámetros de θ y utilizar métodos de descenso por gradiente. El problema ahora radica en que θ tiene una gran cantidad de parámetros y tanto el cálculo del gradiente como el uso de estos algoritmos para tantas dimensiones es muy costoso.

Para solventar esto, Rumelhart, Hinton y Williams (1986) propusieron un algoritmo que aprovecha la estructura modular de las redes neuronales como composición de funciones para obtener el gradiente de $\mathcal{J}$. Este algoritmo, que es hoy en día de uso mayoritario, recibe el nombre de *backpropagation*. Incluimos una breve explicación del desarrollo de este algoritmo, que se puede encontrar en más detalle en la sección 3.2 de Aggarwal (2019). El algoritmo se divide en tres fases:

- **Inicialización**: en primer lugar se asignan valores aleatorios a los parámetros en θ, normalmente cercanos a 0.
- **Fase hacia delante**: en este paso se calcula el resultado de aplicar la red neuronal sobre los datos. Se calculan en este proceso los valores intermedios de la red como los valores de las capas $\mathbf{h}^{(i)} = \bigcirc_{l=i}^{1} \hat{f}^{(l)}(\mathbf{x})$ y lo valores preactivación $\mathbf{a}^{(i)} = A^{(i)}(\mathbf{h}^{(i-1)})$. Denotaremos además $\mathbf{o} = (o_1, \dots, o_m) = g(\mathbf{x})$ a los valores de la capa de salida de la red. Con estos valores se puede calcular la función de coste $\mathcal{J}$ y su derivada con respecto de $o_1, \dots, o_m$.
- **Fase hacia atrás**: se calculan las derivadas parciales de $\mathcal{J}$ con respecto a cada uno de los parámetros de θ. Para hacer esto, se comienza calculando las derivadas con respecto a los parámetros de las últimas capas y utilizándolas junto con la regla de la cadena para obtener las derivadas con respecto a las capas anteriores. Esta es la razón por la que se denomina propagación hacia atrás.

Veamos esta última etapa con más detalle. Utilizaremos la regla de la cadena multivariante:

$$\frac{\partial f(g_1(w),\dots,g_k(w))}{\partial w} = \sum_{i=1}^{k} \frac{\partial f(g_1(w),\dots,g_k(w))}{\partial g_i(w)} \cdot \frac{\partial g_i(w)}{\partial w} \tag{1.14}$$

Nos centraremos en el caso de solo una neurona de salida $m = 1$ por simplificar los cálculos. Para cada neurona de la capa $V^{(i)}$ con salida $h_j^{(i)}$ definimos $w_{(h_k^{(i-1)};h_j^{(i)})}$ como el peso asignado a $h_k^{(i-1)}$ en el cálculo de $h_j^{(i)}$. Supongamos que queremos calcular la derivada

$$\frac{\partial \mathcal{J}}{\partial w_{(h_k^{(i-1)};h_j^{(i)})}}. \tag{1.15}$$

Debido a la estructura de la red (vease figura 1.1), existen múltiples neuronas en las capas $V^{(i+1)},\dots,V^{(d)}$ que dependen del valor de $w_{(h_k^{(i-1)};h_j^{(i)})}$. Esto tiene efecto en la derivada de $\mathcal{J}$, ya que la función de coste depende a su vez de todas estas neuronas. Para marcar en qué variables tiene efecto el cambio en $w_{(h_k^{(i-1)};h_j^{(i)})}$ definimos el conjunto $P_{h_j^{(i)},o}$ de caminos en el grafo desde la neurona $h_j^{(i)}$ hasta o, es decir, $(h_j^{(i)},\dots,o)$; además definimos para cada $p \in P_{h_j^{(i)},o}$, la longiud $|p|$ que es el numero de nodos que componen p y $p_t,\ t = 1,...,T = |p|$ al elemento t-ésimo del camino (en particular $p_1 = h_j^{(i)}$ y $p_T = o$). De esta forma ya se puede calcular la derivada anterior:

$$\frac{\partial \mathcal{J}}{\partial w_{(h_k^{(i-1)};h_j^{(i)})}} = \underbrace{\frac{\partial \mathcal{J}}{\partial o} \sum_{p\in P} \prod_{t=2}^{T} \frac{\partial p_{t+1}}{\partial p_t}}_{\delta(h_j^{(i)})} \cdot \frac{\partial h_j^{(i)}}{\partial w_{\left(h_k^{(i-1)};h_j^{(i)}\right)}}. \tag{1.16}$$

Esta descomposición es lo fundamental del algoritmo, ya que el cálculo de $\delta(h)$ es recursivo. En el primer paso del algoritmo se inicializa $\delta(o)$ como $\delta(o) = \frac{\partial \mathcal{J}}{\partial o}$, que ya se ha obtenido en la fase hacia delante. En los siguientes pasos se calcula recursivamente de la siguiente forma: supongamos que hemos obtenido $\delta(p_{t+1})$ con p_t, p_{t+1} neuronas del camino $p \in P_{h_j^{(i)},o}$. Entonces

$$\delta(p_t) = \frac{\partial \mathcal{J}}{\partial p_{t+1}} \frac{\partial p_{t+1}}{\partial p_t} = \delta(p_{t+1}) \frac{\partial p_{t+1}}{\partial p_t}, \tag{1.17}$$

por lo que solamente es necesario calcular $\frac{\partial p_{t+1}}{\partial p_t}$ para cada neurona:

$$\frac{\partial p_{t+1}}{\partial p_t} = \varphi'(a_{t+1}) w_{(p_t;p_{t-1})}. \tag{1.18}$$

Para terminar el cálculo de la derivada 1.16 se obtiene el último factor de forma similar:

$$\frac{\partial h_j^{(i)}}{\partial w_{\left(h_k^{(i-1)};h_j^{(i)}\right)}} = \varphi'(h_j^{(i)}) \cdot h_k^{(i-1)} \tag{1.19}$$

y por tanto

$$\frac{\partial \mathcal{J}}{\partial w_{(h_k^{(i-1)};h_j^{(i)})}} = h_k^{(i-1)} \varphi'(h_j^{(i)}) \delta(h_j^{(i)}). \tag{1.20}$$

Al final del algoritmo *backpropagation* se obtiene el gradiente de la función $\mathcal{J}$ en cada punto promediando los resultados en cada dato según

$$\nabla_\theta \mathcal{J} = \frac{1}{r} \sum_{i=1}^{r} \nabla_\theta \mathcal{J}(x_i, y_i, \theta). \tag{1.21}$$

1.4.3. Algoritmos de optimización por gradiente

Con este gradiente, el problema de aprendizaje puede solucionarse aplicando métodos de optimización por gradiente para encontrar los mínimos de $\mathcal{J}$. Sin embargo, muchos de los métodos clásicos, como el método de Newton-Raphson, no dan buenos resultados en esta optimización. Esto se debe principalmente a la alta dimensionalidad del problema, teniendo en cuenta que el gradiente tiene tantas componentes como parámetros en θ, provocando que la matriz hessiana sea muy mal condicionada y por otro lado, los cálculos de estos métodos sean muy costosos computacionalmente (ver Goodfellow, Bengio y Courville, 2016:cap 8). En cambio, los métodos más utilizados son los denominados de forma genérica métodos de descenso estocástico. Incluímos aquí dos de los algoritmos más usados de esta clase: el *Minibatch Stochastic Gradient Descent*(Robbins y Monro, 1951) y *Adam*(Kingma y Ba, 2017).

Minibatch Stochastic Gradient Descent (MBSGD)

La idea detrás del MBSGD es tomar de forma aleatoria un submuestreo de los datos, denominado *minibatch*, y calcular una estimación (insesgada) del gradiente mediante *backpopagation*. En la sección 8.3.1 de Goodfellow, Bengio y Courville (2016) se recoge una serie de ventajas del submuestreo, siendo la más importante la mejora en la velocidad de cómputo y la posibilidad de paralelizarlo. Su estructura se resumen en el algoritmo 1.

Algoritmo 1: Minibatch Stochastic Gradient Descent

Hiperparámetros: Tamaño del *minibatch* m, *learning rate* η, criterio de parada.

Se inicializa θ aleatoriamente.

mientras *No se cumple criterio de parada* **hacer**

- Submuestreo de m datos de X con sus correspondientes respuestas de Y.
- Calcular la estimación del gradiente $\widetilde{\nabla_\theta \mathcal{J}} = \frac{1}{m} \sum_{i=1}^{m} \nabla_\theta \mathcal{J}(x_i, y_i, \theta)$
- Actualizar los parámetros como $\theta \leftarrow (\theta - \eta \widetilde{\nabla_\theta \mathcal{J}})$

fin

Normalmente, se realizan varias pasadas por toda la muestra, dividiéndola en r/m *minibatches* cada vez. Cada pasada sobre la muestra se denomina una *epoch*. Por otra parte, el *learning rate* representa la velocidad del paso de descenso; a mayor valor, más rápido entrena el algoritmo pero menos resolución tiene para encontrar mínimos. Finalmente, cabe resaltar que existen numerosas modificaciones de este algoritmo, muchas pueden encontrarse en la sección 8.3 de Goodfellow, Bengio y Courville (2016).

Adam

Adam (Kingma y Ba, 2017) es uno de los algoritmos más utilizados en redes neuronales modernas ya que produce muy buenos resultados. Se basa en dos modificaciones del algoritmo de MBSGD *AdaGrad* (Duchi, Hazan y Singer, 2011) y RMSProp (Tieleman, Hinton et al., 2012). Utiliza un *learning rate* adaptativo, es decir que cambia en cada paso del algoritmo, obtenido a través de una estimación mediante una media móvil de los dos primeros momentos del gradiente. Esta modificación intenta reducir la variabilidad de las direcciones de descenso por el gradiente, mejorando la velocidad de convergencia. Sin embargo, estas medias móviles pueden causar un sesgo al inicializarse (ya que se inician en 0) pero *Adam* incorpora un parámetro para reducir este sesgo. El pseudocódigo del algoritmo puede verse en 2.

Algoritmo 2: Algoritmo Adam. El símbolo $\odot$ representa el producto de vectores por componentes, $\varepsilon = 10^{-8}$ es una constante para evitar problemas numéricos en divisiones cercanas a 0

Hiperparámetros: Tamaño del *minibatch* m, *learning rate* η, parámetros de decaimiento del momento ρ_1, ρ_2, criterio de parada.

Se inicializa θ aleatoriamente.

Se inicializa el estimador del primer momento $m_0 \leftarrow 0$

Se inicializa el estimador del segundo momento $v_0 \leftarrow 0$

Se inicializa el paso $t \leftarrow 0$

mientras *No se cumple criterio de parada* **hacer**

- Submuestreo de m datos de X con sus correspondientes respuestas de Y. $t \leftarrow t + 1$
- Calcular la estimación del gradiente $\widetilde{\nabla_\theta \mathcal{J}}_t = \frac{1}{m} \sum_{i=1}^{m} \nabla_\theta \mathcal{J}(x_i, y_i, \theta)$
- Se calcula la estimación del primer momento $m_t \leftarrow \rho_1 m_{t-1} + (1 - \rho_1)\widetilde{\nabla_\theta \mathcal{J}}_t$
- Se calcula la estimación del segundo momento $v_t \leftarrow \rho_2 v_{t-1} + (1 - \rho_2)\widetilde{\nabla_\theta \mathcal{J}}_t \odot \widetilde{\nabla_\theta \mathcal{J}}_t$
- Se calcula el primer momento con la corrección de sesgo $\hat{m}_t \leftarrow m_t/(1 - \rho_1^t)$
- Se calcula el segundo momento con la corrección de sesgo $\hat{v}_t \leftarrow v_t/(1 - \rho_2^t)$
- Actualizar los parámetros como $\theta \leftarrow (\theta - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \varepsilon})$

fin

1.4.4. Regularización

Como vimos en la sección 1.4.1, el proceso de optimización de redes neuronales trata de minimizar una función de coste a través de un estimador que se calcula en base a los datos. Este conjunto de pasos que hemos visto tienen como resultado o bien reducir el error de optimización (al minimizar el coste empírico), o el error de aproximación (aumentando el número de capas y parámetros de la red). El resultado de esta optimización, debido al balance sesgo-varianza, es que en la mayoría de los casos se tenga un sobreajuste o *overfitting* con una varianza muy elevada y que se trata más de una interpolación de los datos que de un modelo. A efectos prácticos, esto se traduce en la minimización del error en la muestra de entrenamiento y un incremento en la de test.

Para solucionar esto y reducir el error de generalización, se han diseñado una serie de estrategias a las que nos referimos en general como regularización. Existe multitud de tipos de regularización

en redes neuronales, un recorrido más extenso sobre estos procedimientos se puede encontrar en el capítulo 7 de Goodfellow, Bengio y Courville (2016). En particular, incluimos en este trabajo un breve resumen del tipo de regularización que se ha utilizado en el proyecto: la penalización por norma.

