UNIVERSIDAD CATOLICA SANTA MARIA

FACULTAD DE CIENCIAS E INGENIERIAS FISICAS Y FORMALES

ESCUELA PROFESIONAL DE INGENIERÍA ELECTRÓNICA

DISEÑO E IMPLEMENTACIÓN DE UN CLASIFICADOR DE CIBER-ATAQUES BASADO EN MÁQUINAS PROFUNDAS

Tesis presentada por la Bachiller:

MELANIE BETCY VALCÁRCEL YUCRA

Para optar el título profesional de:

Ingeniero Electrónico.

ASESOR:

Ing. Ronald Coaguila Gómez

AREQUIPA – PERU

2017

Diseño e Implementación de un Clasificador de Ciber-ataques basado en Máquinas Profundas

Presentado por: Melanie Betcy Valcárcel Yucra

Arequipa, Perú.
13 de Marzo del 2017

ÍNDICE GENERAL

Resumen

Abstract

Nomenclatura

BP Back Propagation

CE Cross-Entropy

CNN Convolutional Neural Networks

DARPA Defense Advanced Research Projects Agency

DL Deep Learning

DoS Denial-of-Service

FNR False Negative Rate

FPR False Positive Rate

GPU Graphical Processor Unit

IDS Intrusion Detection System

IP Internet Protocol

MSE Mean Squared Error

NN Neural Network

ReLU Rectifier Linear Unit

ROC Receiver Operating Characteristics

SOM Self-Organizing Map

SVM Support Vector Machine

TCP/IP Transmission Control Protocol/Internet Protocol

TNR True Negative Rate

TPR True Positive Rate

Resumen

El incesante cambio y evolución de las técnicas de ataques a los sistemas informáticos representan un problema mayor en el área de la ciber seguridad. Dada la inmensa cantidad de datos que se debe analizar, es necesario contar con herramientas que detecten si una determinada petición de servicios u operación en la red es un ataque o no. Además, existen restricciones al momento de diseñar estos Sistemas de Detección de Intrusiones (IDS-Intrusion Detection System) pues una tasa de falsa alarma elevada hará que los administradores de la red reciban un número grande de alarmas para su análisis, haciendo lenta la revisión de los mismos y generando una ralentización de los servicios demandados por los usuarios de dicho sistema. Dentro de las técnicas modernas que existen para poder realizar tareas de clasificación se encuentran aquellas basadas en redes neuronales de varias capas o redes profundas entrenadas con técnicas tales como Autoencoders o Dropout. Por ello, este trabajo de Tesis presenta un estudio sobre la aplicabilidad y utilización de algoritmos de aprendizaje profundo en este tipo de entornos para diseñar IDSs. También se estudia modificaciones en la función de coste que permitan contrarestrar el desbalanceo propio de este tipo de entornos (número de ataques es muy inferior al número de datos normales). Se presentarán y discutirán los experimentos realizados utilizando base de datos estándares, en particular basados en la base de datos DARPA1999.

Palabras Clave: Ciber seguridad, detección, redes neuronales, dropout.

Abstract

The incessant change and evolution of techniques of attacks on computer systems represent a major problem in the area of cyber security. Given the immense amount of data to be analyzed, it is necessary to have tools that detect whether a particular service request or operation on the network is an attack or not. In addition, there are restrictions when designing these Intrusion Detection Systems (IDS) because a high false alarm rate will cause the network administrators to receive a large number of alarms to be analyzed, slowing the revision of the same and generating a slowdown of the services demanded by users of the given system.

Among the modern techniques available to perform classification tasks are those based on neural networks of several layers or deep networks trained with techniques such as Autoencoders or Dropout. Therefore, this Thesis presents a study on the applicability and use of deep learning algorithms in this type of environment to design IDSs. We also study changes in the cost function that allow counterbalancing the importance given to each class in the data set (number of attacks is much lower than the number of normal data). Experiments using standard databases, particularly based on the DARPA1999 database, will be presented and discussed.

Keywords: Cyber security, detection, neural networks, dropout.

1 Aspectos Generales

En la actualidad, millones de personas accedemos a internet con el fin de utilizar diversos servicios ofrecidos en línea, con ello podemos acceder a redes sociales, servicios de correo electrónico, acceso a noticias, etc. Además, no solo los servicios son de ocio sino que las empresas también utilizan servicios que permiten a sus trabajadores una amplia movilidad. Así por ejemplo, utilizando el acceso a internet, un ingeniero puede continuar supervisando los procesos a su cargo en tiempo real y sin que la distancia sea un impedimento. Sin embargo, este amplio abanico de servicios –gracias a la alta conectividad– trae consigo eventos que podrían suponer riesgos para dicha empresa ya que personal no autorizado podría descubrir vulnerabilidades y realizar ataques exitosos causando daños posiblemente irreparables. Podemos señalar muchos casos en donde las empresas prestadoras de estos servicios han recibido múltiples ataques. Por ejemplo, un resumen breve de últimos acontecimientos nos permite listar algunos:

- El pasado 21 de octubre del 2016 se produjo uno de los ciberataques más poderosos de los últimos años, dirigido contra la empresa Dyn [1] provocando caídas y un comportamiento lento en servicios como Twitter, Spotify, Amazon; junto con medios de comunicación como New York Times y Financial Times; y plataformas como Imgur, PayPal, Playstation Network, pasando por Netflix y HBO Now.

  El ataque utilizado fue el de Denegación de Servicio (DoS, por sus siglas en inglés *Denial of Service*). La estratégia fue infectar con un código *malware* [2] –difundido online semanas antes– equipos inteligentes conectados a internet: routers, impresoras, televisores, entre otros; conviertiendo a estos equipos infectados en una especie de "ejército robot que lanzan los ataques DoS en simultáneo. Durante aproximadamente 11 horas, los atacantes inundaron con una cantidad enorme de solicitudes que saturaron con datos inservibles a los servidores de Dyn, impidiendo a los usuarios reales acceder a las páginas por la sobrecarga del ancho de banda provocada.

- A finales de setiembre del 2016, Yahoo decide realizar una investigación interna descubriendo que en el 2014 se produjo el

robo de 500 millones de cuentas [3]. Con este ataque fueron afectados los centros de datos en la cual se robaron: nombres de usuarios, direcciones de correo electrónico, fechas de nacimiento, contraseñas, preguntas y respuesta de seguridad que sirven para verificar la identidad de los usuarios. Hoy en día, yahoo asegura que se han tomado medidas para asegurar las cuentas afectadas y que el peligro ya está fuera de los sistemas.

- En Agosto del 2012 un empleado fue víctima de la usurpación de sus credenciales y con ello algunos delincuentes se hicieron de 60 millones de cuentas de la plataforma Dropbox [4]. En un primer momento, Dropbox minimizó este hecho argumentando que únicamente se habían filtrado direcciones de correo electrónico. Sin embargo en la actualidad tenemos certeza de que los delincuentes también obtuvieron las respectivas contraseñas.

- La versión gratuita de Spotify también tubo problemas cuando empezaron a aparecer sitios web poco fiables en su navegador. Los encargados de investigar este hecho aseguran que el intruso que atacó este sistema utilizó una mezcla entre *malware* y publicidad que toma el control de algunas funciones de la computadora [5]. Spotify culpó de todo esto a un solo anuncio y dijo que fue superado, pero que se continuará vigilando y se mantendrá en estado de alerta.

- Según LeakedSource [6], Twitter fue víctima del robo de contraseñas de 32 millones de cuentas, esto se llevó a cabo utilizando un malware dirigido a infectar navegadores como Firefox o Chrome.

  Cabe indicar que LeakedSource es una plataforma que recopila bases de datos que se filtran en la red con nombres de usuarios y contraseñas robadas, haciendo posible que cualquier usuario pueda realizar una busqueda para saber si sus datos están disponibles en la red tras un robo de datos y, de estarlo, de qué páginas han sido robados así como la información disponible (correo, dirección IP, contraseña, nombre, teléfono, etc).

- MySpace confirmó el robo de 360 millones de cuentas [7, 8], que incluyen nombre, contraseña y correo electrónico de usuarios

que crearon sus cuentas antes del 11 de junio del 2013.

- Otro suceso que también podemos enumerar es la confirmación de un robó de 100 millones de cuentas a LinkedIn. Esto se produjo en el 2012 [9]. La compañía afirmó que se tomaron medidas para invalidar contraseñas de las cuentas afectadas.

Todos estos eventos soportan sucintamente la necesidad de continuar con el permanente estudio y desarrollo de sistemas que detecten dichos ataques o intrusiones; además de –y principalmente– continuar investigando nuevos enfoques de diseño de tales herramientas. Es más, la mutabilidad de los ataques –por ende la variabilidad de los mismos–, la cantidad de datos que se deben analizar, la rapidez con que se debe tomar acciones, etc. han abierto la puerta para la aplicación de los algoritmos de *aprendizaje máquina* o *automático* [10, 11] y contar con los beneficios que estos métodos ofrecen, ya sea como pieza fundamental de estos sistemas o como complementos de los mismos.

Una gran cantidad de métodos basan su funcionamiento en medidas eurísticas. Estos –aunque han demostrado cierto grado de eficacia, sin embargo– se ven cada vez más desbordados desde distintos ángulos; siendo un talón de aquiles el número de muestras a analizar para así definir reglas que permitan la detección de ataques [12]. Por otro lado, varias empresas han optado por incluir algoritmos de aprendizaje automático con el objetivo primario de aprender patrones de dichos ataques y la posterior detección de los mismos o variaciones de ellos [13, 14, 15, 16], permitiendo así descubrir relaciones dentro de las características de los datos que no son fáciles de descubrir por una persona humana.

La sofisticación de los ataques naturalmente llevan al estudio de algoritmos más potentes para ayudar en la detección de ciberataques. Un grupo de estos algoritmos son aquellos que residen bajo el paraguas conocido como aprendizaje profundo (DL, del inglés *Deep Learning*) [17, 18, 19]. Estos permiten estimar de mejor manera las relaciones que existen entre los datos y así realizar tareas como la de clasificación. Debido a que nuestro objetivo principal es el estudio de algoritmos que permitan contruir modelos multicapa para aprender información dentro de los datos (que contienen muestras tanto de ataques como tráfico normal) y de esta manera analizar sus posibles ventajas para que puedan ser incluidas dentro

de los sistemas que permiten detectar ataques. Nos centraremos a continuación en listar trabajos previos en esta área de investigación y las propuestas previas, resaltando las desventajas que estas muestran y que pretendemos cubrir con este trabajo.

1.1 Estado del arte

El primer IDS fue propuesto en 1980 [20], y a partir de ello y hasta fines de los 80's, estos sistemas utilizaron métodos meramente eurísticos basados en reglas de comportamiento de los datos. Los primeros IDSs que integraron máquinas de tipo Red Neuronal [21] (NN, del inglés *Neural Nets* o *Neural Networks*) son [22, 23, 24], en donde los autores entrenaron redes neuronales para que estas puedan aprender la distribución de los datos normales o comportamiento normal del usuario, y clasificar como ataque todo evento que no se encuentre dentro de los límites de esta distribución (detección de anomalías). En [25] se presenta una modificación a las propuestas anteriores, incorporando frecuencias de palabras claves como características. Además, una segunda NN es entrenada para distinguir entre los distintos tipos de ataques.

En [26] se presenta un IDS más complejo y que tiene una primera etapa de agrupamiento de los datos utilizando una red SOM (del inglés *Self-Organizing Map*) y que son un tipo especial de red neuronal que se entrena de manera no supervisada. La salida de esta red es un vector de datos binarios –cumple la función de un extractor de características– y que se utiliza para entrenar una red neuronal discriminativa. También podemos indicar que esta propuesta usa los números de los puertos TCP/IP y usaron la base de datos DARPA 1999 [27, 28] y que a diferencia de propuestas anteriores –como en [24]– el sistema usa varias muestras dentro de una ventana temporal, mostrando por ello mayor efectividad para detectar comportamientos sospechosos que se desarrollan en periodos de tiempo más largos.

Un área importante de estudio son los que caen dentro del campo de las reglas de asociación y sistemas *Fuzzy*. Pero estando este trabajo centrado en máquinas de tipo Red Neuronal, dejamos que el lector pueda revisar los siguientes artículos para una mayor profundización de estos algoritmos en entornos de ciber seguridad [29, 30].

Otro grupo importante son las llamadas redes Bayesianas [31].

Estas redes son modelos probabilísticos [32] que asumen ciertas interacciones existentes entre las características de los datos. Es importante indicar que como en todos los modelos Bayesianos, el diseñador debe indicar todas las interacciones que existen en los datos [33]. En [34], se presenta un diseño de un IDS basado en este tipo de red utilizando un único nodo o variable oculta que indica si la muestra es o no ataque.

Los árboles de decisión definen otro campo de estudio. Básicamente, estos métodos utilizan una a una cada característica y van diseñando reglas de decisión (nodos) para etiquetar los datos [35, 36]. Ejemplos de estos métodos son [13, 37, 38, 39]. Sin profundizar, podemos indicar que estos métodos, aunque han demostrado efectividad en muchos campos, tienen el inconveniente de que aprenden sobre el espacio de entrada y no aprenden relaciones complejas que existen entre las características de los datos.

Además, dada la gran popularidad que han tenido en años anteriores, las máquinas de vectores soporte (SVM, del inglés *Support Vector Machines*) [40] han recibido mucha atención por su gran performance en comparación a las redes neuronales [41, 42, 43, 44]. Estas sin embargo tienen la desventaja de que el entrenamiento escala de forma $O(n^2)$ en memoria y (On^3) en tiempo de ejecución, siendo n el número de muestras. Y aunque hay versiones para grandes números de datos [45], estos presentan ciertas limitaciones para aprender no linealidades ya que no utilizan *kernels*.

Un grupo más robusto de métodos son los que entrenan conjuntos de máquinas, usualmente máquinas inestables, explotando el hecho de que estas máquinas de forma individual aprenden distintos ángulos de los datos [46, 47, 48, 49, 50, 51]. La principal ventaja es que las tasas de clasificación pueden mejorar pero tiene un inconveniente muy grande: tiempo de entrenamiento y tiempo de operación elevados.

1.2 Descripción del problema

Los ataques de ciberseguridad están evolucionando constantemente además de incrementarse. Los atacantes van desarrollando nuevas técnicas para poder simular tráfico normal y perpetrar dichos ataques. Por ello, es importante mejorar los IDSs existentes para cubrir estos nuevos escenarios. Siempre manteniendo la tasa

de detección alta y el número de falsas alarmas lo más bajo posible[1].

Por otro lado, el resurgimiento de las estructuras tipo NN desde [52] y [53], han permitido avances impresionantes en muchos campos, desde visión artificial, medicina, etc. Su amplia capacidad de aprender jerarquías de caraterísticas utilizando técnicas de preentrenamiento como Autoencoders [53], atenuación de problemas de sobre ajuste con números de datos grandes y regularizaciones tipo Dropout [54], hacen que sea un candidato fuerte para ser aplicado en el ámbito de la ciberseguridad.

Podemos señalar que dentro de sus ventajas están que son más profundas que las SVMs (que a pesar de usar kernels su estructura es lineal) y escalan linealmente con el número de muestras. Aprenden funciones altamente no lineales a diferencia de los árboles. También, debido a su gran performance es posible reemplazar un conjunto de máquinas por una sóla máquina profunda pues es posible alcanzar resultados iguales y normalmente mejores.

Por todo lo anterior, se plantea el estudio de algoritmos de DL para la detección de ataques en entornos propios de la ciberseguridad.

1.3 Hipótesis

El uso de aprendices profundos, gracias a la alta capacidad que estas poseen para modelar funciones altamente no lineales, mejora el rendimiento en términos de tasa de pérdida y falsa alarma cuando son utilizadas para la detección de ataques en entornos de seguridad informática.

1.4 Objetivos de la Tesis

Este trabajo tiene como objetivo fundamental el diseñar, implementar y optimizar un clasificador para detectar intrusiones en redes informáticas utilizando algoritmos de deep learning y así obtener máquinas altamente no lineales que puedan desarrollar dicha tarea, y así ser parte o elemento único de un IDS. Con el fin de alcanzar esta meta, el presente trabajo traza un camino para el entrenamiento de tales máquinas con datos propios de redes informáticas.

[1] En general, el número de falsas alarmas depende de la cantidad de personas que conforman el grupo técnico que decide finalmente si la muestra es o no un ataque, como también el número máximo de muestras que cada persona puede analizar.

Se define dos grupos de objetivos cuyos alcances son de gran importancia tanto para la realización y culminación del presente trabajo como para la complementación del mismo. Estos son:

1.4.1 Objetivo principal

- Diseñar e implementar un IDS utilizando clasificadores de máquinas profundas.

1.4.2 Objetivos secundarios

- Estudiar técnicas avanzadas de aprendizaje máquina y su desenvolvimiento en escenarios de ciberseguridad.
- Definir un conjunto de bases de datos que sean representativos del problema.
- Seleccionar un método adecuado para codificar características de los datos propios de estos entornos.

1.5 Factores Involucrados en la Tesis

Durante el planteamiento de este trabajo, se han definido los factores que inciden a su desarrollo de forma directa (factores dependientes) o indirectamente (factores independientes). Así procedemos a detallarlos a continuación.

1. **Factores dependientes**:
    - Bases de datos: Este factor es de suma importancia ya que permite el correcto desarrollo de este trabajo. Permitirá a los algoritmos de Aprendizaje Máquina descubrir y aprender la información necesaria para detectar posibles ataques.
    - Selección de algoritmos: Internet permite acceder a una vasta cantidad de información, entre ellos, acceso a un conjunto de artículos de investigación, revistas especilizadas y libros. Estas bases de información permiten estudiar y seleccionar algoritmos del estado del arte para desarrollar la presente Tesis.

- Tecnologías escogidas: existen en la actualidad diversas plataformas (Matlab [55], Python [56], R [57], etc.). Se selecciona Python como la plataforma a utilizar en esta Tesis ya que es de libre distribución; y sobre todo, porque se está convirtiendo en el estandar *de facto* en el área del aprendizaje máquina y muchos otros campos de la ingeniería.
- Herramientas de trabajo: básicamente, es necesario el uso de un computador para realizar este trabajo. Sin embargo, en la actualidad existen centros de cómputo de alto rendimiento en el que es posible realizar experimentos en paralelo (sobre todo para la búsqueda de hiper-parámetros).

2. **Factores independientes**:

- Factores legales: estos, aunque inciden de forma directa en el trabajo por el tipo de bases de datos que se consideran, no son manejables pues dependen de las legislaciones de cada país. Un análisis sobre ello permite resaltar que existe una barrera legal muy fuerte al momento de contar con datos de usuarios para su análisis y uso en el desarrollo de nuevos algorítmos para prevenir y detectar ataques informáticos. Así, las bases de datos seleccionadas son el resultado de extensas simulaciones de redes de computación; y, por ello, podemos afirmar que cuentan con muestras anonimizadas.
- Tipos de ataque a detectar: en relación a los tipos de ataques a dectectar, estos vienen definidos por la base de datos a utilizar. Esta restricción no invalida el presente estudio pues –como claramente detalla el apartado de objetivos– este trabajo busca analizar la capacidad de las redes profundas de extraer información de dichas bases de datos para realizar la clasificación entre ataque y tráfico normal.
- Factores económicos: debido a la importancia que la seguridad informática tiene, en la actualidad existe una inversión de recursos económicos cada vez más creciente para el desarrollo de nuevas tecnologías de prevención y detección de ataques a redes informáticas. Por ello rescatamos el hecho de que los algoritmos de Aprendizaje Máquina

son capaces de abaratar dichos costos, ya sea por el hecho del número de personas que se dedica a extraer reglas eurísticas o porque estos algoritmos pueden extraer información relevante de una base de datos grande y en menos tiempo.

- Red informática: dada la inmensa variedad de tipos, arquitecturas y componentes de redes de computadores (sistemas operativos, servicios, etc.), el diseño de algoritmos para detectar todos los tipos de ataques para distintos tipos de redes escapa del ámbito de este trabajo. Por este motivo, las bases de datos que aquí se listan delimitan el área de aplicación. Sin embargo, debemos indicar que este trabajo es de investigación y los pasos tomados para su desarrollo son los que están definidos por el método científico. Gracias a lo anterior, los resultados y conclusiones son relevantes para el área en cuestión.

1.6 Metodología

Siguiendo la Hipótesis planteada líneas arriba, el objetivo es investigar si las máquinas profundas tienen la capacidad de modelar correctamente la función que relaciona los datos disponibles con sus etiquetas (e.g. ataque o tráfico normal). Para ello, se debe indicar que se siguen mecanismos estrictos de validación de resultados haciendo que los modelos obtenidos sean reproducibles. Debido a esto, el método científico define la filosofía sobre el que se desarrolla este trabajo de Tesis.

Debemos indicar que se siguieron las siguientes pautas:

1. Experimentación
2. Discusión de resultados
3. Validación de resultados
4. Comprobación de la Hipótesis

1.7 Relevancia social y actores involucrados

Gracias a este trabajo, avizoramos los siguientes beneficiarios:

- Empresas: podrían incrementar beneficios al estar estas menos susceptibles a ataques de ciertos tipos cuyos patrones fueron aprendidos
- Usuarios: aumenta las capacidades de herramientas ya existentes para estos fines, o las complementa.
- Universidades: cumplen su rol como generador de conocimiento y nuevas tecnologías.
- Desarrolladores de software: nuevas herramientas a tomar en cuenta.

1.8 Implicaciones prácticas

- Una primera aproximación a las implicaciones que conlleva este trabajo de Tesis es la ampliación de la teoría relacionada a las potenciales aplicaciones de tecnologías de tipo red neuronal, especialmente a aquellas que permiten la aproximación de redes profundas.
- Segundo, refuerza la idea de que el Aprendizaje Máquina es cada vez más un área de estudios herramienta transversal a todos los campos. En este caso, al campo de la ciberseguridad.
- Tercero, extiende las herramientas del equipo técnico encargado de la decisión final para determinar la presencia o no de un ataque, cumpliendo un rol importante en la toma de decisiones sobre el tráfico que ingresa a una determinada red.

1.9 Delimitación

En la misma línea de lo señalado anteriormente, los alcances de este trabajo de Tesis estará delimitado por la información que podamos extraer de las bases de datos a utilizar.

Sin embargo, no existe impedimento para que los modelos resultantes del presente trabajo sean utilizados en otras estructuras. Un potencial usuario debería recopilar datos propios de su red y modificar los modelos que aquí se presentan para ajustarlas a su propia arquitectura.

1.10 Estructura del documento

El presente trabajo de Tesis se estructura de la siguiente manera:

- En la Sección 2 se presenta el marco teórico sobre ciberseguridad en el que se suscribe el trabajo.
- La Sección 3 presenta en detalle las bases de datos a utilizar así como sus características.
- En la Sección 4 se presenta la teoría sobre el Aprendizaje Máquina, así como la descripción de los algoritmos que permiten el entrenamiento de máquinas profundas.
- En la Sección 5 se describen los métodos utilizados para acondicionar las características de tal modo que sean útiles para nuestros propósitos, se definen los modelos de máquinas a entrenar y se detallan los experimentos realizados. Además, esta sección discutirá los resultados obtenidos así como su análisis. Presenta las ventajas y desventajas, basado en los resultados obtenidos, que estos métodos ofrecen.
- Finalmente, la Sección 6 presenta las conclusiones, así como líneas futuras de investigación a seguir.

2 Seguridad Informática

Cuando hablamos de Seguridad Informática nos referimos a la capacidad del sistema para proteger la información de riesgos que puedan afectarla. En este trabajo de Tesis se define la seguridad informática como un conjunto de normas, procedimientos y herramientas, que tienen como objetivo garantizar la disponibilidad, integridad, confidencialidad y buen uso de la información que existe en un determinado sistema.

El motivo fundamental para implementar la seguridad en los sistemas es la protección de los datos de usuarios y así evitar el hurto de información, suplantación y caída de los sistemas y servicios. Básicamente, podemos indicar que la red informática se tiene que cuidar de dos causas, una de ellas activa y la otra pasiva.

- Activas: El usuario no conoce las funciones del sistema y muchas aplicaciones pueden ser perjudiciales, por ejemplo no desactivar los servicios de red que el usuario no necesita, permitiendo que otro usuario o atacante puede explotar las vulnerabilidades.
- Pasivos: El no conocer las medidas de seguridad disponibles.

2.1 Medidas de seguridad

Las medidas de seguridad que todo sistema debe tener para proteger sus redes se dividen en tres partes: Prevención/Protección, Detección y Respuesta.

- Prevención/Protección: Consiste en utilizar mecanismos que protegen a las redes informáticas frente a ataques que puedan producirse; luego, se definen las reglas eurísticas a adoptar utilizando *firewalls*.
- Detección de ataques: Se desarrollan actividades para medir el nivel de riesgo analizando y detectando cuándo las medidas de protección no han sido efectivas, por ejemplo podemos utilizar IDSs.
- Reacción: Cuando se detecta un ataque se efectúan actividades de respuesta, de manera automática o de forma manual. Son las actividades re-activas, por ejemplo utilizar los Sistemas de Protección de Intrusos los IPS, como un sistema de respuesta automatizada.