La idea de regularización por norma se basa en penalizar la función de coste con una medida del tamaño de los pesos de la red. No es un tipo de regularización exclusivo de las redes neuronales, por ejemplo, para modelos de regresión, la regularización L_2 es la llamada regresión *ridge* (Goodfellow, Bengio y Courville, 2016:sec. 7.1.1). De forma general, la regularización por norma se pude resumir por la siguiente alteración del coste empírico:

$$\hat{\mathcal{R}}^{reg}(X, Y, \theta) = \hat{\mathcal{R}}(X, Y, \theta) + \alpha\Omega(\theta). \tag{1.22}$$

Con Ω una función de penalización y α un hiperparámetro que marca la fuerza de la regularización. Las dos opciones principales de regularización por norma son la regularización L_1 y L_2.

Si denotamos como $\mathbf{w} = (w_1, \dots w_S)$ al vector de todos los pesos de la red, es decir, excluyendo de θ los sesgos $b_k^{(l)}$. La regularización L_1 se calcula como

$$\hat{\mathcal{R}}^{L_1}(X, Y, \theta) = \hat{\mathcal{R}}(X, Y, \theta) + \alpha\Omega(\theta) = \hat{\mathcal{R}}(X, Y, \theta) + \alpha \sum_{i=1}^{S} |w_i|. \tag{1.23}$$

En Goodfellow, Bengio y Courville (2016) pp. 235,236 puede verse con más detalle como en el proceso de optimizar $\hat{\mathcal{R}}^{L_1}(X, Y, \theta)$ mediante el descenso por su gradiente, la regularización L_1 provoca que una parte de los pesos se vayan acercando a cero, pudiendo ser posible que lleguen a anularse. Una red neuronal donde existe una gran cantidad de pesos igualados a cero se dice que es *sparse*. De esta forma, esta regularización reduce la cantidad de parámetros y, por tanto, el error de generalización.

Por otra parte, la regularización por norma L_2 se computa de la siguiente forma:

$$\hat{\mathcal{R}}^{L_2}(X, Y, \theta) = \hat{\mathcal{R}}(X, Y, \theta) + \frac{\alpha}{2}\mathbf{w} \cdot \mathbf{w}. \tag{1.24}$$

Puede verse también que la regularización L_2 hace que los pesos sean bajos y se acerquen a cero pero nunca sin alcanzarlo. Esta reducción del tamaño de los pesos tiene un efecto similar en la reducción de la varianza del modelo.

Finalmente, como anotación adicional, en este proyecto se hace uso de una técnica que también podría ser considerada como regularización: la parada temprana o *early stopping* que trata de parar el entrenamiento, es decir, dejar de suministrar *minibatches* a los algoritmos de optimización para cuando el error de validación deja de mejorar.

Capítulo 2

Modelos utilizados en el proyecto

En este capítulo se presentan los modelos y herramientas utilizados en el proyecto FARO para detectar casos anómalos de datos de los usuarios del cliente. Se incluyen dos modelos principales. En primer lugar, los *autoencoders*, una clase de redes neuronales que se fundamentan en la base teórica del capítulo 1 y que tienen multitud de aplicaciones, aunque en nuestro caso se utilizarán para la detección de anomalías. Por otra parte, se introduce un modelo de *machine learning* de bastante éxito en otros trabajos de detección de anomalías, el *isolation forest* (Liu, Ting y Zhou, 2008), que también se ha utilizado para el estudio de UEBA del proyecto. Finalmente, se añade una breve explicación de t-SNE (Maaten e Hinton, 2008), una técnica de representación de datos de muy alta dimensión, que ha sido utilizada en las primeras etapas del proyecto para visualizar el conjunto de datos.

2.1. *Autoencoders*

Los *autoencoders* son uno de los modelos con más desarrollo y mejores resultados en la bibliografía para detección de anomalías en datos de muy alta dimensión. En nuestro caso, los utizamos como modelo principal para el proyecto al que está vinculado este trabajo. La forma más general de definir un *autoencoder* es la que puede verse en Baldi (2012) como un modelo de aprendizaje automático que intenta aproximar la función identidad restringido por ciertas condiciones. En nuestro caso, nos restringiremos a dar una definición para el caso de un *autoencoder* para vectores aleatorios en $\mathbb{R}^n$:

Definición 2.1.1. Dada una muestra aleatoria $X = (\mathbf{x}_1, \ldots, \mathbf{x}_m)$ $\mathbf{x}_i \in \mathbb{R}^n$, un *autoencoder* $\mathbf{AE}_n^p$ es una tupla $n, p, f, g, \mathcal{E}, \mathcal{D}, X, \Delta$ donde:

- n, p son enteros positivos.
- $f \in \mathcal{E}$, $g \in \mathcal{D}$, donde $\mathcal{E}, \mathcal{D}$ son subconjuntos de funciones de $\mathbb{R}^n$ a $\mathbb{R}^p$ y de $\mathbb{R}^p$ a $\mathbb{R}^n$ respectivamente.
- Δ es una función de disimilitud en $\mathbb{R}^n$, usualmente una distancia.

Al espacio de llegada de la función f, $\mathbb{R}^p$ suele denominarse espacio latente, y sus imágenes $f(\mathbf{x})$ representaciones latentes de $\mathbf{x}$.

Definición 2.1.2. Se define el error de reconstrucción de un *autoencoder* como

$$E_{f,g}(X) = \sum_{i=1}^{m} \Delta(f \circ g(x_i), x_i). \tag{2.1}$$

Por otro lado, se puede definir el proceso de entrenar un *autoencoder* (*fitting*) como el proceso de encontrar las funciones f, g que mejor minimicen el error de reconstrucción.

Definición 2.1.3. Un *autoencoder* entrenado en una muestra X, es un *autoencoder* donde las funciones f y g cumplen:

$$(f, g) = \underset{(f,g)\in\mathcal{E}\times\mathcal{D}}{\arg\min}\ E_{f,g}(X) \tag{2.2}$$

Desde este marco, el *autoencoder* es una clase de objetos bastante amplia, cuyas aplicaciones y propiedes dependen de las restricciones que se impongan sobre $p, \mathcal{F}$, y $\mathcal{G}$ y de la Δ elegida. Por ejemplo en el caso de que $p = n$, tomando Δ como la norma euclídea en $\mathbb{R}^n$, y $\mathcal{F} = \mathcal{G} = \mathrm{End}(\mathbb{R}^n)$; el resultado de entrenar el *autoencoder*, no es otro que la función identidad, es decir $f, g = \mathrm{idd}$..

2.1.1. El *autoencoder* como red neuronal

Observando en detalle las definiciones anteriores, resulta sencillo implementar un *autoencoder* como una red neuronal de aprendizaje profundo. Detallamos en primer lugar los paralelismos entre ambos para detallar el proceso de construcción de un *autoencoder* con redes neuronales.

Para empezar, podemos tomar el primer conjunto de funciones del *autoencoder* como $\mathcal{E} = \mathcal{N}_{(\mathbb{R}^n,\mathbb{R}^p;\mathfrak{G},\mathcal{F})}$, es decir, el conjunto de redes neuronales con estructura dada por $\mathfrak{G} = (V, E)$ y activaciones $\mathcal{F}$ con entrada en el dominio del *autoencoder* y salida en el espacio latente. Por otro lado se toma el segundo conjunto de funciones como $\mathcal{D} = \mathcal{N}_{(\mathbb{R}^p,\mathbb{R}^m;\mathfrak{H},\mathcal{G})}$ con $\mathfrak{H} = (U, F) \mid U^{(1)} = V^{(l)}$. Es decir, limitamos al *autoencoder* a estar formado por dos redes neuronales $f \in \mathcal{E},\ g \in \mathcal{D}$ que reciben el nombre de codificador o *encoder* y decodificador o *decoder* respectivamente.

Además, es habitual limitar aún más la elección de funciones para dotar de una simetría al *autoencoder*, tomando $\mathcal{D} = \mathcal{N}_{(\mathbb{R}^p,\mathbb{R}^n;\mathfrak{G}',\mathcal{F})}$, donde $\mathfrak{G}' = (V', E')$ es el grafo inverso de $\mathfrak{G}$, definido como $V' = V$ y $E' = \{(i, j) \in V \times V \quad | \quad (j, i) \in E\}$. En nuestro caso siempre trataremos *autoencoders* con esta restricción.

Ya que las redes neuronales son composicionales, siguiendo la definición 1.1.1, la función $\mathbf{ae} = g \circ f,\ f \in \mathcal{E},\ g \in \mathcal{D}$ es también una red neuronal en $\mathcal{N}_{(\mathbb{R}^n,\mathbb{R}^n;\mathfrak{G}\cup\mathfrak{H},\mathcal{F}\cup\mathcal{G})}$. Asimismo, como red neuronal, la función que pretende aproximar es la función identidad, por lo que es fácil ver en las formulaciones 2.1.2 y 1.9 que el error de reconstrucción es equivalente al riesgo empírico y que Δ es equivalente a la función de pérdida $\mathcal{L}$, ya que al estar modelizando la función identidad en la red neuronal, los vectores respuesta $\mathbf{y}_i$ son los propios vectores de datos. De esta forma, es posible tratar el *autoencoder* como una única red neuronal y optimizarla usando los mismos procedimientos tratados en la sección 1.4.3.

Como ya se comentó en la definición de *autoencoders*, si no se imponen restricciones sobre el espacio latente o a los conjuntos de funciones, el resultado de entrenar el *autoencoder* es la propia

función identidad. De la misma forma, para *autonecoders* formados por redes neuronales puede verse rápidamente que existe una solución trivial al problema del *autoencoder* cuando $p \geq n$, que consistiría en una red neuronal con activaciones lineales y matrices de pesos identidad. Por esta razón, a los *autoencoders* se les imponen restricciones, siendo lo más común escoger $p < n$. Los *autoencoders* con esta propiedad se denominan *autoencoders* incompletos y son los que implementamos en este trabajo. La reducción de dimensión en el paso intermedio obliga al *autoencoder* a optimizar el *encoder* y el *decoder* para obtener el mejor proceso de compresión y descompresión de los datos en la representación latente. Además, en la estructura de la red neuronal, es común que la reducción de dimensionalidad se realice jerárquicamente, es decir, escogiendo las capas del *encoder* (y de forma indirecta del *decoder*) de forma que cada capa oculta tenga una dimensión menor a la anterior como puede verse en la figura 2.1.

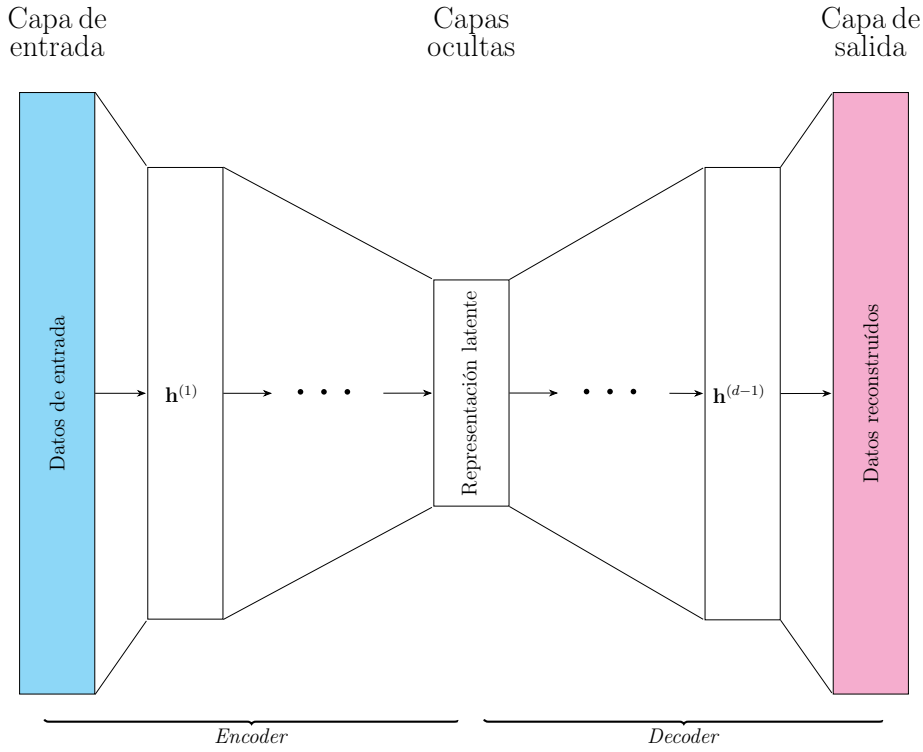


Figura 2.1: Esquema de la estructura de *autoencoder* con redes neuronales construida para el proyecto.

Una propiedad de interés de los *autoencoders* es que la representación latente es una codificación de los datos en dimensión reducida. Por esta razón, los *autoencoders* pueden utilizarse como un método de reducción de la dimensionalidad. Es más, puede verse en Plaut (2018) y en la sección 2.5 de Aggarwal (2019) que si se limita el autencoder a tener funciones de activación lineales y con pesos emparejados en el *encoder* y *decoder*, la solución con pérdida cuadrática que resuelve el *autoencoder* es equivalente a obtener una descomposición de la matriz de datos en valores singulares, o de obtener las primeras p componentes principales si se toman como entrada del *autoencoder* datos centrados en la media. Al añadir la no linealidad en las funciones de activación y la profundidad, se puede pensar en el *encoder* como un algoritmo de reducción de dimensión no lineal. Esto es una gran ventaja ya que el análisis de componentes principales no da resultados, por ejemplo, si los datos se encuentran embebidos en una variedad con curvatura en el espacio de altas dimensiones de los datos.

En los últimos años los *autoencoders* han sido utilizados para múltiples aplicaciones. Además de

como reducción de dimensionalidad, haciendo el papel de un análisis de componentes principales no lineal, los *autoencoders* pueden utilizarse como modelos generativos. Una vez entrenados, se utiliza el *decoder* sobre una muestra sintética de datos latentes con la que se obtienen datos similares a la muestra de entrenamiento. En este ámbito han aparecido varias modificaciones interesantes. Destacamos los *autoencoders variacionales* (VAE) (Kingma y Welling (2019)), que son *autoencoders* que intentan que la representación latente siga una distribución normal multivariable, con lo que es posible obtener nuevos datos muestreando esta distribución. Los VAE también son utilizados para detección de anomalías, por ejemplo en Xiao, Yan y Amit (2020) se construye una medida de la probabilidad de que un dato pertenezca a la misma distribución que la muestra de entrenamiento utilizando VAEs. Finalmente, cabe destacar un último caso de uso de los *autoencoders* que se encuentra en la reducción de ruido en imágenes, los llamados *denoissing autoencoders* que se entrenan modificando con un ruido los datos de entrenamiento de forma que el objetivo del *autoencoder* no es obtener una aproximación de la función identidad, sino de la función inversa a la función generadora del ruido. Los *denoissing autoencoders* son un ejemplo de *autoencoders* sobrecompletos, es decir, con $p \geq n$, ya que no modelan la función identidad y por tanto no requieren esa limitación en el tamaño de la capa.

En este trabajo, los *autoencoders* se utilizan para la detección de datos anómalos, lo cual resulta natural viendo su relación con el análisis de componentes principales.

2.1.2. Deteccion de anomalias con *autoencoders*

Al igual que otros modelos de reducción de dimensionalidad, los *autoencoders* incompletos pueden transformarse fácilmente en un modelo de detección de datos atípicos. La idea central es que debido a que los datos atípicos son poco comunes, el *autoencoder* priorizará al optimizar la reconstrucción de datos no anómalos ya que serán los que reduzcan más el riesgo empírico. De esta forma, tras entrenar el modelo, los datos atípicos no serán reconstruidos correctamente y se podrá utilizar el error de reconstrucción como un marcador de anomalía. Se puede decidir el límite de error de reconstrucción en base a un cuantil específico (un dato es anómalo cuando está, por ejemplo, en el percentil 95 de error) o a una desviación de un valor central (un dato es anómalo cuando tiene un error de k varianzas por encima de la media o k rangos intercuartílicos por encima de la mediana). Además, observando en cual de las variables se obtiene un peor error de reconstrucción, podemos obtener información de en qué dimensiones el dato atípico presenta más distinción. Esto es muy importante en aplicaciones reales porque da una intuición del origen de la anomalía tras la detección. Una gran ventaja de los *autoencoders* es que por su estructura, al ser la respuesta la propia entrada al modelo, se pueden utilizar sin datos etiquetados, lo cual los hace aún más útiles para los casos de grandes cantidades de datos.

Entre algunos ejemplos en los que se han utilizado algún tipo de *autoencoder* para detectar anomalías incluimos de nuevo el trabajo de Xiao, Yan y Amit (2020) que utiliza *autoencoders* variacionales. En Ribeiro, Lazzaretti y Lopes (2018) se propone una herramienta para detectar anomalías en una señal de vídeo, utilizando otra modificación de los *autoencoders*, los *autoencoders convolucionales*. En estos modelos el *encoder* y el *decoder* son redes convolucionales (Lecun et al., 1998), modelos muy usados en el tratamiento de imágenes. En Sakurada y Yairi (2014) se utilizan tanto *autoencoders* incompletos como sobrecompletos obteniendo mejores resultados que otros modelos de detección e anomalías en datos de satélites. Por otro lado, en Zhou y Paffenroth (2017) se utiliza un diseño muy similar al que presentaremos en el capítulo 3, con un *autoencoder* regularizado en norma. Finalmente, destacamos también el trabajo de Morales-Forero y Bassetto (2019) donde se propone un modelo mixto de *deep learning* para detección de anomalías compuesto por un *autoencoder* y una red recurrente LSTM.

2.2. *Isolation forest*

Otro de los modelos que aplicamos para detección de anomalías en este proyecto es el *Isolation forest* o *iforest* introducido por Liu, Ting y Zhou (2008). El *iforest* se considera un modelo de aprendizaje automático, pero no se trata de una red neuronal. Sin embargo, se ha decidido utilizar como modelo base para comparar los modelos de *deep learning* debido a tratarse de un algoritmo con buenos resultados en la bibliografía (Vitorino et al., 2021) y que se ha utilizado con éxito en otros proyectos de Gradiant.

A diferencia de otros modelos clásicos de detección de anomalías, el *isolation forest* no utiliza técnicas de análisis multivariable basadas en distancias o en *clustering*. Se fundamenta en la hipótesis de que los datos atípicos están "aislados" del resto de la muestra; esto es, son poco frecuentes y presentan mucha diferencia con respecto a los datos normales. Al tratar datos aislados, la idea del modelo es encontrar un método para separarlos del resto. El método se basa en realizar cortes en la muestra de datos utilizando hiperplanos de un árbol de decisión hasta que el árbol separe todos los datos en hojas o nodos terminales diferentes. Al terminar el proceso, como los datos anómalos deberían estar más alejados del resto, es esperable que el árbol de decisión los separe en menos cortes que el resto.

La construcción del árbol es sencilla. Primero, se escoge de forma aleatoria una variable i de la muestra de datos $X = \mathbf{x}_1 \dots \mathbf{x}_n$ y un valor de corte t para esa variable; así se construyen dos ramas en este paso: $\{\mathbf{x} \in X \mid \mathbf{x}_i < t\}$, $\{\mathbf{x} \in X \mid \mathbf{x}_i < t\}$. A continuación se repite el proceso anterior en cada rama hasta que los conjuntos en los nodos terminales del árbol solo contengan un dato. Una vez construido el árbol, se define la función longitud de camino *lambda* de forma que $\lambda(\mathbf{x})$ es el numero de aristas que hay que atravesar en el árbol para llegar a $\mathbf{x}$ desde el nodo raíz o, equivalentemente, el número de pasos en el algoritmo de construcción del árbol hasta que $\mathbf{x}$ se encuentre aislado.

Este proceso de construcción de un árbol de decisión se repite numerosas veces, obteniendo un conjunto de T árboles distintos que recibe el nombre de bosque. Con este bosque es posible obtener un estimador de la esperanza de la longitud de camino de cada punto $\overline{\lambda}(\mathbf{x})$ promediando los caminos a $\mathbf{x}$ de todos los árboles del bosque. Con este promedio se calcula una puntuación para cada dato llamada *anomaly score*

$$s(\mathbf{x}, n) = 2^{-\frac{\overline{\lambda}(\mathbf{x})}{c(n)}}, \tag{2.3}$$

donde $c(n)$ es la longitud de camino promedio para separar cada uno de los n datos.

De esta forma:

- Si todos los datos están uniformemente distribuidos y no hay datos anómalos, la longitud promedio de cada dato se acercará a la longitud promedio en el árbol, es decir $\lim_{T\to\infty} \overline{\lambda}(x) = c(n)$, y por tanto siguiendo la ecuación 2.3, $s(\mathbf{x}, n) = 1/2 \; \forall \; \mathbf{x} \in X$.
- Si un dato es muy anómalo, su longitud de camino será muy pequeña, de esta forma, $\lim_{T\to\infty} \overline{\lambda}(\mathbf{x}) = 0$ y por tanto $s(\mathbf{x}, n) = 1$.
- Si un dato alcanza en promedio la máxima longitud de camino posible, $\lim_{T\to\infty} \overline{\lambda}(\mathbf{x}) = n - 1$, se considera que es muy improbable de ser anómalo, con un *score* $s(\mathbf{x}, n) = 0$.

La ecuación 2.3 permite asignar una puntuación entre 0 y 1 a cada valor de la muestra, y se puede escoger un nivel de puntuación al partir del cual consideraremos un dato como atípico. En nuestro caso, se toma como anómalo los valores con puntuación de mas de 0,5, que es lo más habitual.

2.3. t-SNE

Finalmente, la última técnica que utilizamos en el tratamiento de los datos del proyecto es la herramienta *t-Stochastic Neighbor Embedding* (t-SNE), propuesta por Maaten e Hinton (2008) como herramienta de reducción de dimensionalidad diseñada expresamente para la visualización de datos con muchas variables. Al igual que la herramienta en la que se basa, *Stochastic Neighbor Embedding* (SNE), elabora una estimación de la densidad de los datos originales y construye una representación en 2 o 3 dimensiones que conserva las propiedades de esta densidad. En particular, t-SNE está pensado para intentar conservar la forma y estructura de *clustering* en datos multivariantes.

t-SNE comienza realizando una medida de similitud entre los datos. Para cada dato $\mathbf{x}_i$, se construye una gaussiana multivariante centrada en $\mathbf{x}_i$ y de varianza σ_i, esta gaussiana representa la probabilidad de que un punto en las proximidades de $\mathbf{x}_i$ se considere un vecino: a mayor distancia, menos probable que dos datos pertenezcan al mismo *cluster*. Esto se traduce a la probabilidad condicionada

$$p_{j|i} = \frac{\exp\left(-\left\|\mathbf{x}_i - \mathbf{x}_j\right\|^2 / 2\sigma_i^2\right)}{\sum_{k \neq j} \exp\left(-\left\|\mathbf{x}_i - \mathbf{x}_k\right\|^2 / 2\sigma_i^2\right)}. \tag{2.4}$$

De la cual se obtiene la similitud simetrizada

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n}, \tag{2.5}$$

siendo n el tamaño de la muestra X y fijando $p_{ii} = 0 \; \forall \; i$.

La varianza de la distribución asignada a cada punto se calcula en función de la densidad local de los datos. Se intenta que para regiones con muchos datos, σ_i sea pequeña y lo contrario para regiones poco pobladas. Para obtener estas varianzas, se utiliza un hiperparámetro llamado perplejidad que tiene un papel similar a la cantidad de vecinos en una agrupación de k-vecinos más próximos. Más información sobre el cálculo de σ_i en base a la perplejidad puede encontrarse en Maaten e Hinton (2008).

A continuación, para cada representación en bajas dimensiones, se computa una medida de similitud similar pero utilizando distribuciones t de Student con un grado de libertad para el cálculo de los entornos. Si $\mathbf{y}_i$ es la representación de $\mathbf{x}_i$, se obtiene la similitud entre $\mathbf{y}_i$ y $\mathbf{y}_j$ como

$$q_{ij} = \frac{\left(1 + \left\|\mathbf{y}_i - \mathbf{x}_j\right\|^2\right)^{-1}}{\sum_{k \neq j} \left(1 + \left\|\mathbf{y}_i - \mathbf{x}_j\right\|^2\right)^{-1}} \tag{2.6}$$

fijando $q_{ii} = 0 \; \forall \; i$.

Una vez obtenidas estas dos medidas, se comparan las distribuciones de probabilidad conjuntas de los datos originales P y las representaciones en menos dimensiones Q. Para esto se utiliza la divergencia Kullback-Leibler, que mide cuánto de distintas son ambas distribuciones

$$c = D_{KL}(P||Q) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{qij}. \tag{2.7}$$

Finalmente, el algoritmo intenta obtener la mejor representación en baja dimensión que minimice c utilizando técnicas de descenso por gradiente.

Capítulo 3

Aplicación de modelos de UEBA en el conjunto de datos de FARO

En esta sección mostramos la experiencia del desarrollo del proyecto al que ha estado vinculado este TFM y en el que se han utilizado las distintas técnicas y modelos introducidos en las secciones anteriores.

Como se trató en la introducción, este proyecto está enmarcado dentro del proyecto ÉGIDA, por lo que se trata de una colaboración del centro tecnológico con un cliente empresarial, para reforzar sus capacidades de ciberseguridad. En particular, nuestro cliente es una importante entidad bancaria que nos ha solicitado construir una herramienta para complementar sus controles existentes sobre los datos y sistemas que se encuentran en su red. El objetivo es detectar lo antes posible incidencias graves de seguridad que pueden tener como consecuencias el filtrado de información personal o confidencial, el secuestro o inutilización de sistemas o comprometer los datos con información empresarial confidencial. Estas incidencias ocurren en el nivel de los usuarios de la entidad, ya que son quienes tienen acceso a máquinas, sistemas y redes donde se encuentra esta información. Un atacante externo que tome el control de una de las cuentas de estos usuarios puede intentar obtener acceso a los datos, tomar el control de las máquinas y solicitar un rescate o realizar una escalada de privilegios para obtener acceso a otros usuarios con un mayor nivel de seguridad. Por otro lado, los propios usuarios pueden suponer riesgos: tanto por errores o negligencias, por ejemplo usuarios que utilizan dispositivos o cuentas personales para tratar datos sensibles; o usuarios maliciosos y posible espionaje industrial.