A continuación se procede a desarrollar cada uno de los sistemas de detección que debemos tener presente para evitar cualquier tipo de ataque:

- Firewall
- Sistema de Detección de Intrusos
- Sistema de Prevención de Intrusos
- Filtro de Contenidos

2.1.1 Firewall

Los Firewall son sistemas que protegen a una red de las intrusiones provenientes del exterior encargándose de filtrar el tráfico de red. El firewall es un sistema de control de comunicaciones dentro de una red, mantiene monitorizada la red y brinda datos clave al administrador del sistema tales como: tipo de tráfico que pasa a través de él, volumen de datos, número de intentos de conexión desde el exterior, etc.

Firewall se pueden clasificar según el ámbito de uso en:

- **Firewall Personales:** Es una aplicación software que filtra el tráfico que entra o sale de un ordenador.
- **Firewall de Red:** Es un equipo instalado entre dos o más redes (hace puente entre las redes). El firewall puede ejecutarse en un dispositivo de red especializado o sobre un ordenador. En referencia a la capa TCP/IP donde el tráfico puede ser interceptado, existen dos categorías de firewall:
  - **Firewall a nivel de red:** El control de tráfico a nivel de red consiste en analizar todos los paquetes que llegan a un interfaz de red, y decidir si se deja pasar o no en base a la información de sus cabeceras: protocolo, dirección de origen y dirección de destino.
  - **Firewall a nivel de aplicación:** El control se hace analizando las comunicaciones a nivel de su contenido. El firewall es por tanto mucho más inteligente y posee un control más fino de la comunicación, suponiendo una mayor carga de trabajo. Por ejemplo, si se filtra el protocolo SMTP,

se puede configurar para que impida todos los correos destinados a servidores de correos públicos como hotmail o gmail. No puede analizar comunicaciones cifradas, como las comunicaciones HTTPS.

2.1.2 Sistema de Detección de Intrusos

Un Sistema de Detección de Intrusiones (IDS), es un sistema hardware o software encargado de detectar la intrusión o intento de intrusión en un sistema informático o red de comunicaciones.

Lo que buscan son señales de rastreos a puertos abiertos o paquetes malformados. Cuando se detecta una intrusión, el IDS envía una alerta a los administradores de la red y recoge la información más relevante sobre la intrusión. El funcionamiento de los IDS se basa en el análisis del tráfico de red. Los IDS se componen de sensores virtuales, como *sniffers* que capturan tráfico que luego es analizado y de este modo comprobar el tipo de tráfico, su contenido y su comportamiento. Para detectar las intrusiones suelen disponer de una base de datos o muestras debidamente etiquetadas de ataques conocidos. Dichas firmas permiten al IDS distinguir entre el tráfico normal de la red y el tráfico que puede ser resultado de un ataque o intento del mismo. Existen tres tipos de sistemas de detección de intrusos:

- **HIDS** (HostIDS): Un HIDS analiza diferentes áreas para detectar y prevenir actividades maliciosas o intrusiones en la red. Analiza diferentes tipos de registros de kernels, sistemas, servidores, cortafuegos, y los compara contra una base de datos interna sobre ataques conocidos.

- **NIDS** (NetworkIDS): Es un IDS conectado a una red o segmento de red. Su interfaz debe funcionar en modo multitarea capturando así todo el tráfico de la red. Puede estar instalado en uno de los equipos del segmento de red o en un equipo de la red como un hub o switch.

- **DIDS** (DistributedIDS): Sistema basado en la arquitectura cliente-servidor compuesto por una serie de NIDS que actúan como sensores que envían la información de posibles ataques en una unidad central que controla toda la red de forma centralizada. La ventaja es que en cada NIDS se puede fijar reglas

de control especializadas para determinados segmentos de red. Pueden vigilar toda una red LAN compleja de forma centralizada. Según su reacción ante la detección de una intrusión los IDSs se clasifican en dos tipos:

- **Reactivos:** En un sistema reactivo, el IDS responde a la actividad sospechosa reprogramando el firewall para que bloquee tráfico que proviene del atacante. Actualmente, los fabricantes suelen denominar a los IDSs reactivos como IPSs.
- **Pasivos:** Los IDSs pasivos, al detectar una intrusión, guarda la información y manda una señal de alerta. Según el tipo de amenaza, se informa a los administradores de la red.

El IDS no puede detener los ataques por sí solo, excepto los que trabajan junto con un firewall. Los IDSs guardan información de las intrusiones detectadas y copia de los paquetes afectados. Esto permite a los administradores de la red realizar un análisis forense del ataque. La implementación de un IDS puede ser hardware, software o una combinación de ambas. Los IDS hardware son equipos especializados conectados a la red, capaces de controlar una red con mucho tráfico. Los IDS software son aplicaciones ejecutadas en un equipo de la red. Es una solución barata, pero tiene el inconveniente de que si tienen que analizar mucho tráfico, pueden sobrecargar el equipo donde se ejecutan.

Un IDS conocido es la aplicación Snort (Software GPL) y consiste en un NIDS e incluye un sniffer para la captura de los paquetes de las comunicaciones, además de ser capaz de analizar los paquetes **buscando patrones de intrusiones.**

2.1.3 Sistema de Prevención de Intrusos IPS

Los sistemas de prevención de intrusos IPS son una evolución de los IDSs, tienen un desempeño mas proactivo. Los IPSs, además de detectar y prevenir analizan el comportamiento de las comunicaciones ajustando las **reglas eurísticas** y filtros de los sistemas de seguridad; es decir son capaces de detectar las acciones de toma de información y reaccionar para prevenir el ataque. Trabajan a nivel multicapa de red y aplicación, realizan un análisis detallado de

las comunicaciones, identifican el tipo de servicio al que acceden las comunicaciones y supervisan que estas sigan sus normas.

Un IPS pueden combinar las funciones de seguridad preventiva, IDS, antivirus y filtrado de contenido. Dentro de sus funciones podemos listar los siguientes:

- Ser sistemas pro-activos, detectar ataques antes de que se produzcan.
- Actuar cuando se detecte una intrusión de dos formas:
  - Enviando alertas a los administradores de la red.
  - Modificar las reglas para detectar ataques.

  Uno de los principales inconvenientes de los IPSs es que confunden el tráfico inofensivo con el tráfico de riesgo, por este motivo **es importante reducir las tasas de falsa alarma manteniendo la capacidad de detección alta.** Como en el caso de los IDSs, podemos encontrar dos tipos de IPS según su localización en la red:

  - Host IPS: IPS software localizado en los servidores, proveen una protección más específica y son capaces de analizar las comunicaciones cifradas.
  - Network IPS e IPS distribuidos: Equipos hardware con funciones de IPS. Analizan el tráfico que circula por la red. Tiene problemas para analizar las comunicaciones cifradas. Al igual que los IDS también hay IPSs distribuidos que permiten una administración centralizada.

2.1.4 Filtro de contenidos

Este software de control de contenidos, es el más especializado en monitorizar y filtrar la información transmitida a través de la red, utilizando el servicio web. Este filtro determina qué contenido estará disponible para una determinada máquina o red. La finalidad es prevenir la visualización de contenidos que el propietario del ordenador considera prohibidos. Puede estar localizado en:

- Ordenador Personal: Filtra los contenidos que se visualizan en ese ordenador.

- En el servidor de Internet (router o proxy): Filtra los contenidos para toda la red. La capacidad de personalizar los filtros es menor, pero en cambio su mantenimiento es más sencillo. No puede analizar los contenidos encriptados.

2.2 Tipos de ataques

Existe una inmensa cantidad de tipos de ataques que mutan con mucha velocidad. Sin embargo, en este documento analizaremos sólo cuatro de ellos pues son los que existen en las bases de datos con los que se desarrollaron los experimentos. Estos son:

- *Denial of Service* (DoS)
- *Probe*
- *Remote to local* (R2L)
- *User to Remote* (U2R)

A continuación se explicará brevemente de qué trata cada uno de ellos:

2.2.1 DoS

El ataque de denegación de servicio es un tipo de ataque que consiste en que un servidor recibe de golpe muchas peticiones que acaban saturando el computador o el servidor, afectando así a los servicios que deberían prestarse. Por ejemplo, esto puede originarse cuando miles de usuarios empiezan a enviar solicitudes de ingreso a una cuenta de correo con información basura (con contraseñas incorrectas), provocando en muchos casos la caída del sistema de correos electrónicos.

2.2.2 Probe

Este tipo de ataque busca encontrar vulnerabilidades escaneando computadoras o redes en busca de direcciones IP válidas, puertos activos, entre otros.

2.2.3 R2L

En este tipo de ataque, el atacante busca tener acceso a un servicio o computador filtrando archivos y modificando datos que existen en ellos.

2.2.4 U2R

Los ataques U2R son aquellos en los que un usuario local intenta obtener información que le permita tener mayores privilegios o incluso privilegios de administrador.

3 Bases de Datos en Ciberseguridad

3.1 DARPA 1999

Las bases de datos DARPA 1999 fueron simulados en los laboratorios Lincoln del MIT [58] han aportado en las investigaciones de detección de ataques y son usadas para diseñar IDS y hacer pruebas para detectar intrusiones. Estas bases de datos fueron almacenadas en formato *tcpdump* [2]

Con el análisis de los datos se busca detectar las anomalías presentes en esta base de datos. El conjunto de datos DARPA 1999 contiene 190 casos de 57 ataques que incluyen 37 Probes, 63 ataques DoS, 53 ataques R2L, 37 U2R. Los tipos de ataques y su agrupación en clases de ataques se muestran en la Tabla 1

Clase de Ataques	Tipo de ataques
Probe	portsweep, ipsweep, cheese, satan, msscan, ntinfoscan, lsdomain, illegal-sniffer
DoS	apache2, smurf, neptune, dosnuke, land, pod, back, teardrop, tcpreset, syslogd, crashiis, arppoison, mailbomb, selfping, processtable, udpstorm, warezclient
R2L	dict, netcat, sendmail, imap, ncftp, xlock, xsnoop, sshtrojan, framespoof, ppmacro, guest, netbus, snmpget, ftpwrite, httptunnel, phf, named
U2R	sechole, xterm, eject, ps, nukepw, secret, perl, yaga, fdformat, ffbconfig, casesen, ntfsdos, ppmacro, loadmodule, sqlattack

Table 1: Ataques presentes en el conjunto de datos DARPA 1999.

3.1.1 Críticas a la base de datos DARPA 1999

La principal crítica contra el conjunto de datos de evaluación DARPA 1999 es que el software de generación de tráfico de la plataforma de prueba no está disponible públicamente y, por lo tanto, no es posible determinar la exactitud del tráfico de fondo insertado en la evaluación.

Los criterios de evaluación no tienen en cuenta los recursos del sistema utilizados, la facilidad de uso, o qué tipo de sistema es.

[2] *tcpdump* es una manera de capturar los datos y los guarda de la siguiente manera: tiempo de llegada del paquete, nombre del protocolo y direcciones de origen y destino

McHugh [27] presenta sus críticas sobre los procedimientos utilizados en la construcción del conjunto de datos y en la realización de la evaluación.

Los conjuntos de datos de entrenamiento y pruebas no están correlacionados para los ataques R2L y U2R y por lo tanto la mayoría de los algoritmos de reconocimiento de patrones –excepto los detectores de anomalías que aprenden solo de los datos normales– se desempeñarán muy mal en la detección de los ataques R2L y U2R.

El tráfico normal en las redes informáticas reales y en el conjunto de datos no están correlacionados; causando que los algoritmos generen una gran cantidad de falsas alarmas cuando son probadas con datos reales.

Los datos de los ataques DoS y R2L tienen una varianza muy baja haciendo que sea difícil de detectar nuevos ataques de este tipo en un conjunto de prueba.

3.2 KDD-DARPA

La base de datos DARPA 1999 se utilizó como base para desarrollar la competición 1999 KDD cup [59]. Y, a pesar de que arrastra algunos problemas de la base de datos original DARPA 1998, aún es utilizada para probar los métodos de detección de muestras.

En [60], McHugh critica algunos aspectos de esta base de datos recogidos de tráfico, taxonomía de ataque y criterios de evaluación. Con respecto a los datos de tráfico recogidos, critica la falta de evidencia estadística de la similitud con el tráfico típico de la red, las bajas tasas de tráfico y la topología de red plana. En particular algunos ataques son fácilmente detectados debido a los errores de simulación, ejemplo de esto es que muchos paquetes IP son detectados como ataques pues tienen el valor TTL de la trama IP en valores inferiores a los del tráfico simulado como normal, posiblemente este efecto se debe a que los ataques y el tráfico normal fueron sintetizados en diferentes computadoras.

Ataques a servidores DNS, servidores web y de correo electrónico, están correlados con direcciones no vistas, sin embargo esta debería ser la norma en este tipo de servicios.

Varios ataques son detectados por el tamaño incorrecto de ventana TCP.

En general las investigaciones sugieren que muchos atributos del

tráfico de red que parecen ser discriminadores en esta base de datos no lo son en el tráfico real. Como resultado, reglas de detección de ataques, anomalías, entre otros, basadas en uno de estos atributos generarán más falsas alarmas en el tráfico real.

En qué medida lo anterior conduce a una menor cantidad de detección de ataques dependerá de si estos atributos permiten discriminar los ataques del tráfico normal. Podemos describir los siguientes ejemplos:

3.2.1 Detecciones enmascaradas en tráfico real

Los ataques dotnuke, neptuno, netbus, netcat, nt infoscan y cheese tienen valores TTL de 253 o 126 en el conjunto de prueba o *test*, pero estos valores nunca aparecen en el conjunto de entrenamiento. En el tráfico real, es probable que estos y muchos otros valores aparezcan.

3.2.2 Detección de falsas alarmas

Los ataques a servicios públicos como HTTP (apache2, back, crashiis, phf), SMTP (mailbomb, sendmail) o DNS no deberían ser detectables por anomalías de direcciones de origen en tráfico real. Sin embargo, todos estos ataques tienen direcciones IP de origen que nunca aparecen en los datos de entrenamiento. Si bien esto puede ser cierto en ataques reales, este hecho es poco útil.

3.2.3 Ataques simulados incorrectamente

El ataque DoS llamado smurf puede ser detectado por errores de suma de comprobación ICMP en esta base de datos. Se trata de una inundación de paquetes ECHO REQUEST, con una dirección fuente falsificada o copiada de alguna víctima. En una red real, estos ataques se originan en *hosts* neutrales que no generan errores de suma de comprobación. Se debe indicar que este ataque es uno completamente simulado y su código no fue publicado.

3.3 NSL-KDD

Una de las deficiencias más importantes en el conjunto de datos KDD es el gran número de registros redundantes, lo que hace que los

algoritmos de aprendizaje estén sesgados hacia estos registros frecuentes, y así evitar que aprendan registros poco frecuentes que son usualmente más perjudiciales como los ataques U2R y R2L. Además, la existencia de estos registros repetidos en el conjunto de pruebas hará que los resultados de evaluación sean sesgados y métodos que clasifiquen bien estos registros tendrán tasas de detección más altas. El conjunto de datos NSL-KDD tiene las siguientes ventajas sobre el conjunto de datos KDD original:

- No incluye registros redundantes en el conjunto de entrenamiento, por lo que los clasificadores no estarán sesgados hacia registros más frecuentes.
- No hay registros duplicados en los conjuntos de prueba propuestos; por lo tanto, el rendimiento del aprendizaje no está sesgado por los métodos que tienen mejores tasas de detección en los registros frecuentes.
- El número de registros seleccionados de grupos con diferente dificultad es inversamente proporcional al porcentaje de registros en el conjunto de datos KDD original. Como resultado, las tasas de clasificación de distintos métodos varían en un rango más amplio facilitando la evaluación de diferentes técnicas de detección de ataques.
- El número de registros de entrenamiento y prueba son razonables, lo que hace que sea más asequible ejecutar los experimentos sin necesidad de seleccionar al azar una pequeña porción. En consecuencia, los resultados de las evaluaciones de los diferentes trabajos de investigación serán comparables.

En general, las bases de datos en este campo de estudios siempre tendrán críticas en relación a la diferencia existente entre datos simulados y reales, o que los ataques cambian o aparecen nuevos tipos, entre muchos otros aspectos discutibles. A pesar de esta crítica fundamentada, dichas bases de datos se utilizan en todas las investigaciones que proponen nuevos sistemas de detección de intrusos pues las regulaciones de confidencialidad de datos junto con la reticencia de empresas e instituciones para divulgar datos reales imposibilitan simulaciones y pruebas sobre tráfico real. Sin embargo, es válido indicar que el camino para adecuar estos sistemas entrenados con

estas bases de datos es directo si se respetan los pasos del método científico al realizar las experimentaciones.

4 Aprendizaje Máquina

Aprendizaje Máquina es el área de estudio que pretende desarrollar algoritmos para que las computadoras realicen tareas por medio de ejemplos anteriores. Se ha dividido principalmente en dos áreas denominadas aprendizaje supervisado y no supervisado (Fig. 1). El objetivo principal en el aprendizaje no supervisado es encontrar relaciones entre datos sin contar con información alguna de clase, ejemplo de este tipo de aprendizaje es el agrupamiento automático o *clustering*; en donde se busca dividir los datos en grupos que compartan características símiles.

El aprendizaje supervisado, por el contrario, consiste en encontrar relaciones entre datos ($\mathbf{x} \in \mathcal{X}$) y etiquetas ($\mathbf{y} \in \mathcal{Y}$). Más exactamente, consiste en estimar una función que relacione las muestras con sus respectivas etiquetas (Ec. 1).

$$y = f(\mathbf{x}) \tag{1}$$

Principalmente, dos problemas destacan y son los problemas de regresión y de clasificación. En el primer caso la tarea es estimar las etiquetas $y \in \mathbb{R}$ de cada dato $\mathbf{x} \in \mathbb{R}$; en el segundo caso, las etiquetas tienen un número finito de posibles valores llamadas clases y el objetivo es asignar cada muestra o ejemplo $\mathbf{x}$ a una clase determinada $\hat{y}$.

Si el número de clases es igual a dos, entonces decimos que se trata de una clasificación binaria ($y \in \pm 1$); en caso contrario cuando $K > 2$ se llama clasificación multiclase ($y \in 1, 2, 3, ..., K$). También, en el caso en que el número de muestras en cada clase es aproximadamente igual se dice que el problema es balanceado y si por el contrario el número de muestras perteneciente a cada clase es muy distinto entonces decimos que se trata de un problema desbalanceado. Existe en la literatura diversos métodos para diseñar y optimizar o entrenar máquinas. Además, es necesario indicar que la comparación de estos métodos se realiza a través de medidas que se denominan funciones de coste.

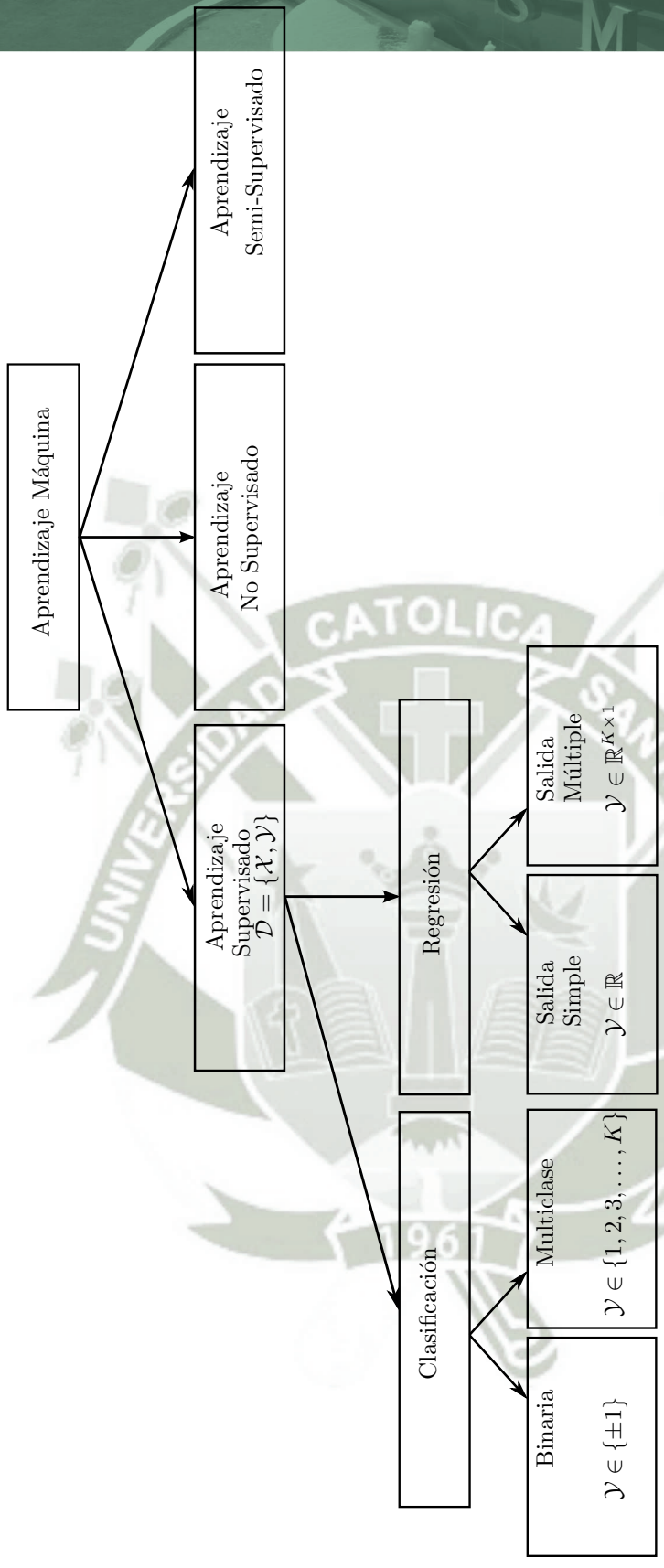

Figure 1: Ramas de estudio más destacables del Aprendizaje Máquina. Se destaca el área del aprendizaje supervisado y en especial los problemas de clasificación Binaria y Multiclase.

4.1 Funciones de coste

Las funciones de coste miden las discrepancias entre las etiquetas verdaderas y aquellas estimadas $\hat{y}$ por una determinada máquina. Tienen la propiedad de ser convexas y poseer un mínimo global. Dentro de las funciones de coste más utilizadas en el área del aprendizaje máquina se encuentran la de Error Cuadrático Medio (MSE, de inglés *Mean Squared Error*) y la de Cross-Entropía (CE).

4.1.1 Mean Squared Error - MSE

Esta función de coste asume que el error sigue una distribución gausiana. Se define como:

$$MSE = \frac{1}{N}\sum_{n=1}^{N}((y^n - \hat{y}^n)^2) \tag{2}$$

En forma vectorial

$$MSE = \mathbb{E}([\bar{\mathbf{y}} - \hat{\mathbf{y}}]^{\mathbf{T}}[\bar{\mathbf{y}} - \hat{\mathbf{y}}]) \tag{3}$$

donde $\bar{\mathbf{y}}$ y $\hat{\mathbf{y}} \in \mathbb{R}^{\mathbf{N}\times\mathbf{1}}$ son los vectores que contienen las etiquetas de las N muestras y sus valores estimados, respectivamente.

4.1.2 Cross Entropía - CE

Esta función de coste es más útil en tareas de clasifcación y se inspira en la teoría de información mide el valor de entropía entre dos distribuciones p y q. La entropía es la esperanza matemática de la información q o $log(1/q)$ a la vista de p. La ecuación 4 muestra la entropía entre p y q cuando estas son discretas. Regresando a nuestro caso de clasificación, para el caso binario, si $y \in [0, 1]$ e $\hat{y} \in [0, 1]$, entonces decimos que las etiquetas siguen una distribución binomial. Así, la entropía es

$$H = \sum_i P_i log(-\frac{1}{q_i}) \tag{4}$$

Esta función de coste es más útil en tareas de clasificación, para utilizarla es necesario que las salidas se limiten entre valores 0 y 1. En el caso la etiqueta k de la n-ésima muestra sea igual a 1, i.e. $y_k = 1$, entonces la cross entropía se reduce a

$$h_1 = -\sum_{k=1}^{K}\left[y_k^{(n)} log\left(\hat{y}_k^{(n)}\right)\right]; \quad \forall y_k = 1 \tag{5}$$

o, por el contrario, cuando $y_k = 0$

$$h_0 = -\sum_{k=1}^{K}\left[(1-y_k)^{(n)} log\left(1-\hat{y}_k^{(n)}\right)\right]; \quad \forall y_k = 0 \tag{6}$$

Combinando ambas ecuaciones, tenemos que la CE es definida de la siguiente manera:

$$H = -\frac{1}{N}\sum_{n=1}^{N}\sum_{k=1}^{K}\left[y_k^{(n)} log\left(\hat{y}_k^{(n)}\right) + (1-y_k)^{(n)} log\left(1-\hat{y}_k^{(n)}\right)\right]; \quad \forall y_k = 1 \tag{7}$$

En clasificación, una medida de mucha utilidad es la tasa de acierto (*Acc*, del inglés *accuracy*) y de error (*Err*) ambas medidas permiten medir la tasa de acierto y de error para un determinado clasificador.

$$Acc = \frac{1}{N}\sum_{n=1}^{N}\mathbb{I}(y^{(n)} = \hat{y}^{(n)}) \tag{8}$$

$$Err = \frac{1}{N}\sum_{n=1}^{N}\mathbb{I}(y^{(n)} \neq \hat{y}^{(n)}) \tag{9}$$

Siendo $\mathbb{I}(\cdot)$ la función indicadora y $\hat{d}^{(n)}$ es la decisión a partir de $\hat{y}^{(n)}$ *i.e* $\hat{d} = \hat{y}^{(n)} \geq \eta$ siendo η una frontera de desición, *e.g* si la función de activación es la tangente hiperbólica entonces un valor apropiado para η es 0 o, si la función de activación es una sigmoide entonces $\eta = 0.5$.