Para conseguir detectar estas incidencias, desde Gradiant se propone diseñar una herramienta basada en UEBA. Se utiliza la información agregada de la que dispone la entidad sobre el uso de sus sistemas y usuarios, lo que incluye datos sobre el uso de red, uso del correo electrónico, uso de los sistemas y otros. Estos datos se tratan con más detalle en la sección siguiente. Con esta información se elaboran modelos para identificar el comportamiento habitual de cada uno de los individuos de forma que, en el caso de una incidencia, pueda alertarse al equipo de seguridad de la entidad para que un analista pueda verificar la causa de la alerta.

Debido a que este trabajo se ha realizado con los datos de un cliente de Gradiant, y en especial al tratarse de datos internos de una entidad bancaria, no se incluye en este documento toda la información

del conjunto de datos ni de la estructura de ciberseguridad de la entidad por razones de confidencialidad. Por otro lado, debido a las características del proyecto, no se han podido cumplir todos los objetivos. Varias de las pruebas y tests para evaluar el rendimiento de los modelos aún no se han podido realizar. Sin embargo, el proyecto continúa en ejecución y varias de las ideas y propuestas que el equipo está implementando en estos momentos o en el futuro próximo se pueden encontrar en las conclusiones al final de este capítulo, en la sección 3.5.

3.1. El *dataset* de FARO

Para este proyecto hemos obtenido acceso a un extenso conjunto de datos y métricas que la entidad ha puesto a nuestra disposición. Siguiendo la filosofía de UEBA, las fuentes de estos datos son muy diversas, lo que ayuda a dibujar un perfil más completo de los usuarios que se intentan modelizar.

Debido a la alta cantidad de datos que se generan continuamente por los usuarios y a las necesidades de los algoritmos de recibir variables de un tipo especifico, fue necesario realizar un gran preprocesado. Este incluye la creación de variables secundarias en base a los datos originales (por ejemplo, tiempo medio entre *logins* conociendo los tiempos en los que se realizan), el tratamiento e imputación de datos no disponibles y la codificación para poder aplicar los modelos.

Parte de este procesado se ha realizado utilizando la herramienta de administración de datos *Splunk*[1] y del entorno de *Splunk* que nos ha habilitado el cliente.

3.1.1. Origen de datos, almacenamiento y preprocesado con *Splunk*

Splunk Enterprise es un producto de análisis y gestión de datos diseñado para el procesamiento de datos masivos, enfocado a la ingesta desde múltiples entradas (máquinas, dispositvos, sensores, etc.). Además, incluye una aplicación web desde la cual se puede obtener acceso a los datos indexados, realizar consultas, aplicar modelos y construir reportes de inteligencia de negocio.

Internamente, *Splunk* almacena todos los datos que obtiene como datos de series de tiempo, asignando un *timestamp* a cada entrada, tomando el momento de obtención si esta no contiene ninguna marca de tiempo. Estos datos son recogidos internamente en *eventos* que *Splunk* almacena en *índices* y a continuación, comprime y almacena. Los índices son estructuras que apuntan a los datos comprimidos para conseguir un acceso dinámico. De esta forma, cuando se realiza una petición de datos a *Splunk*, solo es necesario buscar en el índice, siendo mucho más rápido que buscar en todos los ficheros almacenados.

Debido a esta estructura, *Splunk* no se puede considerar como una base de datos, sino como un motor de búsqueda. Sin embargo, a efectos prácticos de extracción y consulta, podríamos tratarlo como una base de datos NoSQL. La parte encargada de dar acceso a los datos es *Splunk Search*, a la que se puede acceder desde la aplicación web. Para realizar consultas, *Splunk* utiliza el *Search Processing Language* (SPL), un lenguaje de programación basado en la estructura de *pipes* de UNIX que permite las operaciones habituales de una base de datos (cálculos, uniones de tablas, exportación de datos, etc.).

[1] https://www.splunk.com/

La instalación del cliente de *Splunk Enterprise* incluye también *toolkits* para construcción de modelos de aprendizaje automático y de aprendizaje profundo. Se trata de aplicaciones basadas en contenedores *Docker*[2] (Merkel, 2014) accesibles desde la aplicación web, donde el usuario tiene acceso a un entorno de *python*[3] (Van Rossum y Drake, 2009) con *jupyter notebooks*[4] (Kluyver et al., 2016) y las principales librerias para este tipo de modelización, incluyendo *pytorch*[5] (Paszke et al., 2019), *keras*[6] (Chollet et al., 2015) y *tensorflow*[7](Martín Abadi et al., 2015). En estos contenedores es posible diseñar y entrenar un modelo que luego puede ser accedido en *Splunk Search*. De esta forma, una vez que se ha finalizado un modelo, este puede ser aplicado directamente sobre los datos como parte de una consulta; lo que dota de una gran capacidad operativa para realizar análisis de datos en tiempo real.

Internamente, la entidad almacena en *Splunk* datos de todas las máquinas en su red en diversos índices donde se organizan los datos. Entre otros, se recogen datos del *log* de eventos de Windows (*WinEventLog*), del *log* del sistema en dispositivos UNIX (*syslog*), datos de red, alertas del antivirus, alertas del firewall, información sobre los correos electrónicos enviados y recibidos, y otros.

Cada instancia (una alerta de un antivirus o un evento de Windows, por ejemplo) es recogida en estos índices como un evento de *Splunk*, asignándole una marca de tiempo. Cada evento incluye multitud de otras variables que dependen de la clase de evento almacenado. Entre otras podemos encontrar información del usuario y la máquina que han generado el evento, así como información específica de esa instancia (causa de la alerta, código de error, etc).

3.1.2. Preprocesado de datos

Para este trabajo se han seleccionado los datos de una oficina específica de la entidad bancaria, la cual cuenta con cuatro usuarios distintos. Además, se añade un usuario del equipo técnico de la entidad. Con este usuario se pueden obtener datos anómalos con mayor facilidad, ya que tiene la posibilidad de simular incidencias sin afectar al trabajo de la oficina. Los datos de estos usuarios son generados por cada una de las máquinas en las que ingresan los usuarios, y recogidos por *Splunk*. Esto incluye los sistemas de la oficina y todos los dispositivos externos que se conectan a la red interna (por ejemplo, dispositivos móviles). Estos últimos están codificados como un único sistema (sistema 0, o externo) por razones de privacidad.

El conjunto de datos se obtiene de combinar una serie de consultas a los índices correspondientes a los eventos de Windows, los correos electrónicos, los datos del antivirus y del *firewall*. Dado que cada evento ocurre en un momento concreto y la gran cantidad de información, se condensan los datos en una serie de variables resumen agregadas por hora, que fueron escogidas por ser representativas de la clase de anomalías que se intentan detectar (intento de intrusión externa, ataques de toma de control de usuarios, filtrado de información al exterior, entre otros). Así, en el conjunto de datos, cada fila se corresponde con un vector de datos que recoge todos los eventos realizados por un usuario en una máquina determinada durante una hora. El conjunto de estas variables, así como una pequeña explicación de la información que recogen puede verse en el cuadro 3.1.

[2] https://www.docker.com/
[3] https://www.python.org/
[4] https://jupyter.org/
[5] https://pytorch.org/
[6] https://keras.io/
[7] https://www.tensorflow.org/

Cuadro 3.1: Variables recogidas en el conjunto de datos de FARO.

Variable	**Descripción**	**Tipo**
`time`	Fecha y hora en la que se generó el dato.	fecha
`CallerUser`	Nombre de usuario.	factor
`WorkstationName`	Nombre de la máquina donde tuvo lugar el dato.	factor
`num_new_process`	Número de veces que el usuario creó un nuevo proceso en la máquina.	numérico
`num_logins`	Número de veces que un usuario ingresa correctamente en la máquina.	numérico
`avg_seconds_between_logins`	Tiempo promedio en segundos entre cada uno de los ingresos correctos. Si el numero de ingresos es menor que dos, se imputa 3600s.	numérico
`num_failed_logins`	Número de veces que un usuario realiza un intento fallido de ingreso en la máquina.	numérico
`avg_seconds_between_failed_logins`	Tiempo promedio en segundos entre cada uno de los intentos falidos de ingreso. Si el numero de intentos es menor que dos, se imputa 3600s.	numérico
`num_registers_modified`	Número de veces en las que se ha modificado un valor del registro de Windows.	numérico
`num_antivirus_alerts`	Número de incidencias detectadas por el antivirus de la entidad.	numérico
`num_firewall_alerts`	Número de incidencias detectadas por el antivirus de la entidad.	numérico
`sent_emails`	Número de correos enviados por el usuario.	numérico
`received_emails`	Número de correos recibidos por el usuario.	numérico

`incident_emails`	Número de correos detectados como incidencias por el servicio de correo electrónico.	numérico
`sent_emails_size`	Tamaño de correos enviados por el usuario, incluyendo el cuerpo y los adjuntos.	numérico
`received_emails_size`	Tamaño de correos recibidos por el usuario, incluyendo el cuerpo y los adjuntos.	numérico
`total_email_files`	Numero de ficheros adjuntos en los correos del usuario.	numérico
`sent_email_files`	Numero de ficheros adjuntos en los correos enviados por el usuario.	numérico
`received_email_files`	Numero de ficheros adjuntos en los correos recibidos por el usuario.	numérico
`total_email_links`	Numero de enlaces web los correos del usuario.	numérico
`sent_email_links`	Numero de enlaces web los correos enviados por el usuario.	numérico
`received_email_links`	Numero de enlaces web los correos recibidos por el usuario.	numérico
`err_power`	Error de *Powershell*.	numérico
`com_power`	Lanzado un comando en *Powershell*.	numérico
`com_rem_power`	Lanzado un comando remoto en *Powershell*.	numérico
`user_alerts`	Otras alertas detectadas sobre el usuario.	numérico
`workstation_alerts`	Otras alertas detectadas sobre el sistema.	numérico

Como se comentó con anterioridad, en este proyecto se incluyen los datos de una oficina que engloba a cuatro usuarios y un usuario extra al que tiene acceso el equipo técnico de la entidad, y donde se pueden realizar actividades sospechosas o anómalas para probar si los modelos trabajan correctamente. En el futuro, se espera extender la herramienta a otras oficinas con otros tipos de usuarios y que se pueda extender a toda la organización.

Por otro lado, trabajadores de la oficina pueden también agrupase según su puesto de trabajo. Esto

se realiza según la suposición de que trabajadores con un mismo rol tienen comportamientos similares (un comercial envía gran cantidad de correos externos, un técnico se conecta a muchas máquinas varias veces, etc.). La principal ventaja de esto es poder construir modelos con más datos. Este enfoque se mantiene dentro de la filosofía UEBA, ya que los roles de los usuarios se entienden como entidades. La asignación de usuarios y roles se hizo según el cuadro 3.2.

Cuadro 3.2: Usuarios en el conjunto de datos de la entidad y su agrupación por rol

Rol	**Usuario**	**Tamaño de muestra**
Gestión	`gestion_1`	4957
	`gestion_2`	3114
Dirección	`subdirecctor`	2032
	`direcctor`	1149
Usuario de test	`Técnico`	3101
TOTAL		14353

Sobre estos datos se realiza un preprocesado para que puedan ser modelizados. Se tomaron los siguientes pasos:

- Se eliminó la variable de tiempo ya que no se considera la autocorrelación o la dependencia temporal. Esta decisión no está fundamentada en la información que tenemos de los datos, es muy probable que existan patrones temporales (*logins* en horas determinadas, horas sin actividad fuera de el horario laboral, etc.), sin embargo, como el proyecto está aún en sus comienzos, se propuso utilizar modelos sin autocorrelación como el mínimo producto viable original. Otros modelos más complejos se propondrán más adelante. Varias de estas propuestas se recogen en la sección 3.5 de trabajo futuro.
- Se codifica la variable categórica `WorkstationName` según el procedimiento *one-hot* convirtiendo cada nivel del factor en una nueva variable binaria que vale 1 cuando el dato se corresponde a esa máquina.
- Se separan los datos que tengan algún tipo de incidencia (mostrado en las variables con sufijo `_alerts` exceptuando `num_firewall_alerts`). Se eliminan estas variables. La razón de no eliminar `num_firewall_alerts` es debido a que un usuario de una oficina puede realizar de forma relativamente corriente incidencias de red (conectarse a redes sociales o a páginas de prensa desde un dispositivo de la oficina) sin que se trate de una incidencia de seguridad. En caso de un ataque, es posible que estas conexiones sean más elevadas y debería ser identificado por el modelo (por ejemplo, un *script* intentando conectarse a un servidor web externo o que redirige al usuario a una página del atacante).

- Se estandarizan los datos. En primer lugar, se centran y escalan los datos restando la mediana y dividiendo por el rango intercuartílico; se utiliza la función `sklearn.preprocessing.RobustScaler`[8] del paquete *scikit-learn* [9] (Pedregosa et al., 2011) de *python*.

  A continuación se utiliza `sklearn.preprocessing.MinMaxScaler`[10] para reescalar los datos para que todos los valores se encuentren entre 0 y 1.

- Se separan los datos por usuarios y por rol siguiendo el cuadro 3.2.

- Se separan los datos de los usuarios en conjuntos de entrenamiento (80 %) y test (20 %).

- Se obtiene la muestra final que contiene 14353 datos.

3.2. Procedimiento y limitaciones

Para modelizar el comportamiento de los usuarios, se aplican los modelos introducidos en el capítulo 2 de este trabajo: el *isolation forest* y varios *autoencoders*. Se realizan asimismo modelos por usuario, es decir, modelos ajustados solo con los datos de cada usuario, y modelos por rol siguiendo la estructura marcada en el cuadro 3.2. Para identificar la fiabilidad y utilidad de estos algoritmos de detección, nuestro objetivo era probarlos contra amenazas simuladas ofrecidas por la entidad. Estas amenazas serían verdaderos actos anómalos realizados por el usuario del equipo técnico en un entorno cerrado de la red de la entidad, incluyendo intentos de *login*, conexiones a redes externas o envío de correos anómalos o con gran cantidad de adjuntos. Finalmente estas anomalías se contrastarían con los modelos de cada usuario, suponiendo que simulan actividad no deseada de un atacante que ha usurpado su cuenta y en particular con un modelo creado sobre el propio usuario técnico, donde sería un caso más representativo del uso real de la herramienta.

Sin embargo, por limitación de tiempo, no ha sido posible obtener estos datos de amenazas antes de la finalización de este trabajo, por lo que no se han podido incluir. En su lugar, mientras se obtienen estos datos, se propuso como evaluación dentro del enfoque UEBA ver el rendimiento de cada modelo cuando se expone a datos de otros usuarios de la entidad, para ver si la modelización por usuario tiene sentido. Este diseño presenta ciertas particularidades con respecto a un test directo contra anomalías ya que, aunque se van a evaluar los datos de otros grupos como anómalos, esto no siempre tiene porqué ser correcto. No tenemos garantías de que la capacidad de detectar otro usuario demuestre capacidad de los modelos contra un ataque. Por otro lado, veremos en esta sección que en algunos casos los comportamientos de los usuarios son similares, sobre todo entre miembros del mismo rol. Por dar un ejemplo, supongamos el hecho de que al entrar en la oficina, dos personas ingresen en su cuenta en un servidor remoto, si ninguno de los dos realiza ninguna otra acción de interés en una hora, los datos generados en esta hora por estos dos usuarios serán idénticos: un *login* correcto en una máquina. Por esta razón, el número de falsos positivos puede ser una sobreestimación y se debería tratar con cuidado. En el análisis descriptivo que tratamos a continuación podemos ver además que, si bien una parte de los datos se corresponden con comportamientos similares de dos o más usuarios, existe una gran cantidad de datos con comportamientos muy diferentes que el modelo debería ser capaz de identificar como anómalo. Finalmente, destacamos también que este proyecto se ha realizado con técnicas de aprendizaje estadístico no supervisado, como se vio en el capítulo 3; sin embargo, en el entrenamiento de los modelos se asume que los datos son principalmente datos de comportamiento normal, ya que según la información del cliente, no se corresponden con ninguna incidencia. Por precaución se ha

[8] https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.RobustScaler.html
[9] https://scikit-learn.org/stable/index.html
[10] https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html

asumido que puede existir hasta un 5 % de datos anómalos en la muestra de entrenamiento que se seleccionan como los datos con mayor error durante el ajuste. Podría decirse por tanto que las técnicas aplicadas se corresponden a un aprendizaje semisupervisado, donde no se parte de datos etiquetados, pero sí de una cierta información sobre los conjuntos de datos.

3.3. Estudio descriptivo de los datos

Comenzamos mostrando un pequeño análisis descriptivo de la muestra de datos para extraer información de las variables, y las condiciones a las que nos enfrentamos en este trabajo. Debido a la gran cantidad de variables, se incluyen únicamente una pequeña cantidad de ideas y representaciones gráficas. En la figura 3.1 se muestra una representación gráfica de los valores de las variables numéricas del cuadro 3.1.

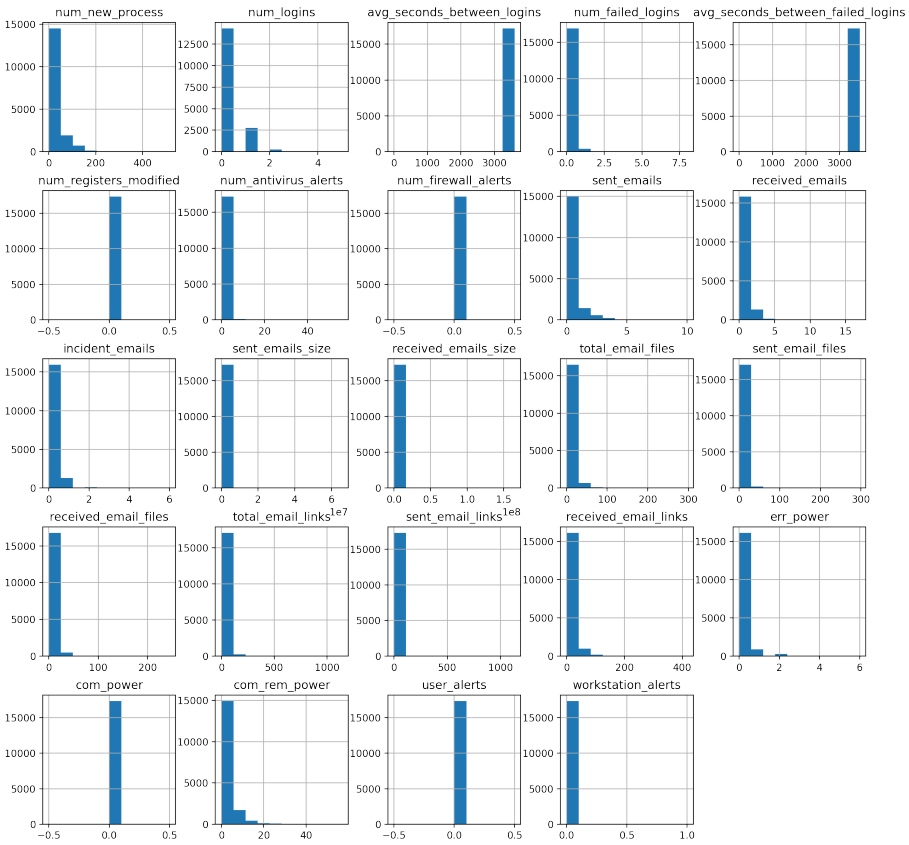


Figura 3.1: Variables del modelo.

Se puede apreciar que los datos son muy asimétricos: en la mayoría de variables se concentran los datos en un valor y en los valores próximos. Una de las preocupaciones que se tuvo al diseñar el modelo es la importancia de la variable `WorkstationName`, es decir, la máquina que genera el dato. Si suponemos un ejemplo de una oficina en la que cada usuario sólo trabaja con un sistema, podría ocurrir que los usuarios pueden clasificarse utilizando únicamente esta herramienta y el resto de variables serían superfluas. En la figura 3.2 se recogen el número de datos de cada máquina correspondiente a cada usuario. Podemos ver que, aunque los usuarios tienen un sistema principal, no es el único que utilizan e, incluso, que varios usuarios utilizan el mismo sistema en numerosas ocasiones, por lo que hay menos riesgo de que el modelo se limite a clasificar usuarios según la máquina que utilizan.

A continuación en las figuras 3.3 y 3.4 incluimos las visualizaciones del conjunto de datos utilizando t-SNE. Para la elaboración de esta representación se ha tomado como valores de hiperparámetros `perplexity` $= 90$, `n_iter` $= 2000$. Donde el `n_iter` es el número de iteraciones en la optimización de t-SNE. Al tratarse de una técnica para visualización de datos, los hiperparámetros de t-SNE se escogieron por ensayo y error. Siguiendo Maaten e Hinton (2008), la representación mejora habitualmente con mayor número de iteraciones, por lo que se ha seleccionado una cantidad suficiente sin requerir mucho tiempo de cómputo. Por otro lado, también en Maaten e Hinton (2008) se sugiere que `perplexity` se tome entre 5 y 50, mientras que en Wattenberg, Viégas y Johnson (2016) se sugiere que, para muestras con muchos datos, valores de perplejidad mayores de 50 dan lugar a mejores representaciones; lo que coincide con nuestras observaciones.

En la figura 3.4 podemos ver en detalle cada usuario por separado. En ambas figuras 3.3 y 3.4, t-SNE representa clústeres de datos, la interpretación de estos clústeres es que se corresponden con comportamientos o actividades de los usuarios. Con esto, se deduce que, como se adelantó en la sección 3.2, existen comportamientos compartidos por los usuarios. Sin embargo, también se pueden apreciar regiones separables del resto de datos que se corresponden a comportamientos distintivos de los usuarios. El objetivo de nuestros modelos al tratar los datos de otro usuario como anómalos será separar estos comportamientos.

3.4. Aplicación de los modelos

En esta sección presentamos los resultados de los modelos entrenados con la muestra de datos. Se realizaron dos planteamientos con el conjunto de datos; en primer lugar, agrupando los usuarios según el cuadro 3.2 y , a continuación, creando un modelo directamente por cada usuario.

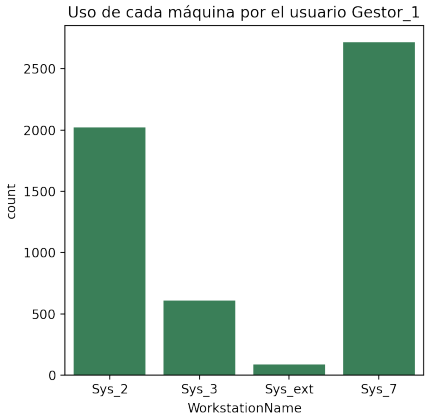
Para cada modelo se muestran los resultados obtenidos sobre la muestra de datos y una serie de métricas de evaluación de clasificación para cuantificar su capacidad (Fernández Casal, Costa Bouzas y Oviedo de la Fuente, 2021). Consideramos como positivo un dato anómalo, que en nuestro caso es un dato de otro rol o usuario. Se incluye el número de verdaderos y falsos positivos (TP,FP), verdaderos y falsos negativos (TN,FN), la tasa de verdaderos positivos (TPR), también llamada sensibilidad, dada por

$$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}, \tag{3.1}$$

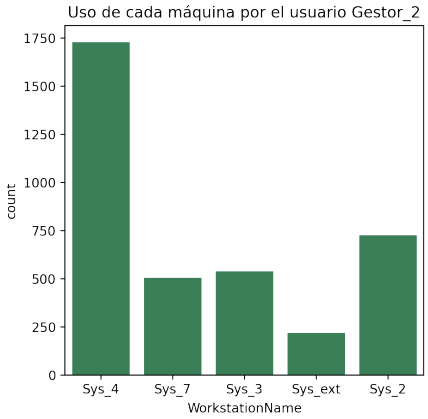
la tasa de verdaderos negativos (TNR) o especificidad

$$\mathrm{TNR} = \frac{\mathrm{TN}}{\mathrm{TN} + \mathrm{FP}}, \tag{3.2}$$

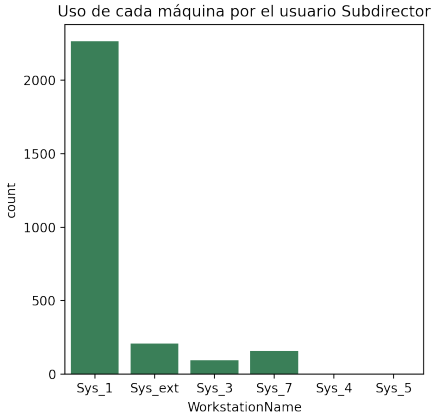
las tasas de falsos negativos y positivos ($\mathrm{FNR} = 1 - \mathrm{TPR}$, $\mathrm{FPR} = 1 - \mathrm{TNR}$), la precisión o valor