4.1.3 Funciones de coste sensitivas al desbalanceo

En el caso de bases de datos balanceadas, una red neuronal puede ser entrenada para optimizar una basta cantidad de funciones de coste, ejemplos de ellos son las siguientes:

$$\mathcal{J} = \frac{1}{N}\sum_{n=1}^{N} l(y^{(n)}, \hat{y}^{(n)}) \tag{10}$$

donde $l(\cdot)$ es una función que mide la discrepancia entre la etiqueta verdadera y la estimada para la muestra n. Un ejemplo de esto es el MSE (o cuando se trata de datos binarios la Cross Entropía o Semicross Entropía).

Sin embargo, la expresión de la Ec. 10 no toma en cuenta el desequilibrio en las proporciones de los datos en cada clase. Por ello, y dentro del amplio abanico de opciones disponibles que la literatura ofrece para atacar este problema, optaremos por una mejor estrategía de entrenar máquinas para que tomen en cuenta esta sobre- y sub-representación de las clases y utilizaremos una función de coste sensitivo (Cost Sensitive Functions) al desbalanceo. Esto implica diseñar una matriz de costes cuyos elementos indican la penalidad en la que se incurre cuando se hacen decisiones. La matriz de costes C presentada en la Ec. 11 y que para el caso binario se tiene:

$$C = \begin{bmatrix} c_{00} & c_{01} \\ c_{10} & c_{11} \end{bmatrix} \tag{11}$$

donde c_{ij} indica el coste en el que se incurre cuando se decide que una muestra dada pertenece a la clase i cuando la correcta es j. Naturalmente, podemos considerar que no se incurre en coste alguno cuando se decide correctamente, *i.e.* $c_{00} = c_{11} = 0$. En cuanto a los términos c_{01} y c_{10} podemos hacer que incluyan información sobre las probabilidades a *a priori* de cada clase Ec. 12

$$C = \begin{bmatrix} 0 & \frac{\alpha}{P_1} \\ \frac{1-\alpha}{P_0} & 0 \end{bmatrix} \tag{12}$$

donde P_0 y P_1 representan las probabilidades a priori de la clase mayoritaría y minoritaria, respectivamente. Una ventaja clara de esta formulación es que los valores para c_{ij} queda representado por un único hiperparámetro α.

Introduciendo el término c_{ij} en nuestra formulación resulta en:

$$\mathcal{J} = \frac{1}{N_0} c_{10} \sum_{n=1}^{N_0} l(y^{(n)}, \hat{y}^{(n)}) + \frac{1}{N_1} c_{01} \sum_{n=1}^{N_1} l(y^{(n)}, \hat{y}^{(n)}) \tag{13}$$

$$\mathcal{J} = \frac{1}{N_0} \frac{1-\alpha}{P_0} \sum_{n=1}^{N_0} l(y^{(n)}, \hat{y}^{(n)}) + \frac{1}{N_1} \frac{\alpha}{P_1} \sum_{n=1}^{N_1} l(y^{(n)}, \hat{y}^{(n)}) \tag{14}$$

Finalmente, incorporando la información a priori del porcentaje de desbalanceo se obtiene:

$$\mathcal{J} = (1-\alpha)\sum_{n=1}^{N_0} l(y^{(n)},\, \hat{y}^{(n)}) + \alpha \sum_{n=1}^{N_1} l(y^{(n)},\, \hat{y}^{(n)}) \qquad (15)$$

Así, variando α es posible hacer que el clasificador sea entrenado para poder trabajar en distintos puntos de operación de la curva ROC. Es correcto indicar que el punto correcto de operación al que debería finalmente trabajar el clasificador está principalmente limitado por el tamaño del equipo técnico y de la respectiva habilidad de dicho equipo de tratar con éxito en número determinado de alarmas generada por el IDS tanto decisiones correctas como falsas alarmas.

4.2 Medidas de Bondad en Bases de Datos desbalanceadas

4.2.1 ROC

Consideremos el caso en que una máquina fue entrenada con datos cuyas etiquetas se muestran en la Tabla 2 bajo el nombre de y, y las etiquetas estimadas por un clasificador los que se muestran la columna bajo $\hat{y}$. El clasificador tiene una taza de error de solo el 10% (o taza de acierto de $Acc = 1 - Err = 0.90$ o $Acc(\%) = 90\%$); sin embargo, claramente el clasificador es uno que es inservible ya que todas las etiquetas estimadas son de la clase "0".

n	y	$\hat{y}$	$\mathbb{I}(y^{(n)} \neq \hat{y}^{(n)})$
1	1	0	0
2	0	0	1
3	0	0	1
4	0	0	1
5	0	0	1
6	0	0	1
7	0	0	1
8	0	0	1
9	0	0	1
10	0	0	1
		Err	$\frac{1}{10}$

Table 2: Tasa de error para un problema de clasificación con clases desbalanceadas.

Una medida más óptima es encontrar la matriz de confusión mostrada en la Tabla 3 y representa el ratio de muestras clasificadas correcta e incorrectamente en cada clase. Si el número de muestras de la clase positiva se designa como N_1 y el de las muestras negativas N_0. Entonces, podemos definir los elementos de la matriz de confusión como:

- **TPR** es el ratio de verdaderos positivos (del inglés *True Positive Rate*), y representa la relación entre el número de muestras clasificadas como positivas y que realmente pertenecen a la clase positiva.

$$TPR = \frac{\sum_{n=1}^{N_1} \mathbb{I}(\hat{y}^{(n)} = 1 \wedge y^{(n)} = 1)}{\sum_{n=1}^{N_1} \mathbb{I}(y^{(n)} = 1)} \tag{16}$$

- **FPR** es el ratio de falsos positivos (del inglés *False Positve Rate*). El número de muestras clasificadas como positivas pero que cuya clase verdadera es la negativa.

$$FPR = \frac{\sum_{n=1}^{N_0} \mathbb{I}(\hat{y}^{(n)} = 1 \wedge y^{(n)} = 0)}{\sum_{n=1}^{N_0} \mathbb{I}(y^{(n)} = 0)} \tag{17}$$

- **TNR** es el ratio de verdaderos negativos (del inglés *True Negative Rate*).

$$TNR = \frac{\sum_{n=1}^{N_0} \mathbb{I}(\hat{y}^{(n)} = 0 \wedge y^{(n)} = 0)}{\sum_{n=1}^{N_0} \mathbb{I}(y^{(n)} = 0)} \tag{18}$$

- **FNR** es el ratio de falsos negativos (del inglés *False Negative Rate*).

$$FNR = \frac{\sum_{n=1}^{N_1} \mathbb{I}(\hat{y}^{(n)} = 0 \wedge y^{(n)} = 1)}{\sum_{n=1}^{N_1} \mathbb{I}(y^{(n)} = 1)} \tag{19}$$

		y	
		p	n
$\hat{y}$	p	Verdaderos Positivos (TPR)	Falsos Positivos (FPR)
	n	Falsos Negativos (FNR)	Verdaderos Negativos (TNR)

(a)

		y	
		p	n
$\hat{y}$	p	0	0
	n	1	1

(b)

Table 3: Matriz de confusión a)Definición y b) Para el ejemplo de la Tabla 2.

La matriz de confusión para el ejemplo de la Tabla 2 se muestran en la Tabla 3b e indica que –evidentemente– la tasa de aciertos de las muestras positivas es 0.

4.3 Espacio ROC

La Característica de Operación del Receptor (ROC, del inglés *Receiver Operating Characteristics*) es una medida que toma relevancia sobre todo en el caso de los problemas desbalanceados –aunque no se limita a ello–.

Define un espacio de dos dimensiones en cuyo eje de las abcisas se encuentra la tasa de falsos positivos (también denominada probabilidad de falsa alarma P_{FA}), y en el eje de ordenadas la probabilidad de verdaderos positivos TPR (también denominada probabilidad de detección P_D).

En el caso de que un clasificador brinde salidas duras –como en el ejemplo de la Tabla 2– entonces solo es posible obtener un punto en el espacio ROC. Sin embargo, si el modelo del clasificador genera salidas de valores reales o también llamadas salidas blandas (limitadas o no a cierto rango de valores) entonces es posible variar la frontera de decisión η para obtener distintos valores de P_{FA} y P_D. Por ejemplo, la Tabla 4 muestra estimaciones blandas obtenidas con un clasificador cuyas salidas están limitadas al rango de valores $[0; 1]$. Para diferentes valores η podemos obtener distintas decisiones sobre la pertenencia de clase de la n-ésima muestra n.

Los puntos triviales $(P_{FA},\ P_D) = (1, 1)$ y $(P_{FA},\ P_D) = (0, 0)$ se encuentran cuando $\eta = -\infty$ y $\eta = +\infty$, respectivamente.

			$\hat{d} = \mathbb{I}(\hat{y} > \eta)$				
n	y	$\hat{y}$	$\eta = +\infty$	$\eta = 0.85$	$\eta = 0.75$	$\eta = 0.35$	$\eta = -\infty$
1	1	0.90	0	1	1	1	1
2	1	0.80	0	0	1	1	1
3	1	0.50	0	0	0	1	1
4	1	0.70	0	0	0	1	1
5	1	0.50	0	0	0	1	1
6	0	0.40	0	0	0	1	1
7	0	0.60	0	0	0	1	1
8	0	0.30	0	0	0	0	1
9	0	0.20	0	0	0	0	1
10	0	0.10	0	0	0	0	1
		P_D	0	1/5	2/5	1	1
		P_{FA}	0	0	0	2/5	1

Table 4: Puntos del espacio ROC para distintos valores del nivel de decisión η.

Debemos resaltar por lo tanto que el espacio ROC mide la habilidad del clasificador para separar las clases. El clasificador perfecto tendrá como parte del espacio ROC el punto formado por $P_D = 1$ y $P_{FA} = 0$.

4.4 AUC

La curva ROC es una herramienta objetiva para comparar clasificadores en cuanto a su capacidad de separar muestras de dos clases, sin embargo no es una métrica para poder comparar dichos clasificadores; además en algunos casos es difícil decidir cuál clasificador es mejor que otro, un ejemplo de ello es la comparación de dos clasificadores A y B a través de sus curvas ROC, Fig. 2 en donde a simple vista no es fácil observar que el AUC para el clasificador A $AUC = 0.85$ es mayor que el AUC para el clasificador B $AUC = 0.80$

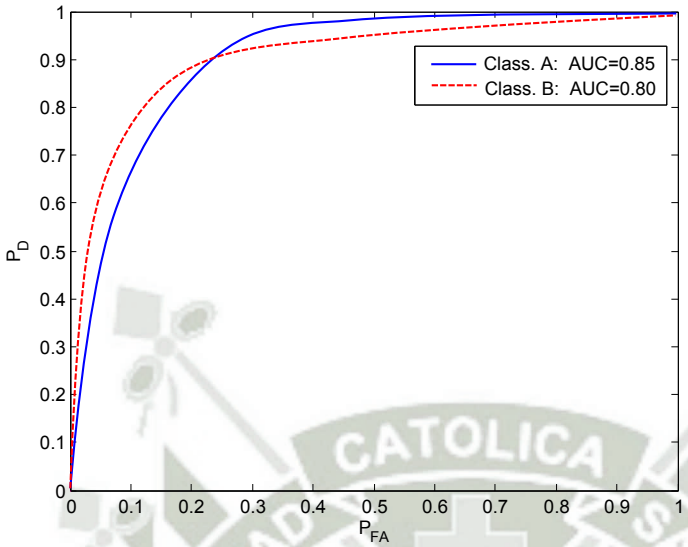

Figure 2: Espacio ROC para dos clasificadores A y B. Visualmente, no es fácil observar que el clasificador A es mejor que el clasificador B. El AUC es el valor de mérito que nos permite comparar ambos clasificadores e indicar que A es mejor que B.

4.5 Clasificadores

Como se mencionó, un clasificador es una función que tiene como argumento a los datos $\mathbf{x}$ y realiza una estima $\hat{y}$ de las etiquetas y.

$$\hat{y} = f(\mathbf{x}) \tag{20}$$

Este trabajo de Tesis se centra en los clasificadores tipo red neuronal, por ello, pondremos más énfasis al desarrollo de estas máquinas tanto redes neuronales estandard como a los algoritmos del estado del arte que permiten obtener redes profundas (DNN, del inglés *Deep Neural Net*). También, al final de esta Sección se menciona escuetamente otros clasificadores muy utilizados pero que no son parte de este trabajo.

4.6 Redes Neuronales

Las NNs son modelos matemáticos no lineales que poseen en su estructura muchas unidades simples llamadas neuronas. Las NNs están inspiradas en las neuronas biológicas. Y aunque fueron relegados por otro tipo de algoritmos no profundos, su importancia ha resurgido debido a los últimos avances que permiten obtener resultados muy superiores a los alcanzados por otros algoritmos.

Es necesario destacar que siendo las NNs el tema central de este trabajo de Tesis, pasaremos a presentarlas con mayor detalle, incluyendo un resumen breve de su desarrollo a través de los años.

4.6.1 Historia

Los primeros modelos de redes neuronales comenzaron a ser propuestos en el año 1943. El neurofisiólogo W. McCulloch y el matemático W. Pitts [61] proponen una teoría de cómo podrían trabajar las neuronas, escriben un modelo matemático que define formalmente la neurona como una máquina que además de binaria estaba dotada de múltiples entradas y salidas (modelo conexionista). Dicho modelo se construyó en base a circuitos eléctricos.

Años después, en 1949, Donald Hebb [62] definió dos conceptos importantes:

- El aprendizaje se localiza en las sinápsis o conexiones entre las neuronas.

- La información se representa en el cerebro mediante un conjunto de neuronas activas o inactivas.

Estos dos conceptos fundamentales se reflejan en la regla de Hebb que indica que los pesos de las sinápsis están en función de la interacción entre las neuronas pre- y post-sinápticas.

La primera conferencia sobre inteligencia artificial (1956) puso énfasis en la capacidad de las máquinas para simular el aprendizaje y propuestas de diferentes tipos de redes neuronales. Seguido de estos primeros pasos, continúan los avances resumidos a continuación:

- En 1959 se dió la primera aplicación de las redes a problemas reales, por ejemplo, filtros adaptativos para eliminar ecos en las líneas telefónicas. Otro hito importante se debió gracias a B. Widrow quién propuso la teoría sobre la adaptación neuronal y formuló el Adaline (Adaptative Linear Neuron) [63] y el Madaline (Multiple Adaline) [64].

- Rosemblatt en 1962 definió al Perceptrón [65] como un identificador de patrones binarios y también salida binaria. Dió lugar a la regla de aprendizaje delta, que permitía emplear señales continuas de entrada y salida.

- Posteriormente en 1969, Minsky y Papert [66] dieron una seria crítica del Perceptrón. Dicha crítica se fundamentaba en la naturaleza lineal y sus limitaciones para aprender funciones no lineales, esto provocó una caída en picada de las investigaciones en el área de las redes neuronales.

- En 1977, Anderson investiga el autoasociador lineal *brain state in a box* y los modelos de memorias asociativas [67].

- En los años 80, Rumelhart, McClelland y Hinton proponen diferentes modelos basados en la aseveración de que la mente aprende no de forma secuencial sino en paralelo (*Parallel Distributed Processing*) [68] .

- Para el año de 1982, Hopfield elabora un modelo de red que, a diferencia del clásico Perceptron unidireccional, tiene propagaciones en ambas direcciones de la red: hacia adelante (del inglés *fordward*) y hacia atrás (del inglés *backward*) [69].

- Kohonen en 1984 desarrolló los mapas de Kohonen [70] basadas en la observación biológica de que las neuronas adyacentes (tanto de capas inferiores y superiores o de salida) aprenden a representar patrones similares.
- Hinton y Sejnowski en 1986 desarrollaron la máquina de Boltzmann [71], una red probabilística cuya función de coste es modelada como una interconección entre todas las unidades de entrada y salida, así como conecciones entre unidades de la misma capa.
- Otro avance importante se da en 1987 cuando Grossberg analiza la inestabilidad de varios modelos presentados por Rumelhart y presenta su modelo ART *Adaptative Resonance Theory* [72].

Hoy en día gracias a los grupos de investigación de distintas universidades, compañías, entre otros, las redes neuronales se estan aplicando en distintos ámbitos de la industria, tecnología, informática, seguridad y telecomunicaciones.

4.6.2 Neurona biológica

Increiblemente, el cerebro humano está compuesto por aproximadamente cien mil millones de neuronas; y un estimado de más de 1000 sinápsis tanto a la entrada y salida de cada neurona. Y aunque el tiempo de conmutación de la neurona es casi un millón de veces mayor que el de las actuales computadoras, ellas tienen una conectividad miles de veces superior que las actuales supercomputadoras; siendo las neuronas y las conexiones entre ellas –sinápsis– las que constituyen la clave para el procesado de la información.

La neurona biológica es la célula fundamental del sistema nervioso, es una célula alargada, especializada en conducir impulsos nerviosos. En las neuronas se pueden distinguir tres partes fundamentales, que son:

- Soma o cuerpo celular: Se encuentra en el citoplasma. Es la parte más voluminosa de la neurona. Tiene una estructura esférica llamada núcleo. Éste contiene la información que dirige la actividad de la neurona.
- Dendritas: son prolongaciones cortas que se originan del soma neural. Su función es recibir impulsos de otras neuronas y enviarlas hasta el soma de la neurona.

- Axón: es una prolongación única y larga. En algunas ocasiones, puede medir hasta un metro de longitud. Su función es sacar el impulso desde el soma neuronal y conducirlo hasta otro lugar del sistema.

4.6.3 Neurona artificial

La neurona artificial se comporta como la neurona biológica pero de una forma muy simplificada. La Fig. 3 muestra el modelo simple inspirado en la biológica.

- Las entradas que reciben los datos de otras neuronas. En una neurona biológica corresponderían a las dendritas.
- Los pesos sinápticos w_{ij}: al igual que en una neurona biológica, se establecen sinápsis entre las dendritas de una neurona y el axón de otra, en la neurona artificial a las entradas que vienen de otras neuronas se les asigna un peso, (un factor de importancia). Este peso, que es un número, se modifica durante el entrenamiento de la red neuronal, y es aquí por tanto donde se almacena la infomación que hará que la red sirva para un propósito u otro.
- Cuerpo: el cuerpo de la neurona realiza operaciones en función a la información aportados por las dendritas. En la neurona artificial esta operación es la sumatoria.
- Cuello de axón: conecta el cuerpo con el axón. En la NN está representada por la función no lineal $f(z)$.

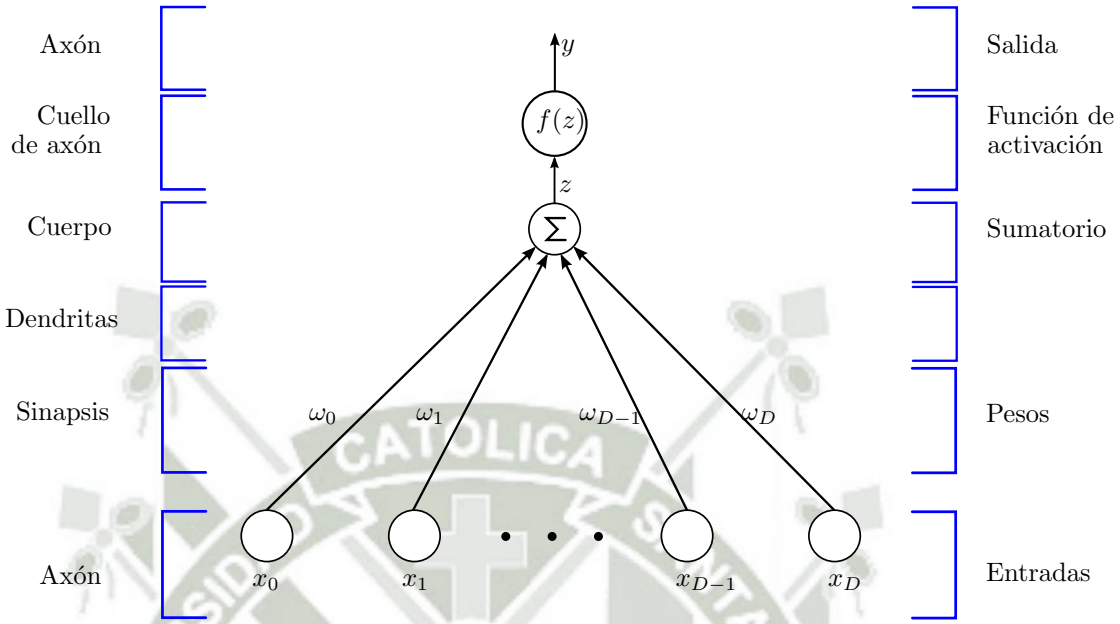

Figure 3: Modelo gráfico de la neurona artificial inspirada en las neuronas biológicas.

4.6.4 Funciones de activación

Las distintas funciones de activación $f(z)$ son utilizadas para restringir el rango de valores de la respuesta de la neurona z. Generalmente los rangos de valores se limitan a $[0; 1]$ o $[-1; 1]$, sin embargo otros rangos son posibles, por ejemplo limitar valores en el rango $[0; +\infty[$

La utilización de una determinada funcion de activación es dependiente del problema, es decir, unas funciones se adaptan de mejor o peor manera al problema a resolver. Sin embargo, dentro de la amplia gama de dichas funciones existen algunas que han mostrado resultados satisfactorios en diversas aplicaciones. Estas son:

- **Sigmoide**: Esta función asigna a cada valor de entrada $u \in \mathcal{R}$ un valor $\sigma(u)$ entre el rango de valores $[0; +1]$ siguiendo la Ec. 21:

$$\sigma(u) = \frac{1}{1 + \exp(-\beta u)} \; ; \; \forall u \in \mathcal{R}, \beta > 0 \tag{21}$$

- **Tangente hiperbólica**: Esta función de activación restringe los valores en el rango entre $[-1; +1]$ y está definida como:

$$\tanh(u) = \frac{1 - \exp(-\beta u)}{1 + \exp(-\beta u)} \; ; \; \forall u \in \mathcal{R}, \beta > 0 \tag{22}$$

- **Unidades Rectificadoras Lineales**: (ReLU, del inglés *Rectifier Linear Units*) es una activación que satura los valores negativos a cero pero no los positivos (estos últimos tienen una relación lineal) (Ec. 23)

$$ReLU(u) = max(0, \beta u) \; ; \; \forall u \in \mathcal{R}, \beta > 0 \tag{23}$$

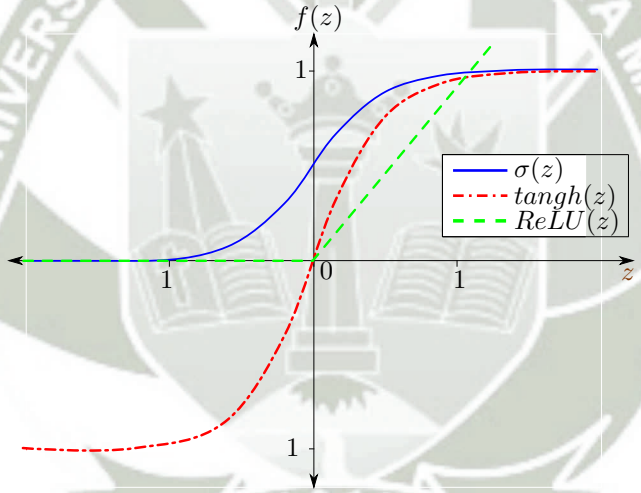

Figure 4: Funciones de activación $f(u)$: sigmoide $\sigma(u)$, tangente hiperbólica $tanh(u)$ y Rectificador Lineal $ReLU(u)$.

4.6.5 Perceptrón multicapa

A partir de la neurona artificial, Rosenblatt (1952-1962) desarrolló el modelo del perceptrón que consiste en una capa de neuronas con pesos y umbral ajustables.