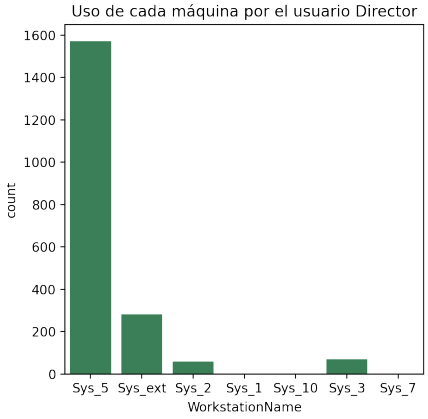
(a) Sistemas de `Gestor_1`



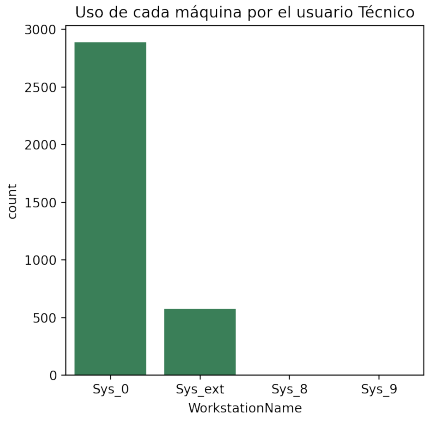
(b) Sistemas de `Gestor_2`



(c) Sistemas de `Subdirector`



(d) Sistemas de `Director`



(e) Sistemas de `Técnico`

Figura 3.2: Conteo de datos en cada sistema por usuario

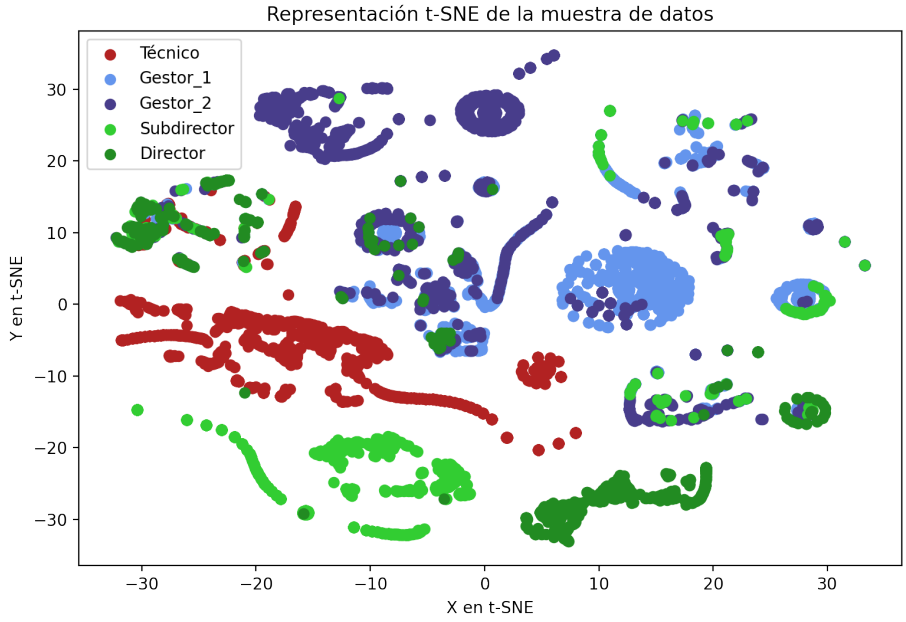


Figura 3.3: Representación t-SNE de los datos.

predictivo positivo (PPV)

$$\mathrm{TNR} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}} \tag{3.3}$$

y finalmente, la métrica F_1 como medida de precisión global

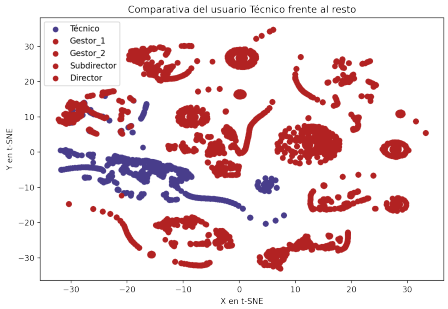
$$\mathrm{F}_1 = \frac{2\mathrm{TP}}{2\mathrm{TP} + \mathrm{FP} + \mathrm{FN}} \tag{3.4}$$

que se ha preferido a la tasa de aciertos (ACC) por el desbalanceo entre los conjuntos de datos que consideramos positivos y negativos.

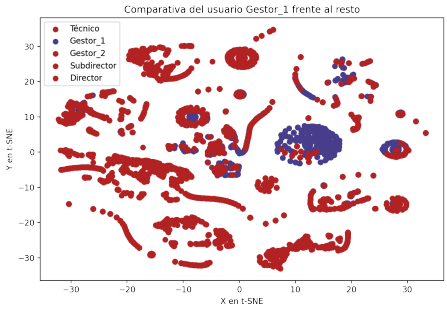
3.4.1. Modelizado usando Autoencoders

Presentamos aquí los resultados de los *autoencoders*. En el entrenamiento de cada modelo, se han utilizado los datos libres de alertas del usuario o rol correspondiente. De estos se ha muestreado un 80 % de los datos para entrenamiento, como se marca en la sección 3.1.1. Los datos de entrenamiento se subdividen de nuevo obteniendo un 20 % de datos de validación que se utilizan para computar el límite de error de reconstrucción a partir del cual se obtiene una anomalía. Los datos de entrenamiento restantes se subdividen una vez más, utilizando un 20 % como datos para validar la selección de hiperparámetros. A este conjunto de datos los denominamos datos de *grid validation*.

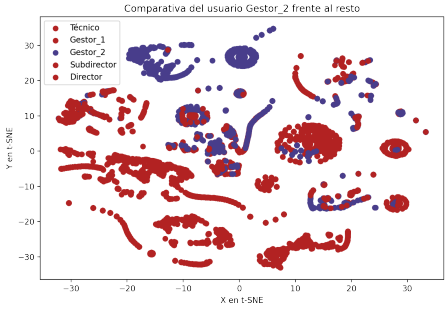
Para la selección de hiperparámetros se ha realizado una búsqueda en una cuadrícula de hiperparámetros escogiendo el modelo que minimiza el error de reconstrucción de los datos de validación de *grid validation*. Para evitar la variabilidad debida a la inicialización aleatoria de los pesos en el



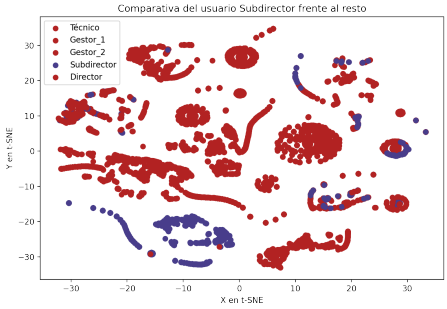
(a) Comparación de `Técnico` contra el resto.



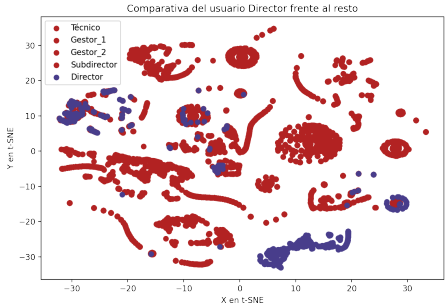
(b) Comparación de `Gestor_1` contra el resto.



(c) Comparación de `Gestor_1` contra el resto.



(d) Comparación de `Subdirector` contra el resto.



(e) Comparación de `Director` contra el resto.

Figura 3.4: Detalle de cada usuario en la representación t-SNE.

entrenamiento, la búsqueda utiliza validación cruzada promediando el error de reconstrucción de 5 repeticiones.

Para todos los modelos se ha seleccionado una estructura fija del *autoencoder*, con 5 capas ocultas y *encoder* y *decoder* simétricos. La estructura del *encoder* puede verse en el cuadro 3.3 y del *decoder* en el cuadro 3.4. Se ha escogido esta estructura en base a otros modelos similares diseñados para otros estudios de UEBA de Gradiant y por los resultados experimentales con otros modelos. En particular, se probaron modelos con menos capas, que resultaron demasiado simples y no llegaron a separar correctamente los datos, y modelos con activación RELU en las capas internas, que resultaron muy propensos a la muerte RELU. Ya que los modelos no tenían un tiempo de entrenamiento muy elevado (del orden de 15 minutos), se optó por activaciones ELU para mejorar este aspecto.

Cuadro 3.3: Estructura del *encoder*

	Capa de entrada	1ª Capa oculta	2ª Capa oculta	3ª Capa oculta	4ª Capa oculta	5ª Capa oculta	Capa de salida (Capa latente)
Dimensión	29	29	24	19	14	9	4
Activación	N/A	tanh	elu	elu	elu	elu	tanh

Cuadro 3.4: Estructura del *decoder*.

	Capa de entrada (Capa latente)	1ª Capa oculta	2ª Capa oculta	3ª Capa oculta	4ª Capa oculta	5ª Capa oculta	Capa de salida
Dimensión	4	4	9	14	19	24	29
Activación	N/A	tanh	elu	elu	elu	elu	tanh

Además, se ha escogido el algoritmo de optimización *Adam*, y regularización L_1 con *early stopping*. Estas elecciones parten de nuevo de la experiencia de Gradiant en otros proyectos con *autoencoders*.

Nótese que en la evaluación se utiliza la partición de test para probar el rendimiento del modelo con datos de su mismo usuario o rol. Sin embargo, para probar la capacidad del modelo con datos que no son de su propia clase, se han utilizado todos los datos de otros usuarios/roles, ya que el modelo no tiene acceso a estos datos en el entrenamiento.

Modelos por rol

En primer lugar, se muestran los *autoencoders* entrenados con los datos agrupados por rol. En el cuadro 3.5 se encuentran los datos detectados como anómalos desagregados por rol; es decir, cuántos datos de cada rol detecta el modelo como anómalo. Así se puede ver la diferencia en capacidad del modelo contra los distintos conjuntos de datos. En el cuadro 3.6 se encuentran métricas de evaluación de la clasificación de datos anómalos considerando los datos de otros usuarios como objetivo a detectar,

como se discutió en la sección 3.2 y al comienzo de esta sección. Los hiperparámetros escogidos en el entrenamiento y los valores de error obtenidos fueron los siguientes:

- **Modelo de Dirección**
  - *Hiperparámetros de Adam*:
    - Tamaño del *minibatch*: $m = 64$.
    - *Learning rate*: $\eta = 0{,}001$.
    - Parámetros de decaimiento del momento: $\rho_1 = 0{,}9$, $\rho 2 = 0{,}999$
    - Numero de *epochs*: 500
  - *Hiperparámetros de Regularización*:
    - Escala de la regularización L_1: $\lambda = 0{,}0001$
    - Parada por *early stopping*: No
  - *Riesgo empírico minimizado*:
    - ECM, mínimo alcanzado en entrenamiento: $8{,}97 \times 10^{-4}$
    - ECM, mínimo alcanzado en test: 2×10^{-3}
- **Modelo de Gestión**
  - *Hiperparámetros de Adam*:
    - Tamaño del *minibatch*: $m = 256$.
    - *Learning rate*: $\eta = 0{,}01$.
    - Parámetros de decaimiento del momento: $\rho_1 = 0{,}9$, $\rho 2 = 0{,}999$
    - Numero de *epochs*: 500
  - *Hiperparámetros de Regularización*:
    - Escala de la regularización L_1: $\lambda = 0{,}0001$
    - Parada por *early stopping*: No
  - *Riesgo empírico minimizado*:
    - ECM, mínimo alcanzado en entrenamiento: $4{,}20 \times 10^{-4}$
    - ECM, mínimo alcanzado en test: $5{,}09 \times 10^{-4}$

Cuadro 3.5: Clasificación usando los modelos por rol

	Datos de test del propio rol		Datos de test del otro rol		Datos de test del usuario "Técnico"	
	Clasificado como anom.	TOTAL	Clasificado como anom.	TOTAL	Clasificado como anom.	TOTAL
Direccion	14	637	1824	8071	2578	3101
Gestión	100	1615	2694	3181	2691	3101

Podemos ver en el cuadro 3.6 que con modelo de gestión se obtienen buenos resultados, detectando más del 85 % del resto de datos. El comportamiento de Dirección es mucho peor, principalmente en la detección del usuario de gestión (ver cuadro 3.10), teniendo muchos falsos negativos para este rol pero detectando correctamente el usuario técnico de la entidad.

Cuadro 3.6: Evaluación de los modelos por rol.