La capa inferior es alimentada por los datos sensoriales $\mathbf{x}$ (Fig. 5) y esta información es pasada a una capa superior inmediata por medio de una función de activación aplicada a la salida de cada neurona (Ec. 24)

A su vez, los valores obtenidos a la salida de una capa intermedia $\mathbf{h}$ son utilizados para activar neuronas en capas superiores utilizando el mismo mecanismo (Fig. 5 y la Ec. 25).

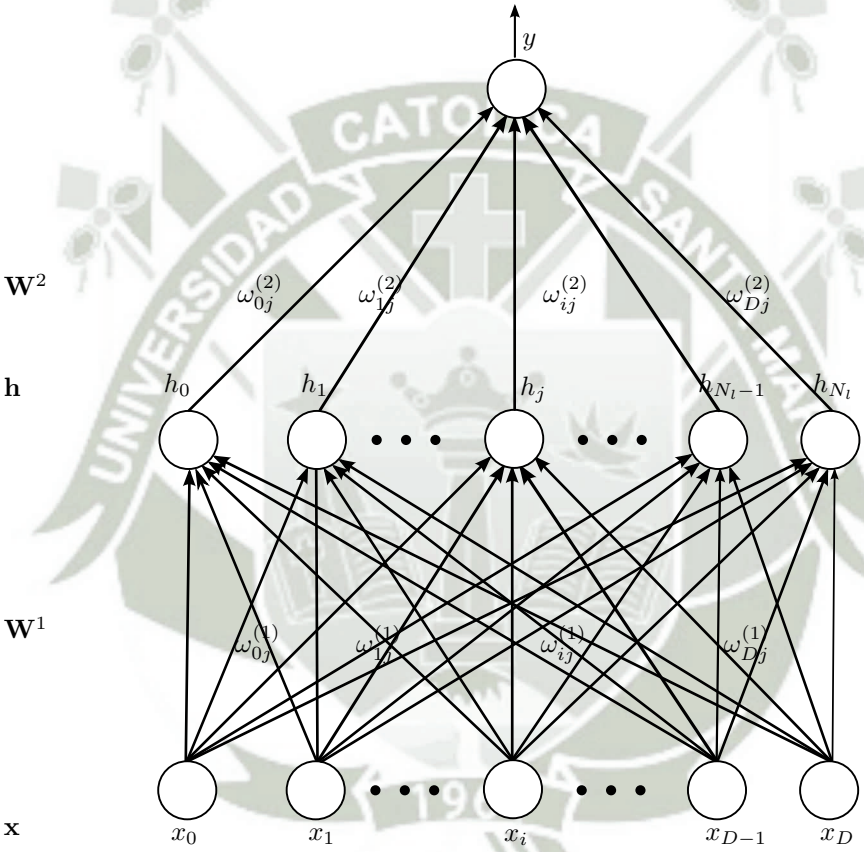

Figure 5: Modelo gráfico de la red neuronal con una capa oculta y una única salida.

$$\mathbf{h} = f_1(\mathbf{W}_1^T \mathbf{x}) \tag{24}$$

$$y = f_2(\mathbf{W}_2^T \mathbf{h}) \tag{25}$$

Dónde $f_l()$, siendo l el número de capa, es una función de activación como las representadas en la Sección 4.6.4.

4.6.6 Entrenamiento en NNs

El algoritmo de entrenamiento del perceptrón consiste en determinar el ajuste o cambio Δw_{ij} que se debe realizar a cada peso w_{ij} de tal manera que el error de generalización o función de coste $\mathcal{J}$ sea minimizado.

1. **Descenso por gradiente** El método de descenso por gradiente consiste encontrar el mínimo –o un mínimo local– de una función de coste derivable dado unos parámetros. Sea Θ una variable que representa al conjunto de parámetros que deseamos aprender ($\Theta = \mathbf{W}_1, \mathbf{W}_2$ en la Fig. 5), entonces dado un punto inicial Θ^τ podremos ir hacia un mínimo si seguimos la dirección contraria al gradiente [11]. El aprendizaje se regula gracias al parámetro η conocido como taza de aprendizaje (Ec. 26).

$$\Theta^{\tau+1} = \Theta^\tau - \eta \left. \frac{\partial \mathcal{J}}{\partial \Theta} \right|_{\mathbf{x}} \tag{26}$$

   La inicialización de los pesos puede ser de forma aleatoria, sin embargo, existen diversos mecanismos para inicializar dichos pesos de tal manera que favorezcan el entrenamiento y estos lleven a la máquina a converger a un mejor mínimo local como se detalla en apartado 4.7.1.

2. **BackPropagation** (BP) La técnica de BackPropagation permite propagar el gradiente de la función de coste capa a capa desde la superior a la inferior. Básicamente, esta técnica sigue la siguiente regla [73].

   - Dada una función de coste $\mathcal{J}$, y siendo $\mathbf{h}_l = f(\mathbf{W}_l^T \mathbf{h}_{l-1})$ la salida de la l-ésima capa con parámetros $\mathbf{W}_l$ con $\mathbf{h}_{l-1}$ como entrada ($\mathbf{h}_0 = \mathbf{x}$ y $y = f(\mathbf{W}_{L+1}^T \mathbf{h}_L)$ con L el número de capas ocultas).

- Luego, de forma iterativa a través de las capas se modifica los pesos de la siguiente manera:

$$\frac{\partial \mathcal{J}}{\partial \mathbf{W}_l} = \frac{\partial f\left(\mathbf{W}_l^T \mathbf{h}_{l-1}\right)}{\partial \mathbf{W}} \frac{\partial \mathcal{J}}{\partial \mathbf{h}_l} \tag{27}$$

$$\frac{\partial \mathcal{J}}{\partial \mathbf{h}_{l-1}} = \frac{\partial f\left(\mathbf{W}_l^T \mathbf{h}_{l-1}\right)}{\partial \mathbf{h}_{l-1}} \frac{\partial \mathcal{J}}{\partial \mathbf{h}_l} \tag{28}$$

4.7 Aprendizaje profundo

Una interminable cantidad de problemas reales necesitan funciones altamente no lineales que nos permitan relacionar los datos disponibles y sus determinadas clases. Y aunque las redes neuronales tradicionales permitieron avances para resolver tareas de muchos dominios de aplicación; sin embargo dados los problemas citados anteriormente en la práctica no es posible entrenar estructuras más profundas que contengan más de una capa oculta.

Deep Learning o aprendizaje profundo es un conjunto de algoritmos que permiten optimizar modelos con estructuras que contengan varias capas ocultas. Estos algoritmos han permitido extender el rango de aplicaciones de las NNs a muchos otros dominios con resultados muy notables. Por ejemplo, las redes convolucionales (CNN, del inglés *Convolutional Neural Networks*) son las máquinas utilizadas por defecto en aplicaciones de visión artificial, audio, etc.

En general, las redes neuronales profundas (DNN, del inglés Deep NN) permiten obtener múltiples niveles de abstracción a partir de los datos en bruto, *e.g.* con redes entrenadas con datos que consisten en imágenes, por ejemplo autos, las primeras capas aprenderán simples estructuras parecidas a los filtros de Gabor (Fig. GABOR), la capa siguiente utiliza esta primera extracción de características para formar relaciones más complejas. Así, cada capa superior relacionará conceptos más complejos. Sobre la capa superior es posible incluir algún clasificador y así permitir tareas de clasificación.

El uso de uno u otro algoritmo depende de la base de datos, sin embargo, dentro de la basta cantidad de algoritmos [18, 74] nosotros destacamos dos grupos: aquellos que inicializan la red en un punto cercano a un buen mínimo local y aquellos que permiten entrenar tales redes profundas utilizando mecanismos de regularización.

1. Algoritmos que utilizan pre-entrenamiento: Estos algoritmos permiten extraer características de los datos de forma no supervisada. Podemos anotar dos algoritmos bastante utilizados:
   - Auto-Encoders (AE)
   - Restricted Boltzmann Machines (RBM)
2. Algoritmos que utilizan regularización:
   - DropOut
   - Drop Connect

4.7.1 Algoritmos que utilizan pre-entrenamiento

1. **AE** estas máquinas en su forma básica tienen la estructura de una red neuronal de una sóla capa oculta –pero pueden incluir más de una–. Consiste en dos etapas muy bien definidas: una primera etapa conocida como el codificador (o *enconder*) y una segunda etapa llamada decodificador (o *decoder*). A groso modo, la idea es que la primera etapa extraiga características de las muestras, codificándolas a través de las unidades ocultas y luego decodifique o reconstruya las muestras desde dicho código. Por esto, la función de coste más común para medir el nivel de reconstrucción de las muestras es la función MSE. Pero cuando los datos son binarios, como en el caso de imágenes, es posible utilizar la función de Cross-Entropía.

   Históricamente, estos algorítmos tuvieron un uso limitado debido a que sólo podían tener un número de unidades ocultas inferior al número de dimensiones –pues de otro modo la función que aprendían era la matriz identidad–o ser entrenados utilizando algún mecanismo de regularización.

   El procedimiento para romper esta limitación fue dado en [53] y consiste en añadir ruido a los datos para luego tratar de reconstruirlos, evitándo aprender la función identidad (además, es posible demostrar que el ruido añadido forma un tipo de regularización de Liapunov). También, el hecho de contaminar los datos con ruido independiente y que este cambie en cada iteración del aprendizaje, se puede ver como si se aumentara un dato distinto cada vez –método que también es visto como una regularización de la red–.

2. **RBM** Estas redes son generativas. El objetivo es aprender la distribución de datos utilizando mecanismos de descenso por gradiente. Su estrutuctura consiste en dos capas, una primera capa que recibe los datos de entrada (capa visible), y una segunda capa latente u oculta $\mathbf{h}$ que aprende interacciones de los datos de entrada. Las relaciones dentro de la red son todas probabilísticas. Por ejemplo, para datos binarios y unidades ocultas binarias, la ecuación de energía es la siguiente:

$$E\left(\mathbf{x}, \mathbf{h}, \boldsymbol{\Theta}\right) = -\mathbf{a}^T\mathbf{x} - \mathbf{b}^T\mathbf{h} - \mathbf{x}^T\mathbf{W}\mathbf{h} \tag{29}$$

donde $\boldsymbol{\Theta} = (\mathbf{a}, \mathbf{b}, \mathbf{W})$ es el conjunto de parámetros que se debe optimizar.

Para un par $\mathbf{x}, \mathbf{h}$, la probabilidad de esta combinación de valores viene dada por

$$P\left(\mathbf{x}, \mathbf{h}, \boldsymbol{\Theta}\right) = \frac{1}{Z} e^{-E(\mathbf{x}, \mathbf{h}, \boldsymbol{\Theta})} \tag{30}$$

Podemos marginalizar y encontrar las probabilidades condicionadas de $\mathbf{x}$ dado $\mathbf{h}$ y viceversa, de tal manera que se permite analizar mejor las relaciones entre las capas visibles y ocultas. Estas relaciones vienen dadas por:

$$P(\mathbf{x}|\mathbf{h}) = \sigma\left(\mathbf{W}\mathbf{h}\right) \tag{31}$$

$$P(\mathbf{h}|\mathbf{x}) = \sigma\left(\mathbf{W}^T\mathbf{v}\right) \tag{32}$$

donde $\sigma(\cdot)$ es la función sigmoidal. El aprendizaje utiliza una técnica llamada Contrastive Divergence (CD) [52].

4.7.2 Algoritmos que utilizan regularización

1. **Dropout**

La dificultad de las redes neuronales de aprender niveles altos de no-linealidad por motivos de sobreajuste, o desvanecimiento del gradiente propagado desde la parte superior hasta la entrada de la red, entre otros. Existen muchos métodos que se han desarrollado para contrarestrar dicho efecto, dentro de ellos los más utilizados

son las regularizaciones en norma L_1 o L_2 –o variaciones de ellas–, combinaciones de las estimaciones de varias redes, entre otros.

Y aunque la media de las salidas de varias máquinas usualmente mejora el resultado ligeramente, el coste computacional en el que se incurre es en muchos casos imposible de asumir a no ser que se cuente con equipo especializado para dichas tareas, como por ejemplo utilizar Unidades de Procesado Gráfico (GPU).

Otro punto que juega en contra es que la combinación de un conjunto de máquinas brinda resultados mejores que máquinas simples cuando los elementos que forman el conjunto son diversos, esto es que aprenden diferentes versiones del problema. Esta diversidad es difícil de conseguir. Una forma de diversidad es aplicar distorsiones a los datos para formar bases de datos distintas a la original y hacer que cada máquina aprenda de cada nueva base de datos creada [74]. En algunos casos es relativamente fácil determinar el tipo de distorsión que nos permitirá encontrar una versión distinta de los datos y que sea útil para entrenar las máquinas buscando diversidad. Por ejemplo, cuando las bases de datos son imágenes las distorsiones válidas serían rotaciones de las imágenes, ampliaciones, reducciones, operaciones elásticas tales como elongaciones, entre muchos otros. Sin embargo, en problemas en donde no es fácil intuir el tipo de distorsión apropiado para generar nuevas bases de datos, como es el caso de los datos propios del entorno de ciberseguridad, las distorsiones comunes son la adición de ruido Gaussiano o ruido uniforme, así la diversidad producida en las máquinas resultantes no ayudarían a mejorar el error en clasificaión. Es más, algunas máquinas tienen el efecto de reducir el ruido Gausiano, eliminando así una posible diversidad inducida por estas bases de datos.

Dropout por otro lado, regulariza la red haciendo que solo algunas unidades participen en el entrenamiento en un momento determinado. Para realizar esto, las unidades de una capa se desactivan, es decir, que los valores de dichas unidades se ponen a cero con probabilidad p. Gráficamente, el resultado de esta operación equivaldría a entrenar una red reducida en cada instante del entrenamiento (Fig. 6). Dado que existe un total de N_h unidades ocultas entonces es posible obtener un total de 2^{N_h} combinaciones distintas en cada una de las N_l capas ocultas. Desde este punto de vista, dropout equivaldría a promediar el resultado de un número exponencial de redes angostas con pN_h unidades activas, aproximadamente.

En el caso del dropout, el hecho de eliminar ciertas unidades de forma aleatoria evita que las unidades de la red se coadapten entre ellas, i.e. las unidades aprenden versiones distintas de datos en cada iteración.

Debido a que en entrenamiento solo están activas una pequeña porción de unidades y en test están activas todas las unidades (Fig. 6), entonces es necesario realizar un escalado por p de los pesos que conectan a las unidades de capas inferiores con las superiores. No realizar dicho escalado produciría una saturación en las unidades de salida de cada capa.

El modelo matemático para una red dropout es el siguiente:

Sea l una determinada capa de las L capas ocultas que tiene la red, i.e $l \in \{0, 1, ..., L\}$, y $\mathbf{z}_l$ el vector de valores previo a la activación, el vector que contiene las unidades luego de realizar el dropout se puede expresar de la siguiente manera:

$$\tilde{\mathbf{h}}_l = \mathbf{r} \odot \mathbf{h} \tag{33}$$

donde $\odot$ representa una operación punto a punto y $\mathbf{r} \in R^{N_l \times 1}$ y $\mathbf{r} \sim Bernoulli(p)$.

$$\mathbf{z}_{l+1} = \mathbf{W}_{l+1}^T \tilde{\mathbf{h}}_l \tag{34}$$

y luego de realizar la activación $f_l(.)$

$$\mathbf{h}_{l+1} = f_l(\mathbf{z}_{l+1}) \tag{35}$$

En la Fig. 6 se muestra el esquema de una red neuronal sin utilizar dropout y utilizando la técnica de dropout.

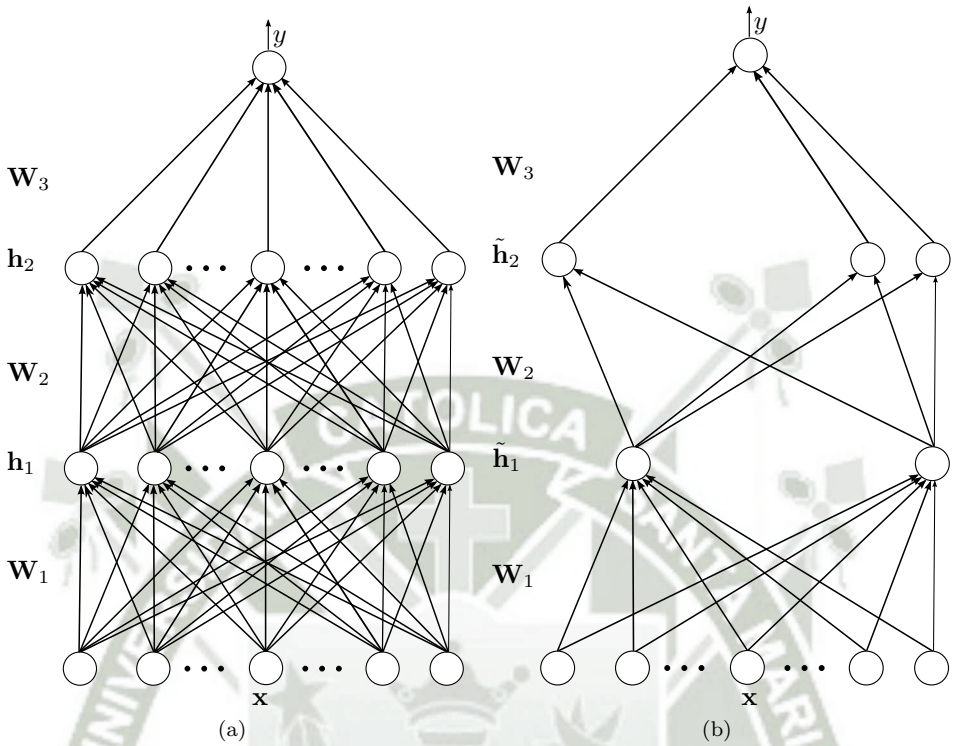

Figure 6: Red neuronal bajo ambos enfoques: a) estandar (sin dropout), en donde todas las neuronas ocultas están activas durante el entrenamiento y b) con dropout, en donde sólo pN_l unidades están activas en cada capa oculta durante el entrenamiento.

4.7.3 Regresor Logístico

La regresión logística es una técnica sencilla que nos permite evaluar la relación existente entre las muestras $\mathbf{x}$ y sus etiquetas y. El modelo matemático es el siguiente:

$$y = f(\mathbf{W}^T\mathbf{x}) \tag{36}$$

4.7.4 SVM

Las Máquinas de vectores Soporte (SVMs, del inglés *Support Vector Machine*) se crearon con la finalidad de resolver problemas de clasificación binaria, pero actualmente utilizadas para resolver

problemas de regresión, agrupamiento, y clasificación multiclase. Los campos en los que se ha utilizado han sido varios como reconocimiento de caracteres, visión artificial, categorización de textos e hipertextos.

4.7.5 Árboles de Desición

Un árbol de decisión está formado por un conjunto de nodos de decisión (interiores) y de nodos-respuesta (hojas):

- Nodo de decisión: El nodo de decisión es la parte fundamental de los árboles de decisión. Cada nodo tiene asociado uno de los atributos y tiene dos o más ramas que salen de él, cada una de ellas representando los posibles valores que puede tomar el atributo asociado.
- Nodo respuesta: nos devuelve la decisión del árbol con respecto al ejemplo de entrada.

4.7.6 Clasificador Bayesiano

El problema de clasificación supervisada consiste en asignar a una muestra descrita por un conjunto de características $\mathbf{x} = [x_1, x_2, ...x_D]^T$ a una de las K clases posibles, c_1, c_2, ..., c_K, tal que la probabilidad de la clase dados las características se maximiza:

$$i^* = \underset{c_i}{arg\,max}\, P\left(c_i \,|x_1, x_2, ..., x_D\right) \tag{37}$$

En general, estos algoritmos tienen el problema de no profundidad de sus arquitecturas, por ello es limitada la capacidad de aproximar relaciones complejas, como las existentes entre la mayor cantidad de datos y sus clases a las que pertenecen. Aún cuando es posible utilizar kernels, como en el caso de las SVMs, la máquina final es un estimador lineal. Por ello es necesario recalcar que los algoritmos propios del DL ayudan a construir máquinas más potentes que incurren en costes inferiores (*e.g.* menor tasa de error) a otros métodos no profundos.

5 Experimentos

Esta sección detalla la configuración experimental y los pasos a seguir para poder entrenar clasificadores de tipo DNN con dropout como método de regularización, empleando la base de datos DARPA KDD 1999, y así poder utilizarlas ya sea como un IDS o como parte de un IDS.

También, y con el objetivo de poder hacer comparaciones en cuanto a la mejora en el performance de las DNNs, primeramente se realiza el entrenamiento de NNs estándares, esto es, con una sóla capa oculta y BP para optimizar sus pesos.

Resumimos a continuación los pasos para poder obtener las máquinas que realicen la clasificación de los registros –o muestras–:

- Codificación de variables de la base de datos KDD DARPA 1999 en variables propicias para ser utilizadas en el entrenamiento de las redes neuronales.
- Validación de hiperparámetros: se hará la comparación de dos tipos de estructuras neuronales NN estandar y profunda, por ello, en cada caso, los hiperparámetros a ser evaluados son los siguientes:
  - NN: se validará el número de neuronas en la capa oculta
  - DNN: para este caso se validará porcentaje de dropout así como número de de capas ocultas. En cuanto al número de capas neuronas en la capa oculta, se fija al mismo número de neuronas en la capa oculta. Esto permitiría la comparación de dos estructuras con el mismo número de de unidades en la primera capa de abstracción de la máquina.

  Además, para selección de hiperparámetros se utilizará la técnica 10-fold cross validation.
- Una vez, los hiperparámetros en cada caso han sido seleccionados, se procede a entrenar máquinas tanto NNs como DNNs. La comparacin se realiza en términos de sus AUC en media sobre 10 simulaciones.

Para este trabajo se ha implementado el código en Phyton utilizando la libreria Keras, además de funciones propias y se muestran en mayor detalle en el Apéndice X.

5.1 Codificación de variables categóricas

Las variables categóricas se codifican utilizando la técnica de "one-hot-encoding", esto es, se genera un vector de longitud igual al número de variables categóricas de puros ceros excepto en la posición correspondiente a la variable categórica de una instancia. Las longitudes de los vectores resutantes se muestran en la Tabla 5

5.2 Selección de hiperparámetros

Los hiperparámetros son seleccionados utilizando la técnica de validación cruzada KFold. Esta técnica divide el conjunto de entrenamiento en K subconjuntos de forma aleatoria, luego se utiliza $K-1$ conjuntos de entrenamiento y el K-ésimo conjunto se utiliza como conjunto de validación, este procedimiento se repite cambiando en cada caso el conjunto de validación. Finalmente, se promedia los resultados sobre el conjunto de validación. La combinación de hiperparámetros que mejor medida de bondad ofrece se seleccionan como hiperparámetros de diseño. Para este trabajo, se ha utilizado 5 fold y como ya se mencionó arriba, se utiliza el AUC como medida de bondad.

5.2.1 Selección de hiperparámetros de la red neuronal

En este primer caso, se valida el número de neuronas N_h en la capa oculta. El resultado del proceso de validación se muestra en la Tabla 6 y Fig. 7. Se debe destacar que el valor de AUC alcanza un máximo desde $N_h = 50$. Seleccionaremos este valor ya que las diferencias con las redes con N_h mayores presentan solo ligeras diferencias.

Como podemos observar en la Tabla 6 y Figuras 7 a-e, las curvas ROC tanto en entrenamiento como en validación van de la mano llegando en todos los casos a valores AUC cercanos a 1 y a simple vista son imperceptibles detalles que se muestran en las curvas ROC. Por ejemplo, si observamos con mayor detenimiento la curva ROC cuando el conjunto de validación es el fold 4 (Fig. 7d) veremos irregularidades en la curva ROC, esto puede observarse en la Fig. 8a que muestra este detalle haciedo un acercamiento en el área de interés. Esta no convexidad en la curva ROC se debe a que la red neuronal se ve atrapada en algunos mínimos locales. Estos efectos se

	Características			
	Grupo	**Nombre**	**Tipo**	**Dimension resultante**
1	Básica	*duration*	real	1
2		*protocol_type*	categórica	3
3		*sevice*	categórica	70
4		*src_bytes*	categórica	11
5		*dst_bytes*	real	1
6		*flag*	real	1
7		*land*	binaria	1
8		*wrong_fragment*	real	1
9		*urgent*	real	1
10	Contenido	*hot*	real	1
11		*num_failed_logins*	real	1
12		*logged_in*	binaria	1
13		*num_compromised*	real	1
14		*root_shell*	real	1
15		*su_attempted*	real	1
16		*num_root*	real	1
17		*num_file_creations*	real	1
18		*num_shells*	real	1
19		*num_access_files*	real	1
20		*num_outboun d_cmds*	real	1
21		*is_hot_login*	binaria	1
22		*is_guest_login*	binaria	1
23	Tráfico	*count*	real	1
24		*serror_rate*	real	1
25		*rerror_rate*	real	1
26		*same_srv_ rate*	real	1
27		*diff_srv_r ate*	real	1
28		*srv_count*	real	1
29		*srv_serror _rate*	real	1
30		*srv_rerror _rate*	real	1
31		*srv_diff_h ost_rate*	real	1
32	Host	*dst_host_count*	real	1
33		*dst_host_srv_co unt*	real	1
34		*dst_host_same_ srv_rate*	real	1
35		*dst_host_diff_sr v_rate*	real	1
36		*dst_host_same_ src_port_rate*	real	1
37		*dst_host_srv_di ff_host_rate*	real	1
38		*dst_host_serror _rate*	real	1
39		*dst_host_srv_se rror_rate*	real	1
40		*dst_host_rerror _rate*	real	1
41		*dst_host_srv_re rror_rate*	real	1
			Total	122

Table 5: Características de la base de datos DARPA. Se dividen en 4 grupos: Básicas, de Contenido, de Tráfico y de *Host*, además se indica el formato de atributo (real, binaria, categórica) y su respectiva codificación (real o bits).

ven contrarestados al promediar curvas ROC de todos los folds, tal y como se muestra en la Fig. 8b (dicha figura también muestra una versión aumentada cerca al punto de interés $(P_{FA}, P_D) = (0, 1)$). Además se ha incluido la desviación estandard de los resultados tanto para P_{FA} como para P_D y distintos valores de $\mathbf{x}$.

	AUC(%)					
N_h	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Media Tr.
10	98.506	98.525	99.233	98.489	99.221	98.795
20	99.688	99.674	99.615	99.675	99.669	99.664
30	99.631	98.877	98.881	99.696	98.886	99.194
50	99.808	99.735	99.652	99.756	99.818	99.754
70	99.804	99.709	99.835	99.776	99.734	99.772
90	99.766	99.792	99.843	99.781	99.753	99.787

(a)

	AUC(%)					
N_h	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Media Val.
10	98.430	98.424	99.223	98.525	99.261	98.773
20	99.617	99.654	99.569	99.680	99.660	99.636
30	99.582	98.788	98.916	99.672	98.858	99.164
50	99.734	99.692	99.595	99.747	99.854	99.724
70	99.741	99.644	99.777	99.779	99.720	99.732
90	99.696	99.734	99.772	99.753	99.769	99.745

(b)

Table 6: Resultado para la red neuronal de una capa oculta en trminos AUC para los 5 folds y la media respectiva sobre todos los folds: a) Para entrenamiento y b) para validacin.

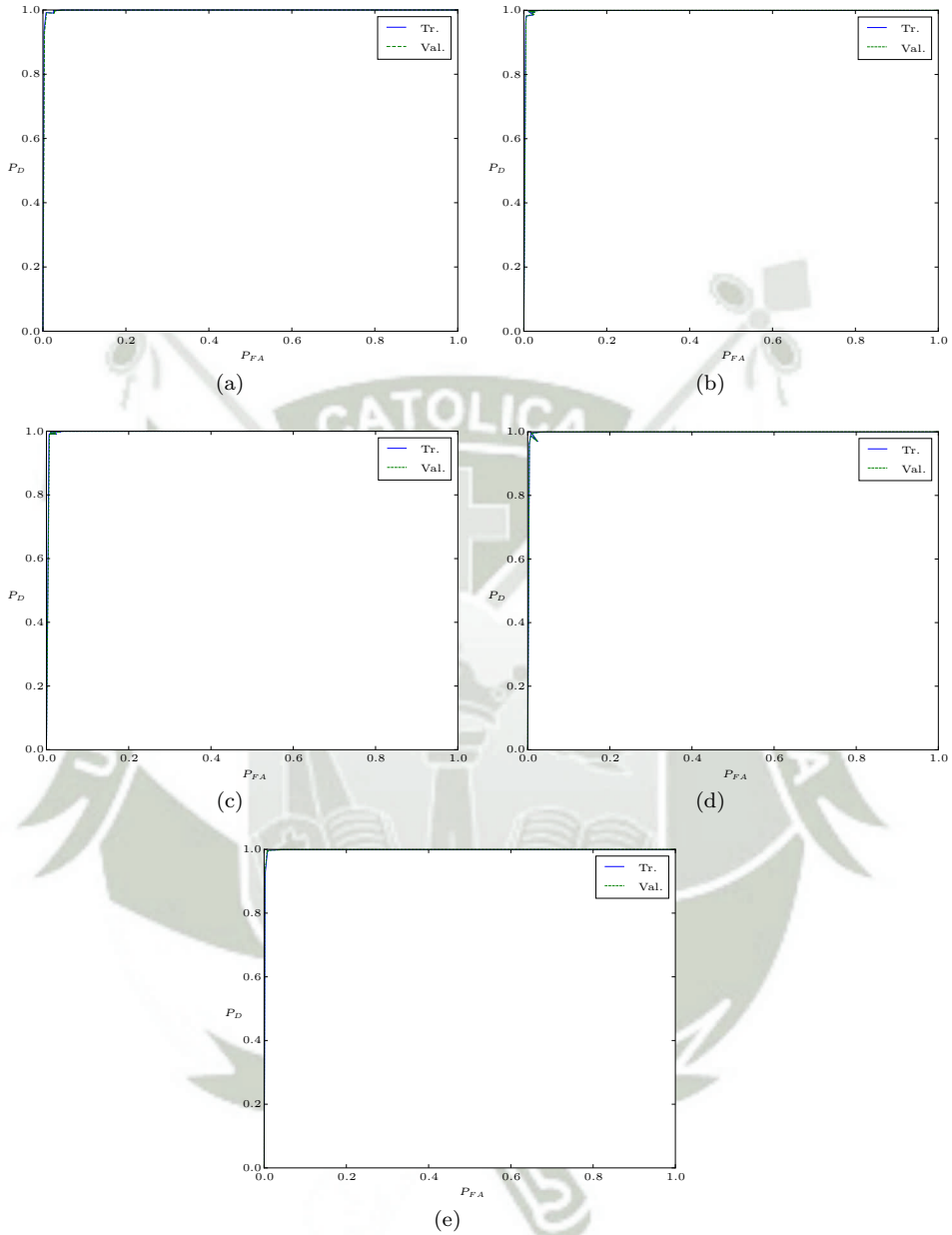

Figure 7: Curvas ROC para los 5 folds: a-e) ROC para los conjuntos de entrenamiento (línea sólida) y sus respectivos conjuntos de validación (línea entrecortada).

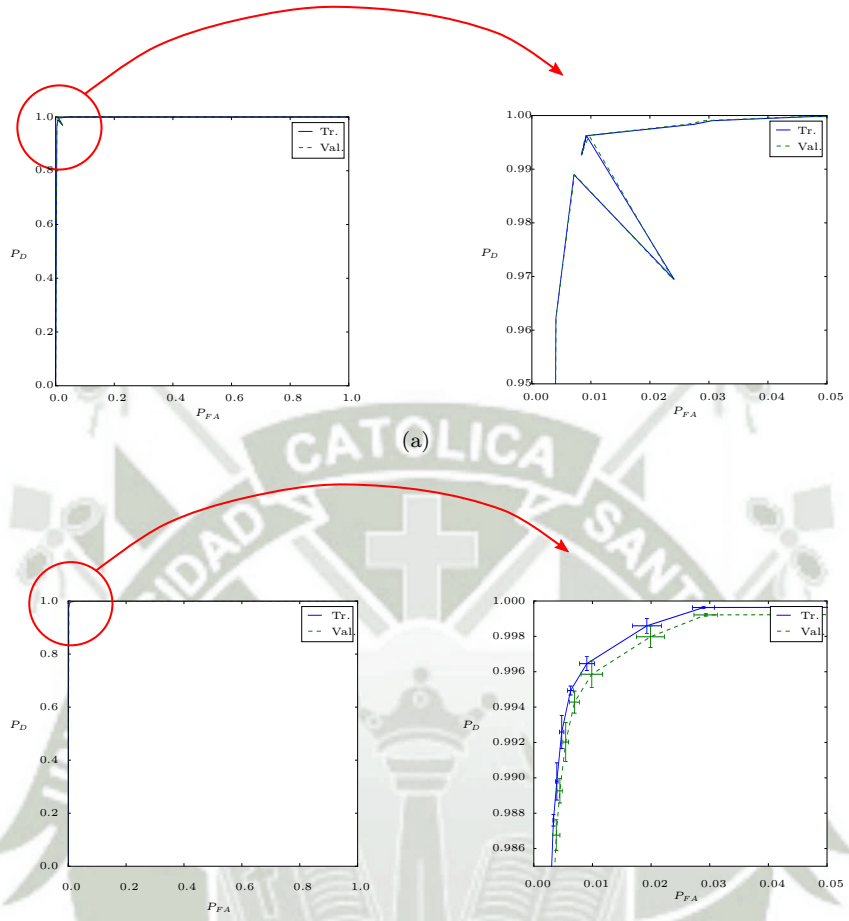

Figure 8: Detalle de las curvas ROC para el conjunto de validación del Fold 4 y la media de todas las curvas ROC de los conjuntos de validación. (a) El Fold 4 presenta una irregularidad en esquina superior izquierda, y b) La irregularidad se ve atenuada al promedias las curvas ROC de los 5 folds. Además, se presenta la desviacin estandar tanto en el eje de P_{FA} como en el P_D.

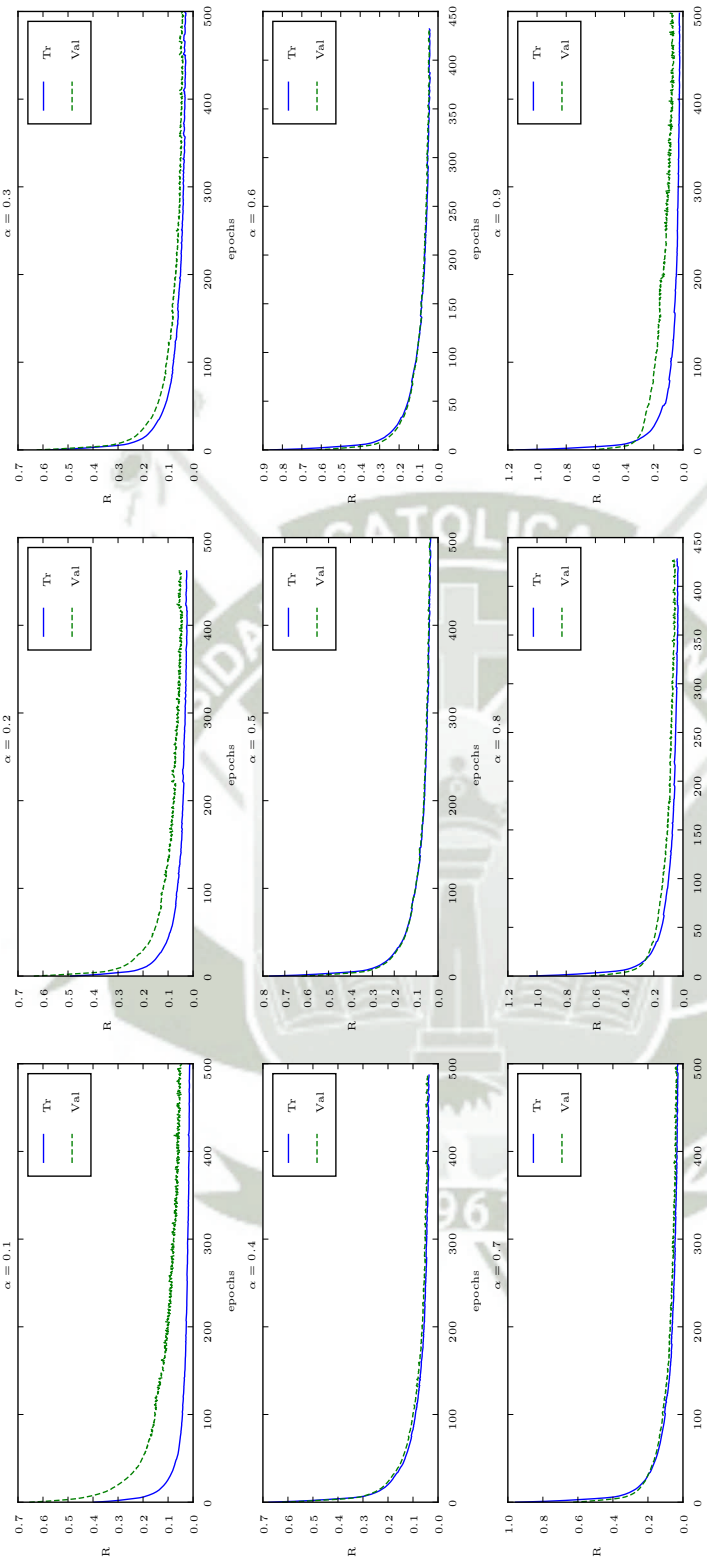

Figure 9: Evolución de la función de coste en validación (conjunto de validación fold 1) para distintos valores de $\alpha \in [0.10.9]$.

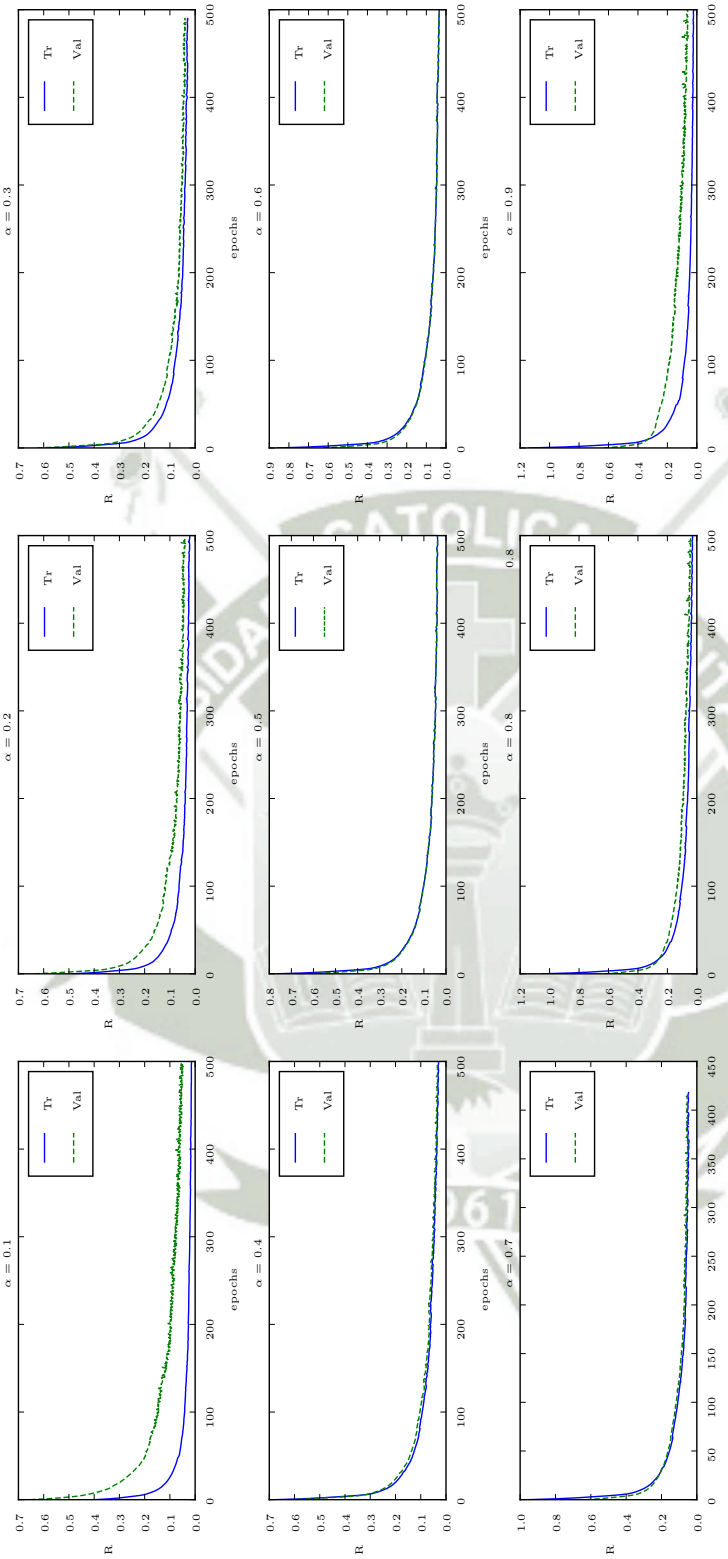

Figure 10: Evolución de la función de coste en validación (conjunto de validación fold 2) para distintos valores de $\alpha \in [0.10.9]$.

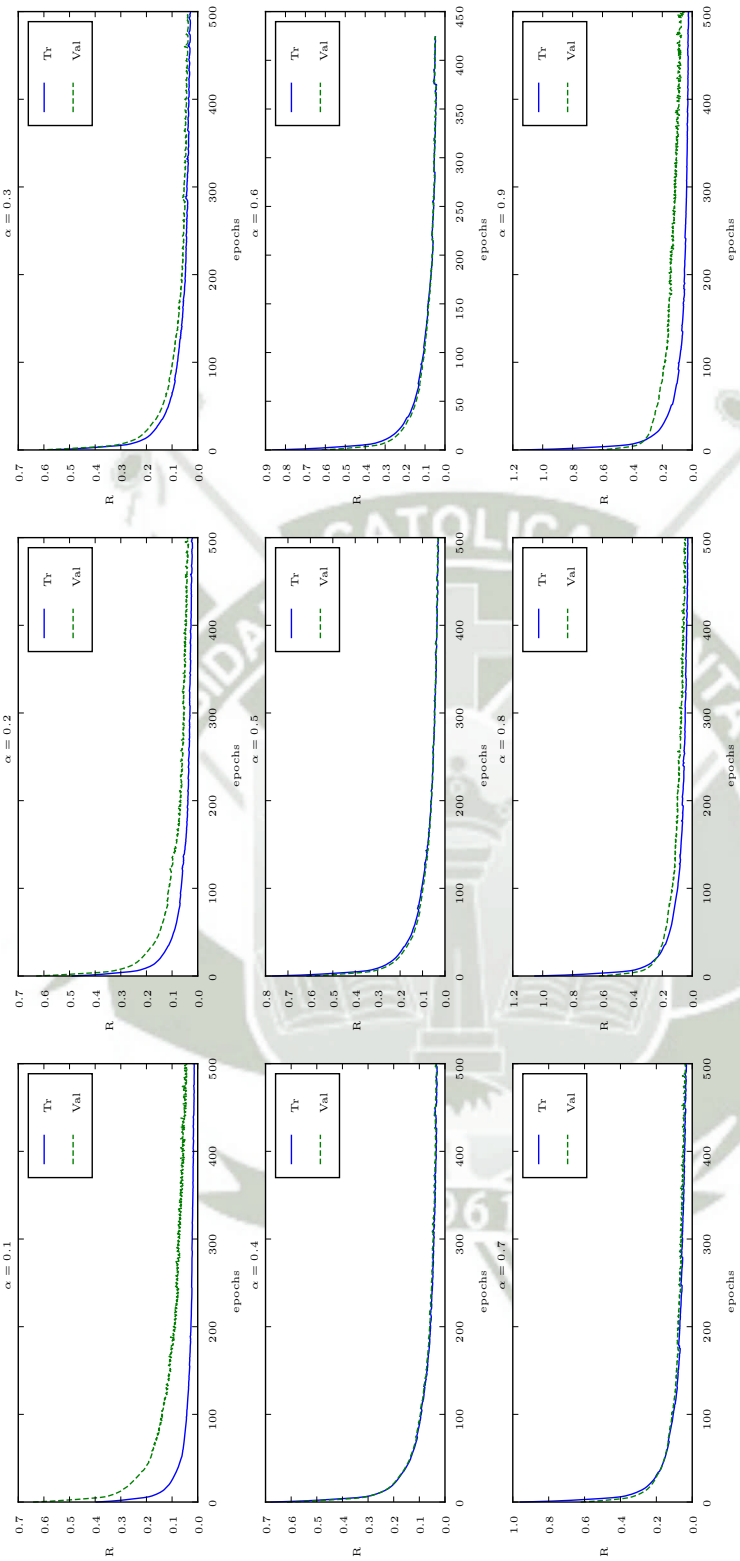

Figure 11: Evolución de la función de coste en validación (conjunto de validación fold 3) para distintos valores de $\alpha \in [0.10.9]$.

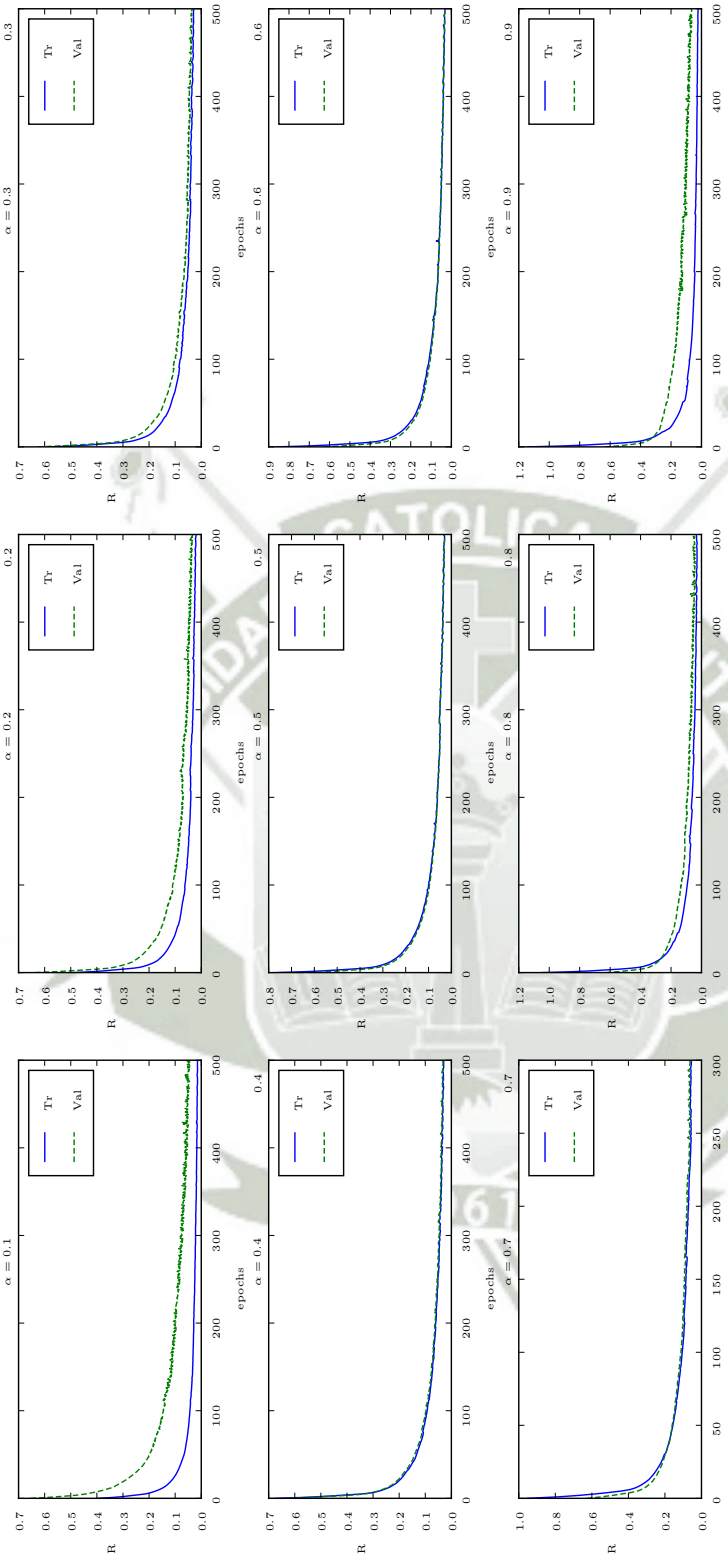

Figure 12: Evolución de la función de coste en validación (conjunto de validación fold 4) para distintos valores de $\alpha \in [0.10.9]$.