	TP	TN	FP	FN	TPR	TNR	FPR	FNR	PPV	F1
Direccion	4402	623	14	6770	0,394	0,978	0,022	0,606	0,997	0,565
Gestión	5385	1515	100	897	0,857	0,938	0,062	0,143	0,982	0,915

Modelos por usuario

En los cuadros 3.7 y 3.8 se muestran los resultados análogos al apartado anterior. Mostramos también los hiperparámetros escogidos para estos modelos y los valores de error obtenidos fueron los siguientes:

- **Modelo de `Gestor_1`**
  - *Hiperparámetros de Adam*:
    - Tamaño del *minibatch*: $m = 64$.
    - *Learning rate*: $\eta = 0{,}0001$.
    - Parámetros de decaimiento del momento: $\rho_1 = 0{,}9,\ \rho 2 = 0{,}999$
    - Numero de *epochs*: 250
  - *Hiperparámetros de Regularización*:
    - Escala de la regularización L_1: $\lambda = 0{,}00001$
    - Parada por *early stopping*: No
  - *Riesgo empírico minimizado*:
    - ECM, mínimo alcanzado en entrenamiento: $1{,}96 \times 10^{-3}$
    - ECM, mínimo alcanzado en test: $1{,}80 \times 10^{-3}$
- **Modelo de `Gestor_2`**
  - *Hiperparámetros de Adam*:
    - Tamaño del *minibatch*: $m = 64$.
    - *Learning rate*: $\eta = 0{,}01$.
    - Parámetros de decaimiento del momento: $\rho_1 = 0{,}9,\ \rho 2 = 0{,}999$
    - Numero de *epochs*: 500
  - *Hiperparámetros de Regularización*:
    - Escala de la regularización L_1: $\lambda = 0{,}00001$
    - Parada por *early stopping*: No
  - *Riesgo empírico minimizado*:
    - ECM, mínimo alcanzado en entrenamiento: $6{,}28 \times 10^{-4}$
    - ECM, mínimo alcanzado en test: $8{,}17 \times 10^{-4}$

- **Modelo de `Subdirector`**
  - *Hiperparámetros de Adam*:
    - Tamaño del *minibatch*: $m = 64$.
    - *Learning rate*: $\eta = 0{,}01$.
    - Parámetros de decaimiento del momento: $\rho_1 = 0{,}9,\ \rho 2 = 0{,}999$
    - Numero de *epochs*: 500
  - *Hiperparámetros de Regularización*:
    - Escala de la regularización L_1: $\lambda = 0{,}0001$
    - Parada por *early stopping*: No
  - *Riesgo empírico minimizado*:
    - ECM, mínimo alcanzado en entrenamiento: $4{,}33 \times 10^{-4}$
    - ECM, mínimo alcanzado en test: $1{,}51 \times 10^{-3}$
- **Modelo de `Director`**
  - *Hiperparámetros de Adam*:
    - Tamaño del *minibatch*: $m = 64$.
    - *Learning rate*: $\eta = 0{,}01$.
    - Parámetros de decaimiento del momento: $\rho_1 = 0{,}9,\ \rho 2 = 0{,}999$
    - Numero de *epochs*: 250
  - *Hiperparámetros de Regularización*:
    - Escala de la regularización L_1: $\lambda = 0{,}00001$
    - Parada por *early stopping*: No
  - *Riesgo empírico minimizado*:
    - ECM, mínimo alcanzado en entrenamiento: $1{,}31 \times 10^{-3}$
    - ECM, mínimo alcanzado en test: $2{,}41 \times 10^{-3}$

Cuadro 3.7: Clasificación usando los modelos por usuario.

	Datos de test del propio usuario		Datos de usuario Gestion 1		Datos de usuario Gestion 2		Datos de usuario Subdirector		Datos de usuario Director		Datos de usuario Técnico	
	Clasificado como anom.	TOTAL	Clasificado como anom.	TOTAL	Clasificado como anom.	TOTAL	Clasificado como anom.	TOTAL	Clasificado como anom.	TOTAL	Clasificado como anom.	TOTAL
Gestión1	52	992			1685	3114	1695	2032	908	1149	2626	3101
Gestión2	42	623	207	4957			1657	2032	899	1149	2615	3101
Subdirector	18	407	1915	4957	2297	3114			923	1149	2592	3101
Director	61	230	3586	4957	2382	3114	1814	2032			2596	3101

Cuadro 3.8: Evaluación de los modelos por usuario

	TP	TN	FP	FN	TPR	TNR	FPR	FNR	PPV	F1
Gestión1	6914	940	52	2482	0,736	0,948	0,052	0,264	0,993	0,845
Gestión2	5441	581	42	5798	0,484	0,933	0,067	0,516	0,992	0,651
Subdirector	10319	389	18	2002	0,838	0,956	0,044	0,162	0,998	0,911
Director	10378	169	61	2826	0,786	0,735	0,265	0,214	0,994	0,878

Vemos en ambas tablas que de nuevo los resultados son mejorables pero no desesperanzadores. En el cuadro 3.8 se puede ver que el F_1 alcanza valores por encima de 0,8 para `Gestión_1` y `Director` y, exceptuando el `Director`, todos tienen menos de un 10 % de falsos positivos. Esta proporción de falsos positivos se acerca a una cantidad suficiente para el uso en análisis de seguridad, ya que no generan demasiado ruido.

Por otro lado, se puede apreciar viendo las tablas desagregadas 3.5, 3.7 lo que ya se trató en la sección 3.2: la evaluación infraestima la capacidad del modelo. Un ejemplo de esto es el modelo de `Gestion_2`; en el cuadro 3.8 presenta muchos falsos negativos, con un FNR de más del 0,5, sin embargo, observando el cuadro 3.7 podemos ver que el modelo tiene problemas principalmente en la detección de `Gestion_1`, ya que muchos de los comportamientos de este usuario son similares a los de `Gestion_2`. Por esta razón, es necesario probar los modelos contra amenazas reales que podamos verificar que son realmente anómalos.

Otra característica importante a destacar, que puede verse tanto en los resultados por rol como por usuario, es que los modelos de los usuarios de Gestión tienen mejor rendimiento que los de Dirección. Por ejemplo, el modelo de `Subdirector` presenta muchos falsos negativos y el de `Director` una gran cantidad de falsos positivos, más de un 25 %, lo que lo haría inviable como herramienta de seguridad. La razón principal de esto es que los miembros del segundo grupo tienen una variedad mayor de comportamientos que los gestores y, además, realizan menos actividades monitorizables por el sistema, por lo que los modelos del rol de Direccion y de los usuarios que lo componen tienen muchos menos datos disponibles para entrenar. Por encima de esto, es posible ver que el modelo del rol Dirección se comporta peor que los modelos de los usuarios `Subdirector` y `Director` por separado. Por esto se ha tomado la decisión de que estos usuarios no comparten comportamientos lo suficiente como para ser considerados un mismo rol.

3.4.2. Modelizado usando *iforest*

Como base para comparar la capacidad de los *autoencoders*, se realizaron modelos utilizando *isolation forest*. De forma similar, se dividen los datos en muestra de entrenamiento (80 %) y test (20 %), y, de nuevo, los datos de entrenamiento en entrenamiento(80 % de la cantidad anterior) y validación(20 % de la cantidad anterior). Estos últimos se utilizan para escoger la cantidad de árboles mediante una búsqueda en cuadrícula de hiperparámetros con validación cruzada con 5 repeticiones; escogiendo el modelo que asocie la menor puntuación de anomalía a la muestra de validación. El resultado de esta búsqueda para cada modelo se puede ver en el cuadro 3.9. En este caso, no es necesario tener una muestra de validación para obtener el *threshold* ya que el algoritmo ya ofrece una puntuación de anomalía.

Cuadro 3.9: Número de árboles óptimo por modelo.

Usuario	T
Gestión	512
Dirección	8192
`gestion_1`	1024
`gestion_2`	1024
`subdirecctor`	1024
`director`	1024

Tratamos como datos anómalos aquellos con una puntuación de anomalía mayor de 0,5. Los resultados del *isolation forest* por rol se muestran en los cuadros 3.10 y 3.11; y por usuario en los cuadros 3.12 y 3.13.

Cuadro 3.10: Clasificación usando los modelos *iforest* por rol.

	Datos de test del propio rol		Datos de test del otro rol		Datos de test del usuario “Técnico"	
	Clasificado como anom.	TOTAL	Clasificado como anom.	TOTAL	Clasificado como anom.	TOTAL
Dirección	84	637	682	8071	348	3101
Gestión	250	1615	1138	3181	1309	3101

Cuadro 3.11: Evaluación de los modelos *iforest* por rol.

	TP	TN	FP	FN	TPR	TNR	FPR	FNR	PPV	F1
Dirección	1030	553	84	10142	0,092	0,868	0,132	0,908	0,925	0,168
Gestión	2447	1365	250	3835	0,390	0,845	0,155	0,610	0,907	0,545

Cuadro 3.12: Clasificación usando los modelos *iforest* por usuario.

	Datos de test del propio usuario		Datos de usuario Gestion 1		Datos de usuario Gestion 2		Datos de usuario Subdirector		Datos de usuario Director		Datos de usuario Técnico	
	Clasificado como anom.	TOTAL	Clasificado como anom.	TOTAL	Clasificado como anom.	TOTAL	Clasificado como anom.	TOTAL	Clasificado como anom.	TOTAL	Clasificado como anom.	TOTAL
Gestión1	176	992			923	3114	722	2032	720	1149	1672	3101
Gestión2	161	623	735	4957			543	2032	630	1149	1362	3101
Subdirector	85	407	835	4957	778	3114			609	1149	955	3101
Director	47	230	381	4957	326	3114	188	2032			398	3101

Cuadro 3.13: Evaluación de los modelos *iforest* por usuario

	TP	TN	FP	FN	TPR	TNR	FPR	FNR	PPV	F1
Gestión1	4037	816	176	5359	0,430	0,823	0,177	0,570	0,958	0,593
Gestión2	3270	462	161	7969	0,291	0,742	0,258	0,709	0,953	0,446
Subdirector	3177	322	85	9144	0,258	0,791	0,209	0,742	0,974	0,408
Director	1293	183	47	11911	0,098	0,796	0,204	0,902	0,965	0,178

Observando los resultados tanto por rol como por usuario, podemos ver que en este conjunto de datos los modelos de *isolation forest* tienen un rendimiento muy limitado.

En los modelos por rol, podemos ver en el cuadro 3.10 como ambos modelos tienen un rendimiento muy pobre. El modelo de Direccion únicamente detecta como dato anómalo menos del 10 % de los datos de gestión y al rededor de un 11 % para los datos del usuario técnico. El de gestión detecta una cantidad algo mayor (alrededor del 14 % de los datos de Direccion y del 42 % del usuario técntico) pero sin ser aceptable. Esta baja capacidad de detectar alertas se puede ver en el valor elevado del FNR en el cuadro 3.11; donde se ve que más del 90 % de los negativos del modelo de Direccion, el más afectado por este problema, deberían haberse sido positivos. Este rendimiento bajo puede verse también en los valores de F_1, mucho más bajos que en los modelos de *autoencoder* correspondientes.

Por otra parte, en los modelos por usuario, vemos resultados similares, también por debajo del *autoencoder*. Se puede ver que ninguno de estos modelos es aceptable, pero es destacable en los modelos de `subdirector` y `director` la muy baja puntuación F_1, de nuevo debido a la baja capacidad de detectar alertas. Una posible explicación de este peor rendimiento puede ser que el hecho de que se dispone de muchos menos datos con respecto a los otros dos usuarios. Finalmente, otro dato a destacar es que a diferencia de los *autoencoders*, viendo el cuadro 3.12 no parece que el rol tenga un efecto especial en el comportamiento de los modelos, por lo que en este caso no se está infraestimando tanto la capacidad de los modelos.

3.5. Conclusiones y trabajo futuro

En este trabajo se han aplicado dos tipos de modelos de estudio de comportamiento para detectar incidencias de seguridad en una entidad bancaria dentro del proyecto FARO. Los resultados aquí mostrados constituyen una prueba de concepto del uso de técnicas UEBA en este ámbito; obteniendo resultados que, si bien presentan margen de mejora, demuestran la utilidad de aplicar modelos de aprendizaje profundo para detección de este tipo de anomalías. Se ha podido ver como los modelos de

autoencoders son superiores a los basados en *isolation forest*, por lo que marcan el camino a seguir en el proyecto. Por otro lado se ha visto que la realización de modelos por rol no es beneficiosa y que debe reconsiderarse la agrupación de los individuos, sobre todo en el conjunto de Dirección. Destacamos de nuevo que estos resultados son todavía orientativos y que la evaluación de los modelos está limitada a la disponibilidad de los datos.

El equipo de FARO en Gradiant continuará trabajando en la línea de *autencoders* por usuario. Para mejorar los resultados obtenidos en este trabajo se han realizado diversas propuestas para llevar a cabo en el futuro. Por un lado, ya está en proceso una nueva evaluación de estos modelos contra una serie de amenazas simuladas por el equipo técnico de la entidad, lo cual dará una estimación de la capacidad real de los modelos. Además, se está valorando la incorporación de más variables a la base de datos que recojan más facetas del comportamiento de los usuarios y mejorar la capacidad de los modelos. También, se ha propuesto reducir el tiempo de agregado de datos a periodos más cortos que una hora para ampliar la granularidad de los datos. Finalmente, se ha propuesto utilizar modelos autorregresivos, ya que es de esperar que los datos de los usuarios presenten patrones diarios o semanales y que esto sea un factor importante a la hora de distinguir actividad maliciosa. Para esto se está valorando combinar los *autoencoders* con redes neuronales recurrentes como LSTMs (Hochreiter y Schmidhuber, 1997).

Bibliografía

Aggarwal, Charu C. (2019). *Neural networks and deep learning: A textbook*. Springer.

Baldi, Pierre (2012). «Autoencoders, Unsupervised Learning, and Deep Architectures». En: *ICML Unsupervised and Transfer Learning*.

Barron, A.R. (mayo de 1993). «Universal approximation bounds for superpositions of a sigmoidal function». En: *IEEE Transactions on Information Theory* 39(3), págs. 930-945. DOI: 10.1109/18.256500. URL: https://doi.org/10.1109%2F18.256500.

Berner, Julius et al. (2021). *The Modern Mathematics of Deep Learning*. arXiv: 2105.04026 [cs.LG].

Cano Córdoba, Felipe (2018). «Theoretical Study of Artificial Neural Networks». Bachelor's Thesis.

Chollet, Francois et al. (2015). *Keras*. URL: https://github.com/fchollet/keras.

Clevert, Djork-Arné, Thomas Unterthiner y Sepp Hochreiter (2016). *Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)*. arXiv: 1511.07289 [cs.LG].