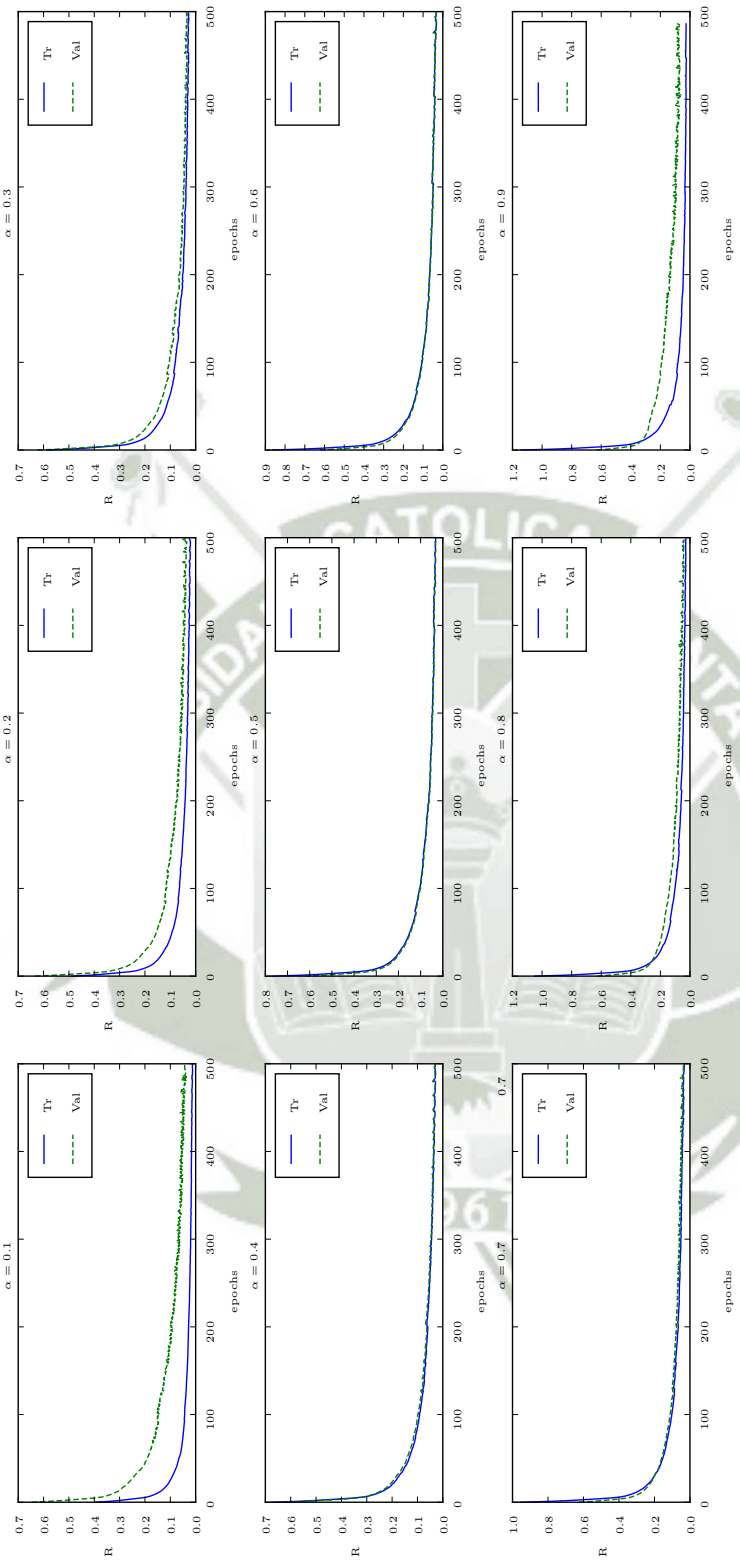

Figure 13: Evolución de la función de coste en validacón (conjunto de validación fold 5) para distintos valores de $\alpha \in [0.10.9]$.

5.2.2 Selección de hiperparámetros de la red neuronal utilizando dropout

En este segundo caso, se valida el número de capas ocultas $N_l = \{1; 2; 3\}$ y el porcentaje de dropout $d = \{0.25; 0.50; 0.75\}$. El número de neuronas ocultas se fija al valor determinado en la etapa anterior, es decir se ha utilizado el mismo valor de neuronas ocultas que la red neuronal entrenada por métodos convencionales $N_h = 50$. De esta manera, además de evitar la validación de este parámetro que de por sí es muy costoso computacionalmente, es posible comparar los dos tipos de arquitecturas en términos de la misma capacidad de extracción de características en la capa inferior.

Los resultados de la validación se muestran en la Tabla 7. El mejor valor en términos de AUC está dado por la combinación de dropout $d = 0.25$ y $N_l = 2$ capas ocultas.

				AUC			
N_l	d	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Media Val.
1	0.25	99.603	99.617	99.632	99.600	99.69	99.628
	0.50	99.688	99.688	99.620	99.731	99.691	99.684
	0.75	99.267	99.506	98.680	99.534	99.385	99.274
2	0.25	99.679	99.766	99.639	99.723	99.854	**99.732**
	0.50	99.662	99.742	99.698	99.773	99.568	99.689
	0.75	99.424	99.536	99.638	99.576	99.561	99.547
3	0.25	99.558	99.709	99.700	99.833	99.736	99.707
	0.50	99.719	99.668	99.530	99.763	99.665	99.669
	0.75	99.458	99.439	99.442	99.518	99.558	99.483

Table 7: Resultado para la red neuronal regularizada con dropout a una taza d y N_l capas ocultas. Los resultados se presentan el AUC para los 5 folds y la media respectiva sobre todos los folds para validación. El mejor valor en validación está resaltado en negrita y corresponde a la combinación de $d = 0.25$ con $N_l = 2$

5.3 Control de entrenamiento

Las Figuras 14 - 18 muestra la evolución de la función de coste tanto para entrenamiento como para validación. Es posible observar que la red en general converge a una buena solución y en ningún caso

existe sobreajuste sobre el conjunto de validación. Esto también muestra que ambos conjuntos pertenecen a la misma distribución.

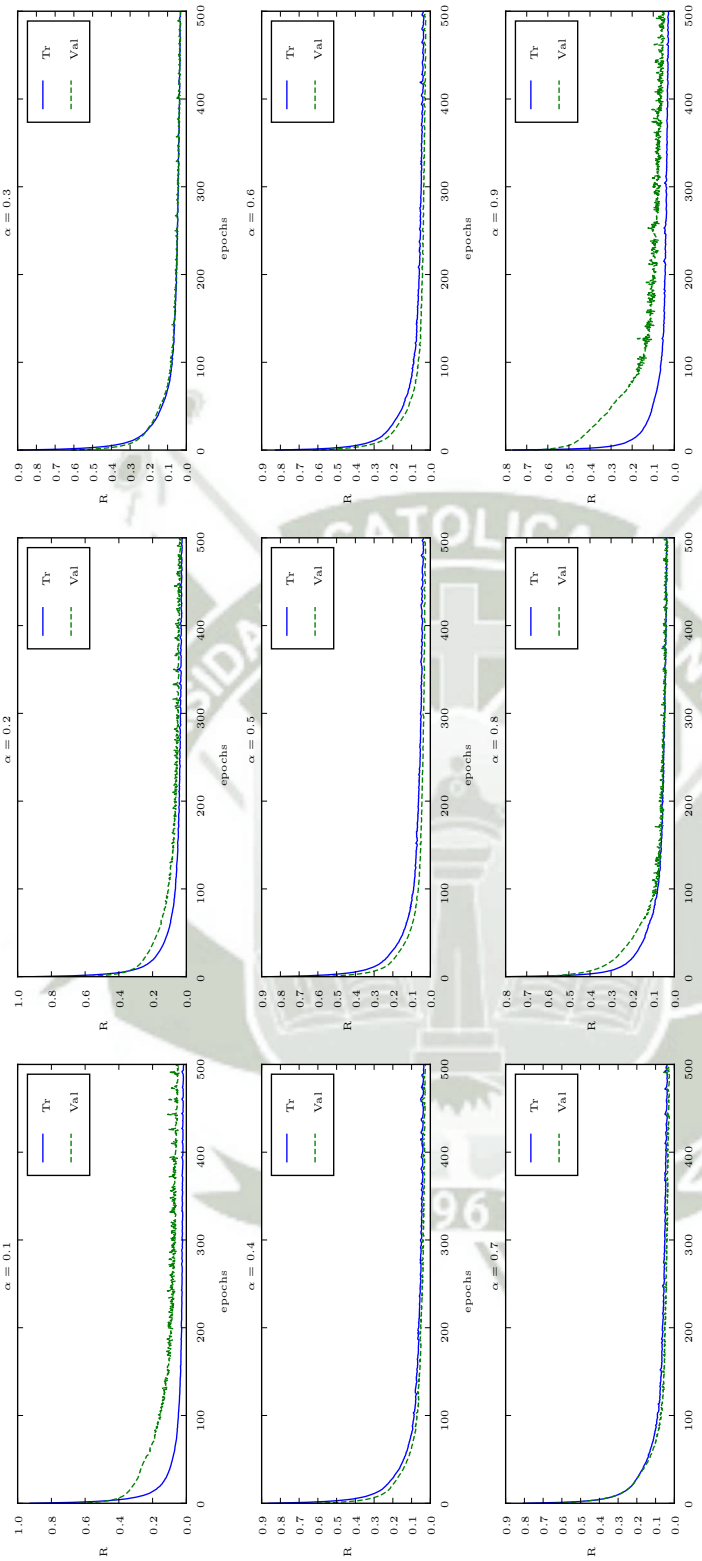

Figure 14: Evolución de la función de coste en validación (conjunto de validación fold 1) utilizando dropout $d = 0.25$ para distintos valores de $\alpha \in [0.10.9]$.

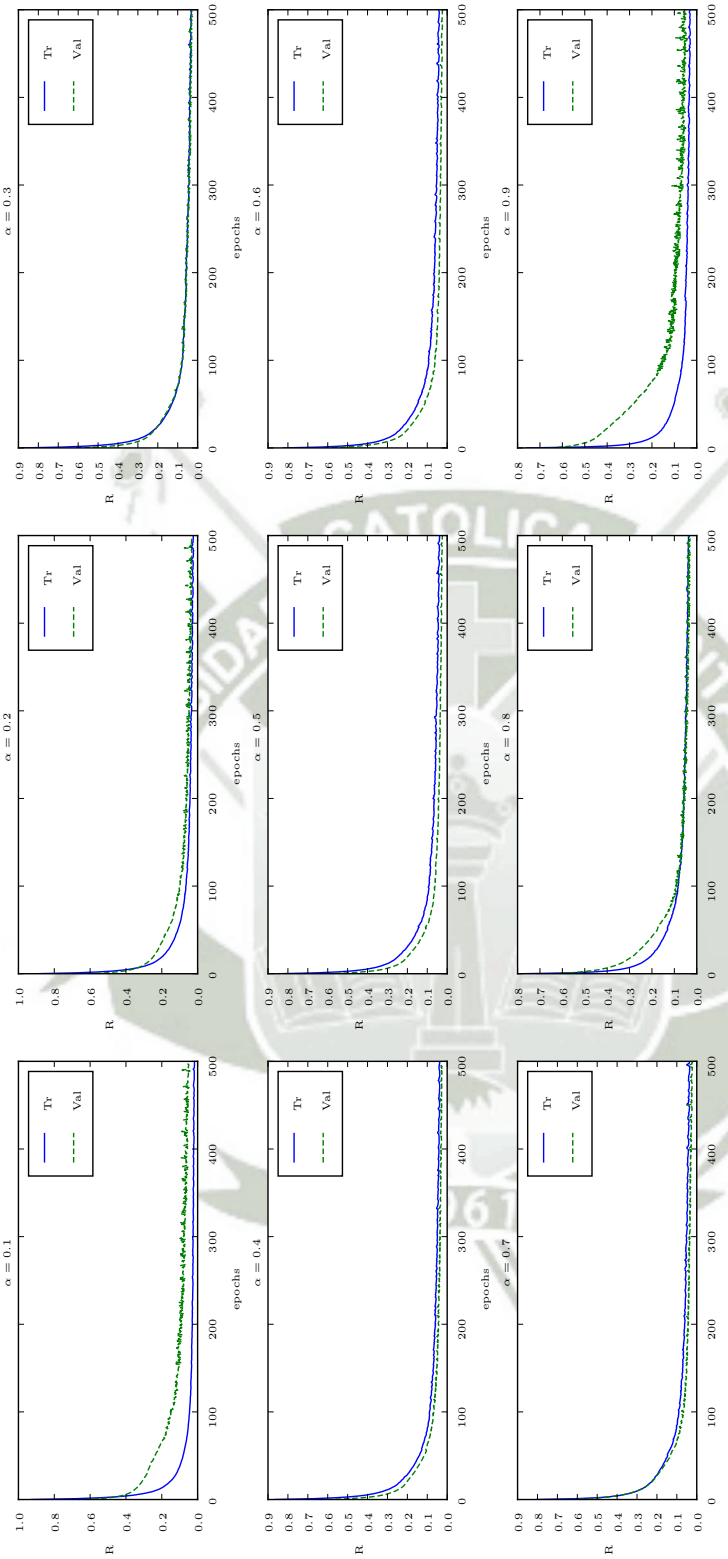

Figure 15: Evolución de la función de coste en validación (conjunto de validación fold 2) utilizando dropout $d = 0.25$ para distintos valores de $\alpha \in [0.10.9]$.

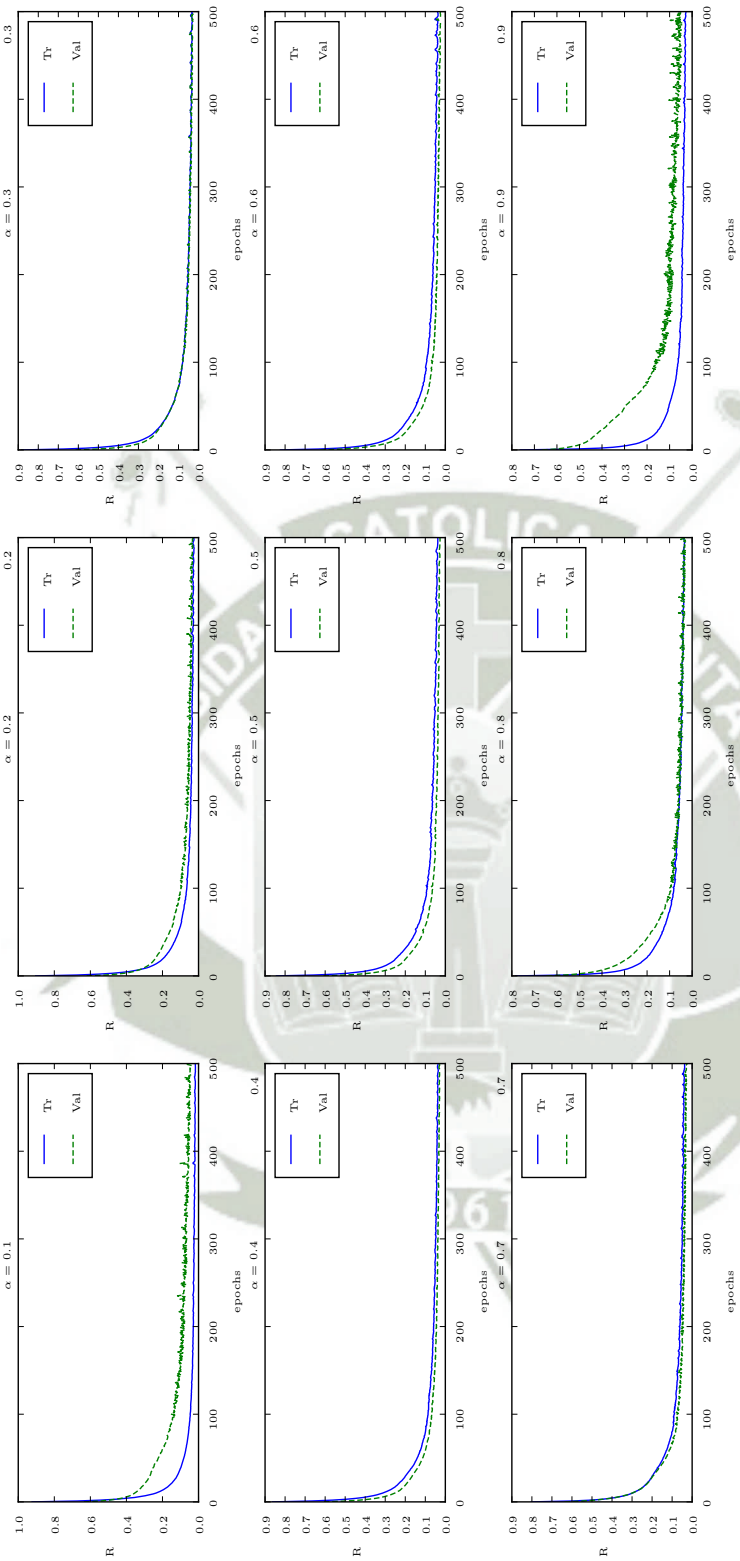

Figure 16: Evolución de la función de coste en validación (conjunto de validación fold 3) utilizando dropout $d = 0.25$ para distintos valores de $\alpha \in [0.10.9]$.

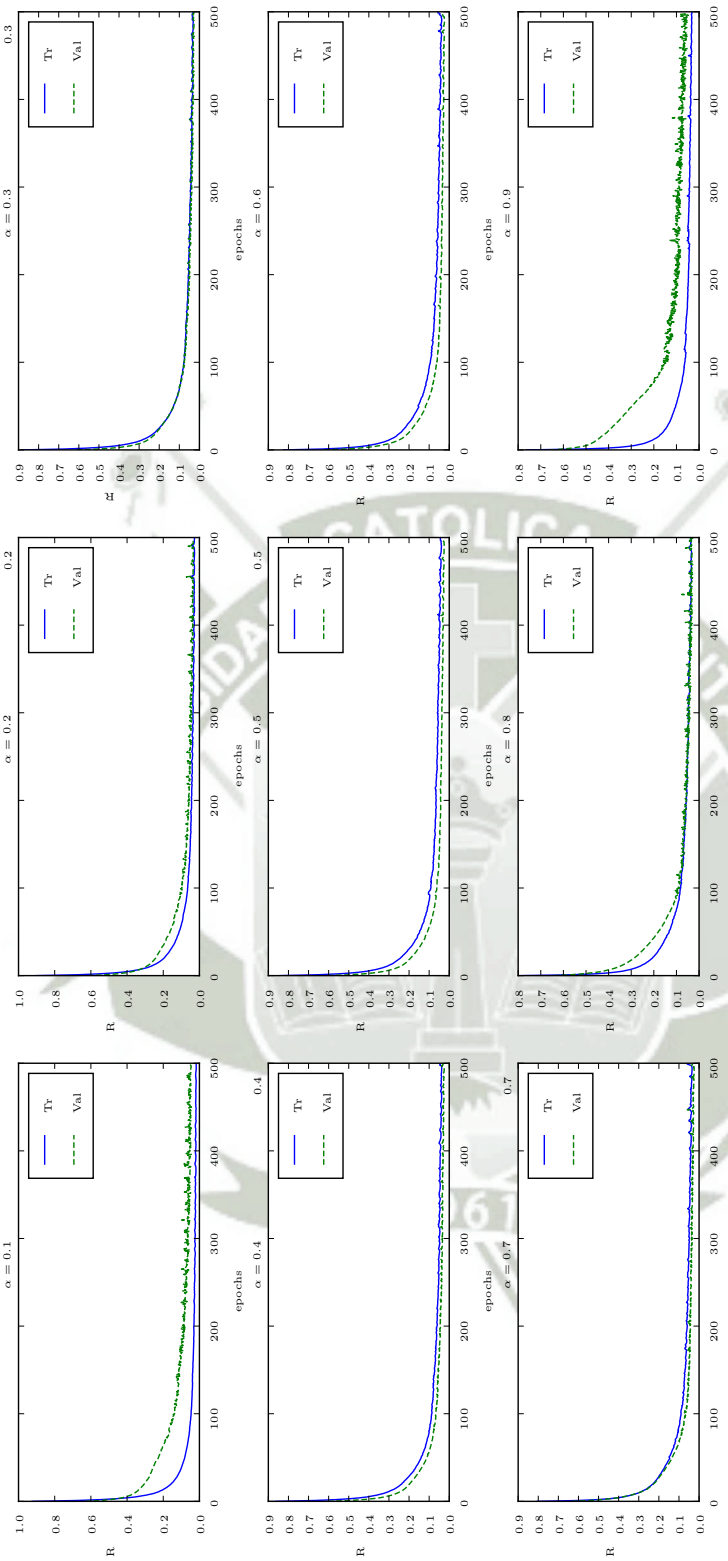

Figure 17: Evolución de la función de coste en validación (conjunto de validación fold 4) utilizando dropout $d = 0.25$ para distintos valores de $\alpha \in [0.10.9]$.

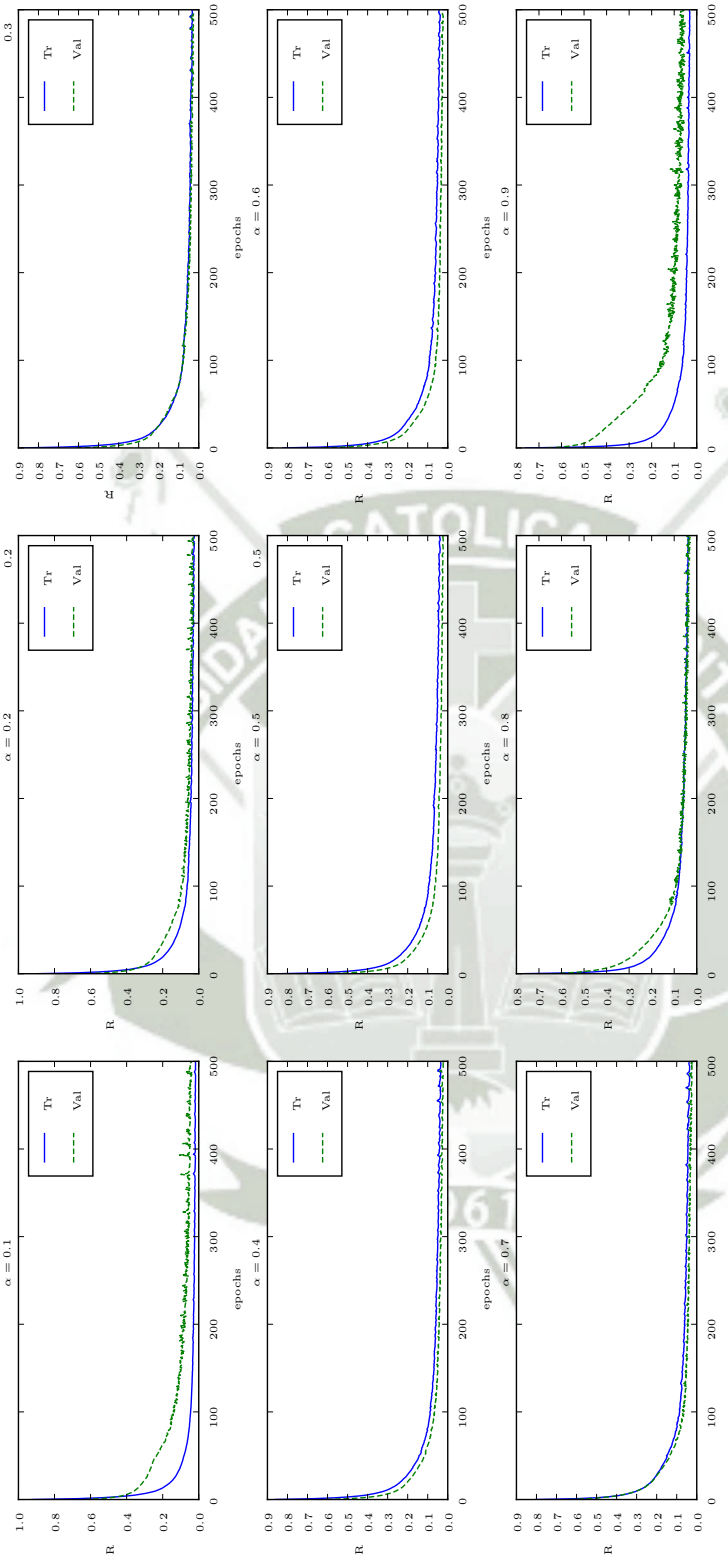

Figure 18: Evolución de la función de coste en validación (conjunto de validación fold 5) utilizando dropout $d = 0.25$ para distintos valores de $\alpha \in [0.10.9]$.

5.4 Resultados

Una vez finalizada la etapa de validación, ya podemos entrenar las máquinas con un adecuado conjunto de hiperparámetros. Los resultados en media sobre 10 simulaciones tanto para la red neuronal como para la red profunda se presentan en la Tabla 8 en términos de AUC tanto para el conjunto de entrenamiento, validación así como el de test. Para esto, se ha separado en cada simulación un 10% de muestras de forma aleatoria de tal manera que este pueda ser utilizado como cojunto de validación y así utilizar el algoritmo de Early Stopping como criterio de parada.

La Tabla 8 muestra que a pesar de que los resultados para entrenamiento y validación de la NN estandar (una sóla capa oculta y BackPropagation), los resultados en test son inferiores a los obtenidos por la DNN con dropout como técnica de regularización, obteniendo esta última red un 2.58% más de AUC. Es plausible indicar que debido a que la NN estándar obtiene un mejor desenvolvimiento en el conjunto de entrenamiento e inferior en test, la NN estandard sobreajusta sobre este conjunto, *i.e.* sobre-aprende el conjunto de entrenamiento llegando incluso a aprender ruido dentro de este conjunto, por ello decimos que la DNN generaliza de mejor manera sobre todo el problema.

			AUC(%)		
N_l	N_h	d	Train	Val.	Test
1	50	-	99.76	99.74	84.00
2	50	0.25	98.17	98.11	86.58

Table 8: Resultados sobre el conjunto de entrenamiento validación y test utilizando los parámetros seleccionados en la Sección 5.2.

6 Conclusiones

En este trabajo de Tesis se ha atacado el problema de la detección de ataques en redes informáticas utilizando métodos propios del área de la Aprendizaje Máquina. Para ello, primero se ha efectuado una revisión de los elementos que componen los sistemas de prevensión y detección de posibles ataques, centrándo el punto de interés sobre todo en los IDSs; y entre estos, se ha enfocado el estudio en aquellos que utilizan una base de datos con tráfico normal y ataques conocidos para poder clasificar nuevos registros o muestras en ataques o no. A diferencia de IDSs clásicos que basan su funcionamiento en reglas eurístias, en este trabajo se ha utilizado redes neuronales tanto estándares como profundas. Con tal fin, se ha procedido a realizar una codificación apropiada, de las características en la base de datos KDD DARPA 1999, y así transformar las características en otras más apropiadas para entrenar las NNs. Esto posibilita el dimensionamiento de las redes neuronales y se procedió a diseñar los clasificadores validando los hiper-parámetros utilizando la técnica de validación cruzada (número de neuronas ocultas en el caso de las NNs estándares además de el porcentaje de dropout y el número de capas ocultas para el caso de las DNNs).

Además, se ha presentado un método para evaluar los clasificadores utilizando la AUC formada al variar un parámetro de la función de coste modificada para sopesar de forma distinta el coste –al estimar erroneamente los valores de las etiquetas– de una clase y los costes de la otra clase . Esto permite entrenar los clasificadores en distintos puntos de la ROC, que a diferencia de los métodos tradicionales que varían el umbral de decisión a la salida del clasificador, trae ventajas adicionales ya que es posible utilizar clasificadores que trabajen en distintos puntos de la ROC y así seleccionar la P_D y P_{FA} acorde a los límites impuestos por el equipo técnico que revisará las muestras clasificadas como ataques.

Una vez se ha seleccionado los hiper-parámetros, se ha procedido a entrenar clasificadores y medir su *performance* también en términos de AUC. Un análisis de los resultados muestra que se obtienen ventajas cuando se utiliza casificadores basados en máquinas profundas. Más concretamente, las DNNs obtienen un 2.58% de margen de ganancia con respecto a las NNs estándares, en media.

Así, este resultado permite afirmar que es posible tener venta-

jas utilizando algoritmos propios del Deep Learning y el IDS resultante es una herramienta más dentro de las muchas que los equipos técnicos pueden utiizar para detectar posibles intrusiones a las redes informáticas.

Además, los resultados en validación y test parecen indicar que los datos en ambos conjuntos provienen de distribuciones distintas. Esto es, las curvas de la evolución de los costes en entrenamiento y validación son similares pero el performance en AUC del conjunto de test es muy inferior en comparación a dichos conjuntos. Esto es entendible si se toma en cuenta que los ataques en test son posiblemente distintos a los que se encuentran en el conjunto de entrenamiento (una evolución o mutación del ataque, o uno nuevo). Esto hace notar la dificultad de obtener IDSs basados en algoritmos de Aprendizaje Máquina y refuerza el objetivo de seguir investigando cómo se desempeñan estás máquinas en estos entornos.

6.1 Trabajos Futuros

Un paso siguiente es el de entrenar clasificadores que se especialicen en cada tipo de ataque, esto es, entrenar los clasificadores bajo el enfoque uno contra todos. Esta nueva arquitectura de IDS será más costosa pero es un complemento a tomar en cuenta. Será necesario analizar si la ganancia en AUC justifica el coste computacional en el que se incurre para entrenar las máquinas que formarán esta nueva arquitectura.

El uso del dropout como técnica de regularización abre la puerta a la utilización de otros algoritmos que mejoren el aprendizaje sobre la base de datos utilizada en esta Tesis. Podemos indicar que el siguiente paso es analizar el problema de *Shift Covariance* en las capas ocultas de la red (caraterísticas reales no están normalizadas en media y/o varianza en las capas ocultas).

Estadísticamente, el hecho de que las muestras de entrenamiento y test tengan comportamientos diferentes se debe a que pertenecen a distribuciones diferentes, por ejemplo podría ser el caso en que la media y varianza es distinta en cada caso. Este problema es conocido como *Concept Drift Learning* y por ello amerita extender el estudio utilizando técnicas propias de este campo de estudio.

Bibliografía

[1] J. Menn, J. Finkle, and D. Volz, "Cyber attacks disrupt Paypal, Twitter, other sites," Disponible en: http://www.reuters.com/article/us-usa-cyber-idUSKCN12L1ME, Revisado: 2016-12-19.

[2] M. Ligh, S. Adair, B. Hartstein, and M. Richard, *Tools and Techniques for Fighting Malicious Code*. Wiley India Pvt. Limited, 2010.

[3] J. Cox, "Yahoo 'aware' hacker is advertising 200 million supposed accounts on dark web," Disponible en: http://motherboard.vice.com/read/yahoo-supposed-data-breach-200-million-credentials-dark-web, Revisado: 2016-12-19.

[4] C. McGoogan, "Cyber attacks disrupt Paypal, Twitter, other sites," Disponible en: http://www.telegraph.co.uk/technology/2016/08/31/dropbox-hackers-stole-70-million-passwords-and-email-addresses, Revisado: 2016-12-19.

[5] J. Mendiola-Zuriarrain, "Spotify investiga la posible infección con 'malware' de su versión gratuita," Disponible en: http://tecnologia.elpais.com/tecnologia/2016/10/05/actualidad/1475687384_392685.html, Revisado: 2016-12-19.

[6] LeakedSource, "LeakedSource," Disponible en: https://www.leakedsource.com, Revisado: 2016-12-19.

[7] Diario Excélsior. Hacker roba 360 millones de cuentas de myspace. Disponible en: http://www.excelsior.com.mx/hacker/2016/06/01/1096191. Revisado: 2017-03-12.

[8] "ABC Tecnología y Redes. Hackean MySpace y ponen a la venta 360 millones de cuentas," Disponible en: http://www.abc.es/tecnologia/redes/abci-hackean-myspace-y-ponen-venta-360-millones-cuentas-201606012050_noticia.html, Revisado: 2017-02-19.

[9] D. Sarabia. 100 millones de cuentas LinkedIn hackeadas por 2.000 euros. Revisado: 2017-03-12. [Online]. Available: http://www.eldiario.es/cultura/tecnologia/privacidad/millones-cuentas-LinkedIn-hackeadas-euros_0_517598934.html

[10] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern classification*. John Wiley & Sons, 2012.

[11] C. M. Bishop, *Neural networks for pattern recognition*. Oxford university press, 1995.

[12] A. Sperotto, G. Schaffrath, R. Sadre, C. Morariu, A. Pras, and B. Stiller, "An overview of IP flow-based intrusion detection," *IEEE Communications Surveys & Tutorials*, vol. 12, no. 3, pp. 343–356, 2010.

[13] M. Roesch *et al.*, "Snort: Lightweight intrusion detection for networks." in *LISA*, vol. 99, no. 1, 1999, pp. 229–238.

[14] Cisco-Systems, "Security Configuration Guide: Cisco IOS Intrusion Prevention System Cisco IOS Release 15.1M&T," Disponible en: http://www.cisco.com/c/en/us/td/docs/ios/sec_data_plane/configuration/guide/convert/sec_data_ios_ips_15_1_book.html, Revisado: 2016-12-19.

[15] "Tripwire," https://www.tripwire.com, Revisado: 2016-12-19.

[16] R. Kozik and M. Choraś, "Machine learning techniques for cyber attacks detection," in *Image Processing and Communications Challenges 5*. Springer, 2014, pp. 391–398.

[17] Y. Bengio, "Learning deep architectures for AI," *Foundations and Trends in Machine Learning*, vol. 2, no. 1, pp. 1–127, 2009.

[18] Y. Bengio, A. C. Courville, and P. Vincent, "Unsupervised feature learning and deep learning: A review and new perspectives," *CoRR, abs/1206.5538*, vol. 1, 2012.

[19] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[20] D. E. Denning, "An intrusion-detection model," *IEEE Transactions on software engineering*, no. 2, pp. 222–232, 1987.

[21] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural networks*, vol. 2, no. 5, pp. 359–366, 1989.

[22] K. Tan, "The application of neural networks to UNIX computer security," in *Procs., IEEE Intl. Conf. on Neural Networks*, vol. 1. IEEE, 1995, pp. 476–481.

[23] J. Ryan, M.-J. Lin, and R. Miikkulainen, "Intrusion detection with neural networks," *Advances in neural information processing systems*, pp. 943–949, 1998.

[24] J. Cannady, "Artificial neural networks for misuse detection," in *Procs. National Information Systems Security Conf.*, 1998, pp. 368–81.

[25] R. P. Lippmann and R. K. Cunningham, "Improving intrusion detection performance using keyword selection and neural networks," *Computer Networks*, vol. 34, no. 4, pp. 597–603, 2000.

[26] A. Bivens, C. Palagiri, R. Smith, B. Szymanski, M. Embrechts *et al.*, "Network-based intrusion detection using neural networks," *Intelligent Engineering Systems through Artificial Neural Networks*, vol. 12, no. 1, pp. 579–584, 2002.

[27] M. V. Mahoney and P. K. Chan, "An analysis of the 1999 DARPA/Lincoln laboratory evaluation data for network anomaly detection," in *International Workshop on Recent Advances in Intrusion Detection*. Springer, 2003, pp. 220–237.

[28] R. Lippmann, J. W. Haines, D. J. Fried, J. Korba, and K. Das, "The 1999 DARPA off-line intrusion detection evaluation," *Computer networks*, vol. 34, no. 4, pp. 579–595, 2000.

[29] S. X. Wu and W. Banzhaf, "The use of computational intelligence in intrusion detection systems: A review," *Applied Soft Computing*, vol. 10, no. 1, pp. 1–35, 2010.

[30] A. L. Buczak and E. Guven, "A survey of data mining and machine learning methods for cyber security intrusion detection," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 2, pp. 1153–1176, 2015.

[31] D. Heckerman, "A tutorial on learning with Bayesian networks," in *Learning in graphical models*. Springer, 1998, pp. 301–354.

[32] T. D. Nielsen and F. V. Jensen, *Bayesian networks and decision graphs*. Springer Science & Business Media, 2009.

[33] F. Jemili, M. Zaghdoud, and M. B. Ahmed, "A framework for an Adaptive Intrusion Detection system using Bayesian network." in *ISI*, 2007, pp. 66–70.

[34] S. Benferhat, T. Kenaza, and A. Mokhtari, "A naive Bayes approach for detecting coordinated attacks," in *2008 32nd Annual IEEE International Computer Software and Applications Conference*. IEEE, 2008, pp. 704–709.

[35] J. R. Quinlan, "Induction of decision trees," *Machine learning*, vol. 1, no. 1, pp. 81–106, 1986.

[36] ——, *C4. 5: programs for machine learning*. Elsevier, 2014.

[37] C. Kruegel and T. Toth, "Using decision trees to improve signature-based intrusion detection," in *International Workshop on Recent Advances in Intrusion Detection*. Springer, 2003, pp. 173–191.

[38] L. Bilge, E. Kirda, C. Kruegel, and M. Balduzzi, "EXPOSURE: Finding malicious domains using passive DNS analysis." in *NDSS*, 2011.

[39] L. Bilge, S. Sen, D. Balzarotti, E. Kirda, and C. Kruegel, "Exposure: a passive dns analysis service to detect and report malicious domains," *ACM Transactions on Information and System Security (TISSEC)*, vol. 16, no. 4, p. 14, 2014.

[40] V. Vapnik, *The nature of statistical learning theory*. Springer Science & Business Media, 2013.

[41] Y. Li, J. Xia, S. Zhang, J. Yan, X. Ai, and K. Dai, "An efficient intrusion detection system based on support vector machines and gradually feature removal method," *Expert Systems with Applications*, vol. 39, no. 1, pp. 424–430, 2012.

[42] F. Amiri, M. R. Yousefi, C. Lucas, A. Shakery, and N. Yazdani, "Mutual information-based feature selection for intrusion detection systems," *Journal of Network and Computer Applications*, vol. 34, no. 4, pp. 1184–1199, 2011.

[43] W. Hu, Y. Liao, and V. R. Vemuri, "Robust support vector machines for anomaly detection in computer security." in *ICMLA*, 2003, pp. 168–174.

[44] C. Wagner, J. François, T. Engel *et al.*, "Machine learning approach for IP-flow record anomaly detection," in *International Conference on Research in Networking*. Springer, 2011, pp. 28–39.

[45] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin, "Liblinear: A library for large linear classification," *Journal of machine learning research*, vol. 9, no. Aug, pp. 1871–1874, 2008.

[46] J. H. Friedman, "Multivariate adaptive regression splines," *The annals of statistics*, pp. 1–67, 1991.

[47] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[48] S. Mukkamala, A. H. Sung, and A. Abraham, "Intrusion detection using an ensemble of intelligent paradigms," *Journal of network and computer applications*, vol. 28, no. 2, pp. 167–182, 2005.

[49] F. Gharibian and A. A. Ghorbani, "Comparative study of supervised machine learning techniques for intrusion detection," in *Fifth Annual Conference on Communication Networks and Services Research (CNSR'07)*. IEEE, 2007, pp. 350–358.

[50] P. Long and R. Servedio, "Boosting the area under the ROC curve," in *Advances in neural information processing systems*, 2007, pp. 945–952.

[51] J. Zhang, M. Zulkernine, and A. Haque, "Random-forests-based network intrusion detection systems," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 38, no. 5, pp. 649–659, 2008.

[52] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural computation*, vol. 18, no. 7, pp. 1527–1554, 2006.

[53] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders," in *Proceedings of the 25th international conference on Machine learning*. ACM, 2008, pp. 1096–1103.

[54] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting." *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[55] *MATLAB*, The Mathworks, Inc., Natick, Massachusetts, 2015.

[56] G. Van Rossum and F. L. Drake, "Python tutorial, release 2.2. 1," 2002.

[57] R. C. Team *et al.*, "R: A language and environment for statistical computing," 2013.

[58] M. L. Labs., "DARPA intrusion detection evaluation," Disponible en: http://www.ll.mit.edu/mission/communications/ist/corpora/ideval/index.html, Revisado: 2016-12-19.

[59] "KDD Cup 1999," Disponible en: http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html, Revisado: 2016-12-19.

[60] J. McHugh, "Testing intrusion detection systems: a critique of the 1998 and 1999 DARPA intrusion detection system evaluations as performed by Lincoln laboratory," *ACM Transactions on Information and System Security (TISSEC)*, vol. 3, no. 4, pp. 262–294, 2000.

[61] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[62] D. O. Hebb, *The organization of behavior: A neuropsychological theory*. Psychology Press, 2005.

[63] B. Widrow *et al.*, "Adaptive "adaline" neuron using chemical memistors," Standford University, Tech. Rep., 1960.

[64] C. R. Winter and B. Widrow, "Madaline rule ii: a training algorithm for neural networks," in *Secondf Annual International Conference on Neural Networks*, 1988, pp. 1–401.

[65] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain." *Psychological review*, vol. 65, no. 6, p. 386, 1958.

[66] M. Minsky and S. Papert, "Perceptrons: An introduction to computational geometry." 1969.

[67] J. Anderson, "Neural models with cognitive implications. basic processes in reading perception and comprehension models," 1977.

[68] D. E. Rumelhart, J. L. McClelland, P. R. Group *et al.*, *Parallel distributed processing*. IEEE, 1988, vol. 1.

[69] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proceedings of the national academy of sciences*, vol. 79, no. 8, pp. 2554–2558, 1982.

[70] H. Ritter and T. Kohonen, "Self-organizing semantic maps," *Biological cybernetics*, vol. 61, no. 4, pp. 241–254, 1989.

[71] G. E. Hinton and T. J. Sejnowski, "Learning and releaming in boltzmann machines," *Parallel Distrilmted Processing*, vol. 1, 1986.

[72] S. Grossberg, "Competitive learning: From interactive activation to adaptive resonance," *Cognitive science*, vol. 11, no. 1, pp. 23–63, 1987.

[73] Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, "Efficient BackProp," in *Neural networks: Tricks of the trade*. Springer, 2012, pp. 9–48.

[74] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85–117, 2015.

Apéndices

A. Normalización

El procedimiento de normalización consiste en escalar los datos reales a una distribución con media cero y varianza unidad. Para ello:

- Se estima la media μ y desviación estándar σ muestral de la característica en cuestión utilizando solo los datos de entrenamiento.
- Luego se procede a escalar los datos utilizando la Ec. 38.
- Para escalar las características de los conjuntos de validación y test se utiliza las medias y desviación estándar muestral del conjunto de entrenamiento y la Ec. 38.

$$x_{\mu=0,\sigma^2=1} = \frac{x - \mu}{\sigma} \tag{38}$$

También es posible utilizar la siguiente Ec. 39 para escalar las amplitudes.

$$x_{\mu=0} = \frac{x - \mu}{x_{max} - x_{min}} \tag{39}$$

donde x_{max} y x_{min} representa al valor máximo y mínimo de x.

B. Regresor Lineal

En cuanto al regresor lineal

$$y = \mathbf{W}^T \mathbf{X} \tag{40}$$

donde $\mathbf{W}$ son los parámetros del modelo y $\mathbf{X}$ es la matriz de diseño cuyas columnas representan a los datos de entrenamiento. Es posible optimizar los pesos $\mathbf{W}$ de las siguientes manera:

- Solución exacta: cuando el número de dimensiones o características de $\mathbf{x}$ es $D < 10000$:

$$\mathbf{W} = \mathbf{X}_{tr}^{\mp} \mathbf{y}_{tr} \tag{41}$$

  donde $(\cdot)^{\mp}$ representa la pseudo inversa de Moore-Penrose.

- Descenso por gradiente: cuando el número de dimensiones o características de $\mathbf{x}$ es $D > 10000$

$$(\Theta)^{(\tau+1)} \leftarrow (\Theta)^{(\tau)} - \eta \frac{\partial \mathcal{J}(\Theta)}{\partial \Theta} \tag{42}$$

donde $\Theta = \mathbf{W}$ representa los parámetros del modelo, η representa la tasa de aprendizaje, y $\mathcal{J}(\Theta)$ es la función de coste a minimizar.

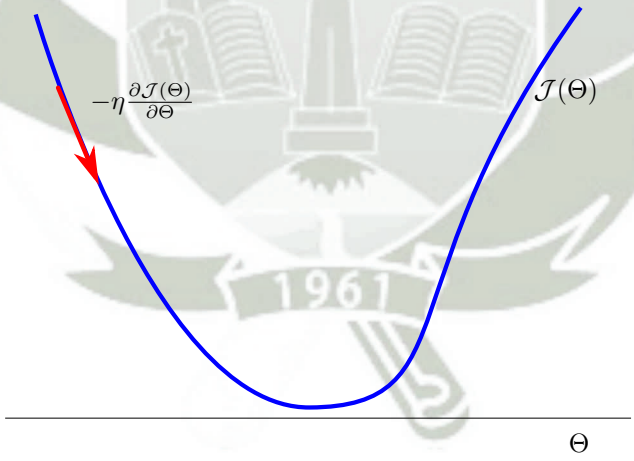

Figure 19: Descenso por gradiente para optimizar los parámetros $\Theta = \mathbf{W}$ que minimicen la función de coste $\mathcal{J}(\Theta)$.

C. Tutorial y listado de los códigos implementados en Python

El siguiente tutorial presenta el procedimiento utilizado para llevar a cabo el presente trabajo de Tesis. Además, detalla la función de cada programa implementado.

Diagrama de bloques

En la Fig. 20 muestra el diagrama de bloques de manera general los pasos tomados en el trabajo de Tesis. En general, se procede a:

1. Cargar datos de entrenamiento y encontrar la media y varianza de las variables reales.
2. Se procede a normalizar los datos para que estas tengan media nula y varianza unidad.
3. Se codifica las variabes categóricas utilizando la técnica de "one-hot encodigng"
4. Se procede a entrenar distintos clasificadores con diferentes combinaciones de hiperparámetros (número de capas ocultas, número de neuronas ocultas, etc) según sea el caso (red neuronal estándar or regularizada con dropout).
5. Una vez se han escogido una buena combinación de hiperparámetros, se procede a entrenar el clasificador final.
6. Por último, se estima las etiquetas utilizando el clasificador final sobre el conjunto de datos de prueba o test.

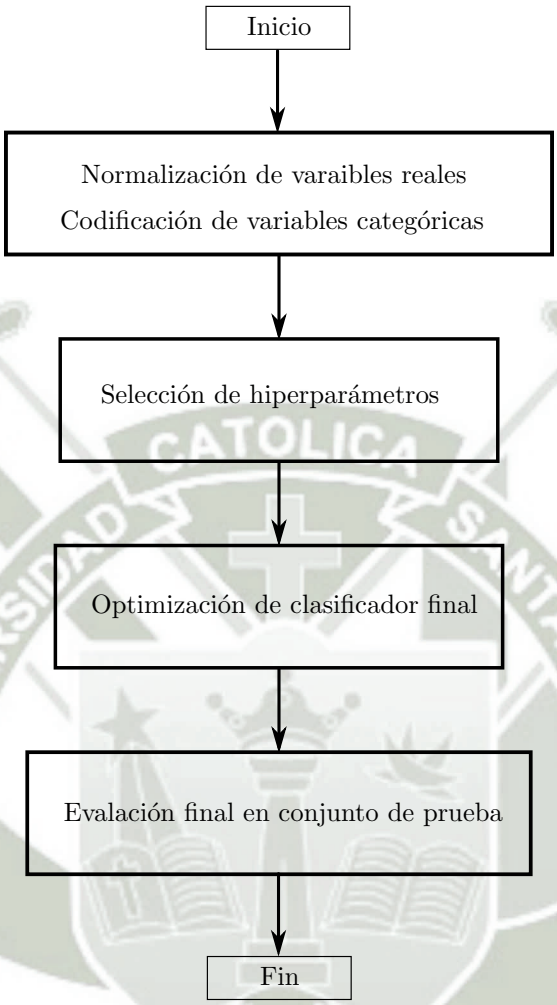

Figure 20: Diagram de bloques de los pasos tomados en cuenta para entrenar redes neuronales, tanto estándares como regularizadas con dropout.

Programas en python

Para realizar las tareas, se han escrito los siguientes archivos:

1. `main_class_weigths_mlp_cros_entrop_validacion.py`: Este archivo procede a realizar la validación del número de neuronas en la capa oculta de una red neuronal simple.

2. `main_class_weigths_mlp_cros_entrop_clasificador_final_NN.py`: Este archivo entrena una red neuronal standand utilizando el número de neuronas ocultas validadas en la etapa anterior. También realiza la clasificación en el conjunto de prueba para saber el performance de la red neuronal estándar final.

3. `main_class_weigths_dnn_cros_entrop_validacion.py`: Este archivo implementa los pasos para validar los hiperparámetros de las máquinas profundas (número de capas ocultas y porcentaje de dropout).

4. `main_class_weigths_dnn_cros_entrop_clasificador_final_NN.py`: Entrena el clasificador final utilizando los hiperparámetros encontrados en la Sección 2.3. También realiza la estimación de etiquetas del conjunto de prueba para saber el performance de estas redes.