Cser, Andras (2016). *Market overview: Security user behavior analytics (suba), 2016*. URL: https://www.forrester.com/report/market-overview-security-user-behavior-analytics-suba-2016/RES133332?objectid=RES133332.

Cybenko, G. (dic. de 1989). «Approximation by superpositions of a sigmoidal function». En: *Mathematics of Control, Signals, and Systems* 2(4), págs. 303-314. DOI: 10.1007/bf02551274. URL: https://doi.org/10.1007%2Fbf02551274.

Duchi, John, Elad Hazan y Yoram Singer (jul. de 2011). «Adaptive Subgradient Methods for Online Learning and Stochastic Optimization». En: *J. Mach. Learn. Res.* 12(null), págs. 2121-2159. ISSN: 1532-4435.

Fernández Casal, Rubén, Julián Costa Bouzas y Manuel Oviedo de la Fuente (2021). «1.3.5 Evaluación de un método de clasificación». En: *Aprendizaje Estadístico*, págs. 25-30. URL: https://rubenfcasal.github.io/aprendizaje_estadistico/.

Fukushima, Kunihiko (abr. de 1980). «Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position». En: *Biological Cybernetics* 36(4), págs. 193-202. DOI: 10.1007/bf00344251.

Gates, Carrie y Carol Taylor (2006). «Challenging the Anomaly Detection Paradigm: A Provocative Discussion». En: *Proceedings of the 2006 Workshop on New Security Paradigms*. NSPW '06. As-

sociation for Computing Machinery: Germany, págs. 21-29. ISBN: 9781595939234. DOI: 10.1145/1278940.1278945. URL: https://doi.org/10.1145/1278940.1278945.

Gilon, Yosefa et al. (2021). *Neural Networks Part 1: Setting up the Architecture*. URL: https://cs231n.github.io/neural-networks-1/.

Goodfellow, Ian, Yoshua Bengio y Aaron Courville (2016). *Deep learning*. MIT Press.

Hanin, Boris (oct. de 2019). «Universal Function Approximation by Deep Neural Nets with Bounded Width and ReLU Activations». En: *Mathematics* 7(10), pág. 992. ISSN: 2227-7390. DOI: 10.3390/math7100992. URL: http://dx.doi.org/10.3390/math7100992.

Hanin, Boris y Mark Sellke (2018). *Approximating Continuous Functions by ReLU Nets of Minimal Width*. arXiv: 1710.11278 [stat.ML].

Hastie, Trevor J (2017). *Generalized additive models*. Routledge.

Hebb, Donald Olding (1949). *The organisation of behaviour: a neuropsychological theory*. Science Editions New York.

Hochreiter, Sepp y Jürgen Schmidhuber (nov. de 1997). «Long Short-Term Memory». En: *Neural Computation* 9(8), págs. 1735-1780. DOI: 10.1162/neco.1997.9.8.1735. URL: https://doi.org/10.1162%2Fneco.1997.9.8.1735.

Hornik, Kurt (1991). «Approximation capabilities of multilayer feedforward networks». En: *Neural Networks* 4, págs. 251-257.

Hornik, Kurt, Maxwell Stinchcombe y Halbert White (ene. de 1989). «Multilayer feedforward networks are universal approximators». En: *Neural Networks* 2(5), págs. 359-366. DOI: 10.1016/0893-6080(89)90020-8. URL: https://doi.org/10.1016%2F0893-6080%2889%2990020-8.

Kidger, Patrick y Terry Lyons (2020). *Universal Approximation with Deep Narrow Networks*. arXiv: 1905.08539 [cs.LG].

Kingma, Diederik P. y Jimmy Ba (2017). *Adam: A Method for Stochastic Optimization*. arXiv: 1412.6980 [cs.LG].

Kingma, Diederik P. y Max Welling (2019). «An Introduction to Variational Autoencoders». En: *Foundations and Trends® in Machine Learning* 12(4), págs. 307-392. ISSN: 1935-8245. DOI: 10.1561/2200000056. URL: http://dx.doi.org/10.1561/2200000056.

Kluyver, Thomas et al. (2016). «Jupyter Notebooks – a publishing format for reproducible computational workflows». En: *Positioning and Power in Academic Publishing: Players, Agents and Agendas*. Ed. por F. Loizides y B. Schmidt. IOS Press, págs. 87-90.

Lecun, Y. et al. (1998). «Gradient-based learning applied to document recognition». En: *Proceedings of the IEEE* 86(11), págs. 2278-2324. DOI: 10.1109/5.726791.

Lederer, Johannes (2021). *Activation Functions in Artificial Neural Networks: A Systematic Overview*. arXiv: 2101.09957 [cs.LG].

Leopold, George (jun. de 2016). *Gartner: Role of analytics in security is growing*. URL: https://www.datanami.com/2016/06/15/gartner-role-analytics-security-growing/.

Leshno, Moshe et al. (ene. de 1993). «Multilayer feedforward networks with a nonpolynomial activation function can approximate any function». En: *Neural Networks* 6(6), págs. 861-867. DOI: 10.1016/s0893-6080(05)80131-5. URL: https://doi.org/10.1016%2Fs0893-6080%2805%2980131-5.

Liu, Fei Tony, Kai Ming Ting y Zhi-Hua Zhou (dic. de 2008). «Isolation Forest». En: *2008 Eighth IEEE International Conference on Data Mining*. IEEE. DOI: 10.1109/icdm.2008.17. URL: https://doi.org/10.1109%2Ficdm.2008.17.

Maaten, Laurens van der y Geoffrey Hinton (nov. de 2008). «Viualizing data using t-SNE». En: *Journal of Machine Learning Research* 9, págs. 2579-2605.

Maher, Danny (ago. de 2017). «Can artificial intelligence help in the war on cybercrime?» En: *Computer Fraud & Security* 2017(8), págs. 7-9. DOI: 10.1016/s1361-3723(17)30069-6. URL: https://doi.org/10.1016%2Fs1361-3723%2817%2930069-6.

Martin, Alejandro G. et al. (sep. de 2021). «An approach to detect user behaviour anomalies within identity federations». En: *Computers & Security* 108, pág. 102356. DOI: 10.1016/j.cose.2021.102356. URL: https://doi.org/10.1016%2Fj.cose.2021.102356.

Martín Abadi et al. (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. URL: https://www.tensorflow.org/.

McCulloch, Warren S. y Walter Pitts (dic. de 1943). «A logical calculus of the ideas immanent in nervous activity». En: *The Bulletin of Mathematical Biophysics* 5(4), págs. 115-133. DOI: 10.1007/bf02478259.

Merkel, Dirk (2014). «Docker: lightweight linux containers for consistent development and deployment». En: *Linux journal* 2014(239), pág. 2.

Mhaskar, H. N. y T. Poggio (oct. de 2016). «Deep vs. shallow networks: An approximation theory perspective». En: *Analysis and Applications* 14(06), págs. 829-848. DOI: 10.1142/s0219530516400042. URL: https://doi.org/10.1142%2Fs0219530516400042.

Mhaskar, H. N. y T. Poggio (2019). *Function approximation by deep networks*. arXiv: 1905.12882 [cs.LG].

Morales-Forero, A y Samuel Bassetto (2019). «Case Study: A Semi-Supervised Methodology for Anomaly Detection and Diagnosis». En: *2019 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM)*. IEEE, págs. 1031-1037.

Nadaraya, E. A. (ene. de 1964). «On Estimating Regression». En: *Theory of Probability & Its Applications* 9(1), págs. 141-142. DOI: 10.1137/1109020.

Nwankpa, Chigozie et al. (2018). *Activation Functions: Comparison of trends in Practice and Research for Deep Learning*. arXiv: 1811.03378 [cs.LG].

Paszke, Adam et al. (2019). «PyTorch: An Imperative Style, High-Performance Deep Learning Library». En: *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., págs. 8024-8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

Pedregosa, Fabian et al. (2011). «Scikit-learn: Machine learning in Python». En: *Journal of machine learning research* 12(Oct), págs. 2825-2830.

Plaut, Elad (2018). *From Principal Subspaces to Principal Components with Linear Autoencoders*. arXiv: 1804.10253 [stat.ML].

Pusara, Maja y Carla E. Brodley (2004). «User re-authentication via mouse movements». En: *Proceedings of the 2004 ACM workshop on Visualization and data mining for computer security -*

VizSEC/DMSEC '04. ACM Press. DOI: 10.1145/1029208.1029210. URL: https://doi.org/10.1145%2F1029208.1029210.

Ribeiro, Manassés, André Eugênio Lazzaretti y Heitor Silvério Lopes (abr. de 2018). «A study of deep convolutional auto-encoders for anomaly detection in videos». En: *Pattern Recognition Letters* 105, págs. 13-22. DOI: 10.1016/j.patrec.2017.07.016.

Robbins, Herbert y Sutton Monro (1951). «A Stochastic Approximation Method». En: *The Annals of Mathematical Statistics* 22(3), págs. 400-407. DOI: 10.1214/aoms/1177729586. URL: https://doi.org/10.1214/aoms/1177729586.

Rosenblatt, F. (1958). «The perceptron: A probabilistic model for information storage and organization in the brain.» En: *Psychological Review* 65(6), págs. 386-408. DOI: 10.1037/h0042519.

Rumelhart, D., G. Hinton y James Mcclelland (ene. de 1986). «A General Framework for Parallel Distributed Processing». En: *Parallel Distributed Processing: Explorations in the Microstructure of Cognition* 1.

Rumelhart, David E., Geoffrey E. Hinton y Ronald J. Williams (oct. de 1986). «Learning representations by back-propagating errors». En: *Nature* 323(6088), págs. 533-536. DOI: 10.1038/323533a0.

Sakurada, Mayu y Takehisa Yairi (2014). «Anomaly Detection Using Autoencoders with Nonlinear Dimensionality Reduction». En: *Proceedings of the MLSDA 2014 2nd Workshop on Machine Learning for Sensory Data Analysis*. MLSDA'14. Association for Computing Machinery: Gold Coast, Australia QLD, Australia, págs. 4-11. ISBN: 9781450331593. DOI: 10.1145/2689746.2689747. URL: https://doi.org/10.1145/2689746.2689747.

Shashanka, Madhu, Min-Yi Shen y Jisheng Wang (dic. de 2016). «User and entity behavior analytics for enterprise security». En: *2016 IEEE International Conference on Big Data (Big Data)*. IEEE. DOI: 10.1109/bigdata.2016.7840805. URL: https://doi.org/10.1109%2Fbigdata.2016.7840805.

Slipenchuk, Pavel y Anna Epishkina (2019). «Practical user and entity behavior analytics methods for fraud detection systems in online banking: A survey». En: *Advances in Intelligent Systems and Computing*, págs. 83-93. DOI: 10.1007/978-3-030-25719-4_11.

Szandała, Tomasz (oct. de 2021). «Review and Comparison of Commonly Used Activation Functions for Deep Neural Networks». En: *Studies in Computational Intelligence*. ISSN: 1860-9503. DOI: 10.1007/978-981-15-5495-7. URL: http://dx.doi.org/10.1007/978-981-15-5495-7.

Szegedy, Christian et al. (2014). *Going Deeper with Convolutions*. arXiv: 1409.4842 [cs.CV].

Tieleman, Tijmen, Geoffrey Hinton et al. (2012). *Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude*.

Titterington, D. M., U. E. Makov y A. F. Smith (1995). *Statistical analysis of finite mixture distributions*. Wiley.

Van Rossum, Guido y Fred L. Drake (2009). *Python 3 Reference Manual*. CreateSpace: Scotts Valley, CA. ISBN: 1441412697.

Vitorino, João et al. (2021). *A Comparative Analysis of Machine Learning Techniques for IoT Intrusion Detection*. arXiv: 2111.13149 [cs.CR].

Voris, Jonathan et al. (nov. de 2019). «Active authentication using file system decoys and user behavior modeling: results of a large scale study». En: *Computers & Security* 87, pág. 101412. DOI: 10.1016/j.cose.2018.07.021. URL: https://doi.org/10.1016%2Fj.cose.2018.07.021.

Watson, Geoffrey S. (1964). «Smooth Regression Analysis». En: *Sankhyā: The Indian Journal of Statistics, Series A (1961-2002)* 26(4), págs. 359-372. ISSN: 0581572X. URL: http://www.jstor.org/stable/25049340.

Wattenberg, Martin, Fernanda Viégas y Ian Johnson (2016). «How to Use t-SNE Effectively». En: *Distill*. DOI: 10.23915/distill.00002. URL: http://distill.pub/2016/misread-tsne.

Xiao, Zhisheng, Qing Yan y Yali Amit (2020). *Likelihood Regret: An Out-of-Distribution Detection Score For Variational Auto-encoder*. arXiv: 2003.02977 [cs.LG].

Zhou, Chong y Randy C. Paffenroth (2017). «Anomaly Detection with Robust Deep Autoencoders». En: *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '17. Association for Computing Machinery: Halifax, NS, Canada, págs. 665-674. ISBN: 9781450348874. DOI: 10.1145/3097983.3098052. URL: https://doi.org/10.1145/3097983.3098052.