1. Archivo "main_class_weigths_mlp_cros_entrop_validacion.py"

```
# cargar librerias

from __future__ import absolute_import

from __future__ import print_function
#forma de displayar en pantalla

from __future__ import division #division
con decimales

import numpy as np #manejo de matrices

from scipy.io import loadmat #libreria
cientifica, abrir archivos.mat

from keras.models import Sequential #
haer el modelo en keras y el sequential
empezar a realizar el modelo

from keras.layers.core import Dense,
Activation #, Dropout, Dense conecion de
neuronas una a una, activacion tipo de
funcion a activar

from keras.optimizers import RMSprop
#SGD #Adam # SGD #, #RMSprop desenso
por gradiente, forma automatica tasa de
aprendizaje

from keras.callbacks import
EarlyStopping #para cuando converge el
conj de validacion

import matplotlib.pyplot as plt #de la
libreria matplot importa pyplot como plt

from functools import partial #aepte
variables que no estan en el programa

np.random.seed(1337) # for reproducibility

#%% Funciones propias en extrafunctions y
bayesian_cost_functions

from extrafunctions import
calcula_tabla,calcula_auc_ps

#from extrafunctions import
preprocess_data

#%%

nb_classes = 2 #numero de clases

#%%

NN = 50 #numero de neuronas en la capa
oculta NNs = np.array([10,20,30,50,70,90])

nb_epoch = 500


#%% Cargar Datos

Kfolds = np.arange(5)

auc_5fold=np.zeros((1,5))

auc_5fold_tr=np.zeros((1,5))

alphas = np.linspace(0.1, 0.9, 9)

idx_05=alphas==0.5

n_alphas = len(alphas)

PFA_kfolds_val = np.zeros((n_alphas + 2,5))

PFA_kfolds_val[0,:] = 1;

PD_kfolds_val = np.zeros((n_alphas + 2,5))

PD_kfolds_val[0,:] = 1;

PFA_kfolds_tr = np.zeros((n_alphas + 2,5))

PFA_kfolds_tr[0,:] = 1;

PD_kfolds_tr = np.zeros((n_alphas + 2,5))

PD_kfolds_tr[0,:] = 1;


#%%

acc = np.zeros((n_alphas,1))

costes_matrix_tr = np.zeros((n_alphas,5))

costes_matrix = np.zeros((n_alphas,5))

PFA = np.zeros((n_alphas + 2,1))

PFA[0] = 1;

PD = np.zeros((n_alphas + 2,1))

PD[0] = 1;

PFA_tr = np.zeros((n_alphas + 2,1))

PFA_tr[0] = 1;

PD_tr = np.zeros((n_alphas + 2,1))

PD_tr[0] = 1;

#%%
```

```
for ik in Kfolds:

    k = ik + 1

    print('Normalized vals. Num. de Fold =
', k)

    Datos_all =
loadmat('D:\TesisMel\datos\KDD_norm_tr_fol
d_' + str(k) + '.mat')



    #%% Formato adecuado "float32"

    X = Datos_all['x_tr'].astype("float32")

    Y = Datos_all['y_tr'].astype("float32")



    Xtst =
Datos_all['x_val'].astype("float32")

    Ytst =
Datos_all['y_val'].astype("float32")



    D = X.shape[1]

    N = len(X)

    Ntst = len(Xtst)



    #%%

    #print('All samples: ', N)

    #print('Test samples: ', Ntst)

    N_pos = len(Y[Y==+1])

    N_neg = len(Y[Y==-1])

    Ntst_pos = len(Ytst[Ytst==+1])

    Ntst_neg = len(Ytst[Ytst==-1])

    IR      = N_neg/N_pos

    IR_tst = Ntst_neg/Ntst_pos

    #print('N_pos: ', N_pos,' - N_neg:
',N_neg, ' - IR : ', IR)

    #print('Ntst_pos: ', Ntst_pos,' -
Ntst_neg: ', Ntst_neg, ' - IR_tst: ', IR_tst)



    #%% normalization

    #X, scaler      = preprocess_data(X)

    #Xtst,scaler    =
preprocess_data(Xtst,scaler)

    #%% Parametros Bayes



    #%%

    f = k;

    #acc = np.zeros((f,1))
#array para guardar el resultado en
validacion

    print('---'*10,'Fold ',f,'---'*10)
#para simular separacion en pantalla

    xtr  = X

    ytr  = Y

    xval= Xtst

    yval= Ytst

    Ntr_pos  = len(ytr[ytr==+1])

    Ntr_neg  = len(ytr[ytr==-1])

    Nval_pos = len(yval[yval==+1])

    Nval_neg = len(yval[yval==-1])

    Ntr  = len(xtr)

    P0 =  Ntr_neg/Ntr    #probabilidad de
la clase 0

    P1 =  Ntr_pos/Ntr    #probabilidad de
la clase 1

    #%%

    alphas_idxs =  np.arange(0,n_alphas)

    optzer  = RMSprop()
```

```
    earlyStopping =
EarlyStopping(monitor='val_loss',
patience=10, verbose=0, mode='min')


    t_tr = ytr > 0.5

    y_tr = t_tr.astype('float32')

    t_val = yval > 0.5

    y_val = t_val.astype('float32')

    t_tst = Ytst > 0.5

    y_tst = t_tst.astype('float32')




    for ialpha in alphas_idxs:

        np.random.seed(1337)  # for
reproducibility

        alpha_num = alphas[ialpha];

        print('alpha = ', alpha_num)

        C10 = alpha_num/P0
#penalizacion cuando se falla en la clase
"0"

        C01 = (1 - alpha_num)/P1
#penalizacion cuando se falla en la clase
"1"

        class_weight = {0:C10, 1:C01}
#[C01,C10]

        #print(class_weight)

        model = Sequential()

        model.add(Dense(output_dim = NN
, input_dim = D))

        model.add(Activation('tanh'))

        model.add(Dense(output_dim = 1 ,
input_dim = NN))

        model.add(Activation('sigmoid'))

        model.compile(loss =
'binary_crossentropy', optimizer = optzer)

        coste = model.fit(xtr, y_tr,
batch_size=Ntr, nb_epoch=nb_epoch,
verbose=0, callbacks=[earlyStopping],
validation_data=(xval, y_val), class_weight =
class_weight)

        #score = model.evaluate( xval,
y_val, show_accuracy = True, verbose = 0
)

        score = model.evaluate( xval,
y_val, verbose = 0 )

        print('Test score en fold     ' , k
,' = ', score)

        acc[ialpha] = score

        print('---fin fold---')

        #%%

        coste1 =  coste.history

        plt.figure(10+k)

        plt.subplot(3,3,ialpha+1)

        plt.plot(coste1['loss'])       # a
diferencia del matlab no es necesario
poner hold on para sobreponer otra
grafica

        plt.plot(coste1['val_loss'],'--')  # a
diferencia del matlab no es necesario
poner hold on para sobreponer otra
grafica

        plt.xlabel('epochs')

        plt.ylabel('H')

        plt.legend(('Tr','Val'))

        plt.title( r'$\alpha $')

        plt.title( alpha_num, loc='right')

        #%%

        ye_tr = model.predict(xtr,
batch_size=Ntr)

        te_tr = ye_tr > 0.5

        Ye_tr = 2.0*(te_tr.astype('float32'))
- 1.0

        w00, w10, w01, w11 =
calcula_tabla(ytr,Ye_tr)

        costes_matrix_tr[ialpha,:] =
np.array([w00, w10, w01, w11,
alpha_num*w10 + (1. - alpha_num)* w01])
```

```
        ye_tst = model.predict(Xtst,
batch_size=Ntst)

        te_tst = ye_tst > 0.5

        Ye_tst =
2.0*(te_tst.astype('float32')) - 1.0

        w00, w10, w01, w11 =
calcula_tabla(Ytst,Ye_tst)

        costes_matrix[ialpha,:] =
np.array([w00, w10, w01, w11,
alpha_num*w10 + (1. - alpha_num)* w01])

    #%%


    PFA[1:n_alphas+1] =
np.reshape(costes_matrix[:,1],(n_alphas,1))

    PD[1:n_alphas+1]  =
np.reshape(costes_matrix[:,3],(n_alphas,1))

    auc,ps = calcula_auc_ps(PFA,PD,idx_05)

    auc_5fold[0,ik]=auc


    PFA_tr[1:n_alphas+1] =
np.reshape(costes_matrix_tr[:,1],(n_alphas,1
))

    PD_tr[1:n_alphas+1]  =
np.reshape(costes_matrix_tr[:,3],(n_alphas,1
))

    auc_tr,ps_tr =
calcula_auc_ps(PFA_tr,PD_tr,idx_05)

    auc_5fold_tr[0,ik]=auc_tr


    PFA_kfolds_val[:,ik] = PFA[:,0];

    PD_kfolds_val[:,ik] = PD[:,0];

    PFA_kfolds_tr[:,ik] = PFA_tr[:,0];

    PD_kfolds_tr[:,ik] = PD_tr[:,0];


    #%%

    plt.figure(k)

    plt.plot(PFA_tr,PD_tr)

    plt.plot(PFA,PD,'--')        # a
diferencia del matlab no es necesario
poner hold on para sobreponer otra
grafica

    plt.legend(('Tr.','Val.'))

    plt.xlabel('PFA')

    plt.ylabel('PD')

    #%%

auc_sim_tr = np.mean(auc_5fold_tr[0])

print('Normalized vals.')

print('auc tr. para los 5 folds: ',
auc_5fold_tr)

print('auc tr. media de los 5 folds: ',
auc_sim_tr, 'para NN = ' , NN)


auc_sim = np.mean(auc_5fold[0])

print('auc val. para los 5 folds: ',
auc_5fold)

print('auc val. media de los 5 folds: ',
auc_sim, 'para NN = ' , NN)


#%%

file_name = 'resultados_mats/NN_' +
str(NN) + '_epochs_'+ str(nb_epoch)+
'_validacion_norm.npz'

np.savez(file_name, auc_sim_tr =
auc_sim_tr, auc_sim = auc_sim,
auc_5fold_tr=auc_5fold_tr,
auc_5fold=auc_5fold,
         PFA_kfolds_val= PFA_kfolds_val,
         PD_kfolds_val = PD_kfolds_val,
         PFA_kfolds_tr = PFA_kfolds_tr,
         PD_kfolds_tr  = PD_kfolds_tr)


#%%

PFA_mean_kfolds_tr =
np.mean(PFA_kfolds_tr,axis=1)

PFA_std_kfolds_tr =
np.std(PFA_kfolds_tr,axis=1)
```

```
PD_mean_kfolds_tr  =
np.mean(PD_kfolds_tr,axis=1)

PD_std_kfolds_tr  =
np.std(PD_kfolds_tr,axis=1)



PFA_mean_kfolds_val  =
np.mean(PFA_kfolds_val,axis=1)

PFA_std_kfolds_val  =
np.std(PFA_kfolds_val,axis=1)



PD_mean_kfolds_val  =
np.mean(PD_kfolds_val,axis=1)

PD_std_kfolds_val  =
np.std(PD_kfolds_val,axis=1)



plt.figure(200)

plt.plot(PFA_mean_kfolds_tr,
PD_mean_kfolds_tr, 'b')

plt.plot(PFA_mean_kfolds_val,
PD_mean_kfolds_val, '--g')

plt.legend(('Tr.','Val.'))

plt.xlabel('$P_{FA}$')

plt.ylabel('$P_D$')



plt.figure(201)

plt.errorbar(PFA_mean_kfolds_tr,
PD_mean_kfolds_tr,
yerr=[PD_std_kfolds_tr, PD_std_kfolds_tr],
xerr=[PFA_std_kfolds_tr,
PFA_std_kfolds_tr], fmt='b')

plt.errorbar(PFA_mean_kfolds_val,
PD_mean_kfolds_val,
yerr=[PD_std_kfolds_val,
PD_std_kfolds_val],
xerr=[PFA_std_kfolds_val,
PFA_std_kfolds_val], fmt='--g')

plt.legend(('Tr.','Val.'))

plt.xlabel('$P_{FA}$')

plt.ylabel('$P_D$')



plt.figure(202)

plt.errorbar(PFA_mean_kfolds_tr,
PD_mean_kfolds_tr,
yerr=[PD_std_kfolds_tr, PD_std_kfolds_tr],
xerr=[PFA_std_kfolds_tr,
PFA_std_kfolds_tr], fmt='b')

plt.errorbar(PFA_mean_kfolds_val,
PD_mean_kfolds_val,
yerr=[PD_std_kfolds_val,
PD_std_kfolds_val],
xerr=[PFA_std_kfolds_val,
PFA_std_kfolds_val], fmt='--g')

plt.legend(('Tr.','Val.'))

plt.xlabel('$P_{FA}$')

plt.ylabel('$P_D$')

plt.axis([0, 0.05, 0.985, 1]) #îzoom



'''

NNs = np.array([10,20,30,50,70,90])

auc_means_tr = np.array([98.6406,99.6384,
99.6454, 99.7176, 99.7127, 99.7368 ])

auc_means_tst = np.array([98.6406,99.6384,
99.6454, 99.7176, 99.7127, 99.7368 ])



plt.figure(101)

plt.plot(NNs,auc_means_tr,'b')

plt.plot(NNs,auc_means_tr,'-r')

plt.xlabel('NN'), plt.ylabel('auc'),
#plt.axis([np.min(NNs), np.max(NNs), 98,
100])
```

2. Archivo "main_class_weigths_mlp_cros_entrop_clasificador_final_NN,py"

```
# cargar librerias

from __future__ import absolute_import

from __future__ import print_function
#forma de displayar en pantalla

from __future__ import division #division
con decimales

import numpy as np #manejo de matrices

from scipy.io import loadmat #libreria
cientifica, abrir archivos.mat

from keras.models import Sequential #
haer el modelo en keras y el sequential
empezar a realizar el modelo

from keras.layers.core import Dense,
Activation #, Dropout, Dense conecion de
neuronas una a una, activacion tipo de
funcion a activar

from keras.optimizers  import RMSprop
#SGD #Adam # SGD #, #RMSprop desenso
por gradiente, forma automatica tasa de
aprendizaje

from keras.callbacks   import
EarlyStopping #para cuando converge el
conj de validacion

import matplotlib.pyplot as plt  #de la
libreria matplot importa pyplot como plt

#from functools import partial #aepte
variables que no estan en el programa


#%% Funciones propias en extrafunctions y
bayesian_cost_functions

from extrafunctions import
calcula_tabla,calcula_auc_ps

#from extrafunctions import
preprocess_data

#%%

nb_classes = 2  #numero de clases

#%%

NN = 50 #numero de neuronas en la capa
oculta

#d   = 0.5 #porcentaje de dropout


#%% Cargar Datos

Datos_all =
loadmat('D:\TesisMel\datos\KDD_norm_tr_fin
al' + '.mat')

#%% Formato adecuado "float32"

xtr  = Datos_all['x_tr'].astype("float32")

ytr  = Datos_all['y_tr'].astype("float32")

xval = Datos_all['x_val'].astype("float32")

yval = Datos_all['y_val'].astype("float32")

Datos_all =
loadmat('D:\TesisMel\datos\KDD_norm_tst_fi
nal' + '.mat')

xtst = Datos_all['x_tst'].astype("float32")

ytst = Datos_all['y_tst'].astype("float32")


D = xtr.shape[1]

N = len(xtr)

Nval = len(xval)

Ntst = len(xtst)


#%%

#print('All samples: ', N)

#print('Test samples: ', Ntst)

N_pos = len(ytr[ytr==+1])

N_neg = len(ytr[ytr==-1])

Ntst_pos = len(ytst[ytst==+1])

Ntst_neg = len(ytst[ytst==-1])

IR      = N_neg/N_pos

IR_tst = Ntst_neg/Ntst_pos

#print('N_pos: ', N_pos,' - N_neg: ',N_neg,
' - IR : ', IR)
```

```
#print('Ntst_pos: ', Ntst_pos,' - Ntst_neg: ',
Ntst_neg, ' - IR_tst: ', IR_tst)


#%% normalization


#X, scaler      = preprocess_data(X)
#xtst,scaler    = preprocess_data(xtst,scaler)


  #%% Parametros Bayes


alphas = np.linspace(0.1, 0.9, 9)
idx_05=alphas==0.5
n_alphas = len(alphas)
R = 5
repetitions = np.arange(R)
auc_tr_repts=np.zeros((1,R))
auc_val_repts=np.zeros((1,R))
auc_tst_repts=np.zeros((1,R))


#%%
PFA_tr = np.zeros((n_alphas + 2,1))
PFA_tr[0] = 1;


PD_tr  = np.zeros((n_alphas + 2,1))
PD_tr[0] = 1;


PFAs_tr = np.zeros((n_alphas+2,R))
PFAs_tr[0,:] =1
PDs_tr = np.zeros((n_alphas+2,R))
PDs_tr[0,:] =1


#%%
PFA_val = np.zeros((n_alphas + 2,1))
PFA_val[0] = 1;


PD_val  = np.zeros((n_alphas + 2,1))
PD_val[0] = 1;


PFAs_val = np.zeros((n_alphas+2,R))
PFAs_val[0,:] =1
PDs_val = np.zeros((n_alphas+2,R))
PDs_val[0,:] =1


#%%
PFA_tst = np.zeros((n_alphas + 2,1))
PFA_tst[0] = 1;


PD_tst  = np.zeros((n_alphas + 2,1))
PD_tst[0] = 1;


PFAs_tst = np.zeros((n_alphas+2,R))
PFAs_tst[0,:] =1
PDs_tst = np.zeros((n_alphas+2,R))
PDs_tst[0,:] =1


#%%


Ntr_pos  = len(ytr[ytr==+1])
Ntr_neg  = len(ytr[ytr==-1])


Nval_pos = len(ytst[ytst==+1])
Nval_neg = len(ytst[ytst==-1])


Ntr = len(xtr)


P0 = Ntr_neg/Ntr    #probabilidad de la
clase 0
P1 = Ntr_pos/Ntr    #probabilidad de la
clase 1
```

```
acc = np.zeros((n_alphas,1))


costes_matrix_tr = np.zeros((n_alphas,5))

costes_matrix_val = np.zeros((n_alphas,5))

costes_matrix_tst = np.zeros((n_alphas,5))

#%%

alphas_idxs =  np.arange(0,n_alphas)

optzer = RMSprop()

earlyStopping =
EarlyStopping(monitor='val_loss',
patience=200, verbose=0, mode='min')

nb_epoch = 20

t_tr = ytr > 0.5

y_tr = t_tr.astype('float32')

t_val = yval > 0.5

y_val = t_val.astype('float32')

t_tst = ytst > 0.5

y_tst = t_tst.astype('float32')



for ik in repetitions:

    k = ik + 1

    print('Num. de Repeticion = ', k)


    #%%

    acc = np.zeros((n_alphas,1))
#array para guardar el resultado en
repet.

    print('---'*10,'Rept.  ',k,'---'*10)
#para simular separacion en pantalla


    for ialpha in alphas_idxs:

        #%%

        np.random.seed(ik)  # for
reproducibility

        alpha_num = alphas[ialpha];

        print('alpha = ', alpha_num)

        C10 = alpha_num/P0
#penalizacion cuando se falla en la clase
"0"

        C01 = (1 - alpha_num)/P1
#penalizacion cuando se falla en la clase
"1"

        class_weight = {0:C10, 1:C01}
#[C01,C10]

        #print(class_weight)

        model = Sequential()

        model.add(Dense(output_dim = NN
, input_dim = D))

        model.add(Activation('tanh'))

        model.add(Dense(output_dim = 1 ,
input_dim = NN))

        model.add(Activation('sigmoid'))

        model.compile(loss =
'binary_crossentropy', optimizer = optzer)

        coste = model.fit(xtr, y_tr,
batch_size=Ntr, nb_epoch=nb_epoch,
verbose=0, callbacks=[earlyStopping],
validation_data=(xval, y_val), class_weight =
class_weight)

        #score = model.evaluate( xval,
y_val, show_accuracy = True, verbose = 0
)

        score = model.evaluate( xtst,
y_tst, verbose = 0 )

        print('Test score en rept.    ' , k
,' = ', score)

        acc[ialpha] = score

        print('---fin rept.---')

        #%%

        coste1 =  coste.history

        plt.figure(10+k)

        plt.subplot(3,3,ialpha+1)

        plt.plot(coste1['loss'])       # a
diferencia del matlab no es necesario
poner hold on para sobreponer otra
grafica
```

```
        plt.plot(coste1['val_loss'])  # a diferencia del matlab no es necesario poner hold on para sobreponer otra grafica

        plt.xlabel('epochs')

        plt.ylabel('R')

        plt.legend(('Tr','Val'))

        plt.title( r'$\alpha $')

        plt.title( alpha_num, loc='right')


        #%%

        ye_tr = model.predict(xtr, batch_size=Ntr)

        te_tr = ye_tr > 0.5

        Ye_tr = 2.0*(te_tr.astype('float32')) - 1.0

        #

        w00, w10, w01, w11 = calcula_tabla(ytr,Ye_tr)

        costes_matrix_tr[ialpha,:] = np.array([w00, w10, w01, w11, alpha_num*w10 + (1. - alpha_num)* w01])

        #%%

        ye_val = model.predict(xval, batch_size=Nval)

        te_val = ye_val > 0.5

        Ye_val = 2.0*(te_val.astype('float32')) - 1.0

        #

        w00, w10, w01, w11 = calcula_tabla(yval,Ye_val)

        costes_matrix_val[ialpha,:] = np.array([w00, w10, w01, w11, alpha_num*w10 + (1. - alpha_num)* w01])

        #%%

        ye_tst = model.predict(xtst, batch_size=Ntst)

        te_tst = ye_tst > 0.5

        Ye_tst = 2.0*(te_tst.astype('float32')) - 1.0

        #

        w00, w10, w01, w11 = calcula_tabla(ytst,Ye_tst)

        costes_matrix_tst[ialpha,:] = np.array([w00, w10, w01, w11, alpha_num*w10 + (1. - alpha_num)* w01])

    #%%

    PFA_tr[1:n_alphas+1] = np.reshape(costes_matrix_tr[:,1],(n_alphas,1))

    PD_tr[1:n_alphas+1]  = np.reshape(costes_matrix_tr[:,3],(n_alphas,1))

    auc_tr,ps_tr = calcula_auc_ps(PFA_tr,PD_tr,idx_05)

    auc_tr_repts[0,ik]=auc_tr


    PFAs_tr[:,ik] = np.reshape(PFA_tr[:],11);

    PDs_tr[:,ik]  = np.reshape(PD_tr[:],11);


    PFA_val[1:n_alphas+1] = np.reshape(costes_matrix_val[:,1],(n_alphas,1))

    PD_val[1:n_alphas+1]  = np.reshape(costes_matrix_val[:,3],(n_alphas,1))

    auc_val,ps_val = calcula_auc_ps(PFA_val,PD_val,idx_05)

    auc_val_repts[0,ik]=auc_val


    PFAs_val[:,ik] = np.reshape(PFA_val[:],11);

    PDs_val[:,ik]  = np.reshape(PD_val[:],11);


    PFA_tst[1:n_alphas+1] = np.reshape(costes_matrix_tst[:,1],(n_alphas,1))

    PD_tst[1:n_alphas+1]  = np.reshape(costes_matrix_tst[:,3],(n_alphas,1))
```

```
        auc_tst,ps_tst =
calcula_auc_ps(PFA_tst,PD_tst,idx_05)

        auc_tst_repts[0,ik]=auc_tst


        PFAs_tst[:,ik] =
np.reshape(PFA_tst[:],11);

        PDs_tst[:,ik]   =
np.reshape(PD_tst[:],11);




        #%%

        plt.figure(k)

        plt.plot(PFA_tr,PD_tr)          # a
diferencia del matlab no es necesario
poner hold on para sobreponer otra
grafica

        plt.plot(PFA_val,PD_val,'r')

        plt.plot(PFA_tst,PD_tst,'g')

        plt.xlabel('PFA')

        plt.ylabel('PD')

        #%%

auc_tr_rep = np.mean(auc_tr_repts[0])

auc_val_rep = np.mean(auc_val_repts[0])

auc_tst_rep = np.mean(auc_tst_repts[0])

print('auc tr para los s repeticiones: ',
auc_tr_repts)

print('auc tr media de s repeticiones: ',
auc_tr_rep, 'para NN = ', NN)


print('auc val para los s repeticiones: ',
auc_val_repts)

print('auc val media de s repeticiones: ',
auc_val_rep, 'para NN = ', NN)




print('auc tst para los s repeticiones: ',
auc_tst_repts)

print('auc tst media de s repeticiones: ',
auc_tst_rep, 'para NN = ', NN)
```

```
#%%


PFA_tr_mean = np.mean(PFAs_tr,axis=1)

PD_tr_mean = np.mean(PDs_tr,axis=1)

auc_tr_mean,ps_tst_mean =
calcula_auc_ps(PFA_tr_mean,PD_tr_mean,id
x_05)


PFA_val_mean = np.mean(PFAs_val,axis=1)

PD_val_mean = np.mean(PDs_val,axis=1)

auc_val_mean,ps_val_mean =
calcula_auc_ps(PFA_val_mean,PD_val_mean,i
dx_05)


PFA_tst_mean = np.mean(PFAs_tst,axis=1)

PD_tst_mean = np.mean(PDs_tst,axis=1)

auc_tst_mean,ps_tst_mean =
calcula_auc_ps(PFA_tst_mean,PD_tst_mean,i
dx_05)

#%%

auc_tr_mean_str = str(auc_tr_mean);

auc_val_mean_str = str(auc_val_mean);

auc_tst_mean_str = str(auc_tst_mean);


legends=('Tr: ' + auc_tr_mean_str[0:5]
,'Val: ' + auc_val_mean_str[0:5] ,'Tst: ' +
auc_tst_mean_str[0:5] )

plt.figure(101)

plt.plot(PFA_tr_mean,PD_tr_mean)

plt.plot(PFA_val_mean,PD_val_mean,':r')

plt.plot(PFA_tst_mean,PD_tst_mean,'--g')

plt.xlabel('P_{FA}'), plt.ylabel('P_{D}'), #
plt.axis([np.min(NNs), np.max(NNs), 98, 100])

plt.legend(legends)

#%%

np.savez('resultados/Resultado_NN50_e20_fin
al.npz',PFAs_tr=PFAs_tr,PDs_tr=PDs_tr,
```

```
PFAs_val=PFAs_val,PDs_val=PDs_val,
            PFAs_tst=PFAs_tst,PDs_tst=PDs_tst,
            PFA_tr_mean =
PFA_tr_mean,PD_tr_mean=PD_tr_mean,
            PFA_val_mean=PFA_val_mean,
PD_val_mean=PD_val_mean,
            PFA_tst_mean = PFA_tst_mean,
PD_tst_mean=PD_tst_mean,
auc_tr_mean_str=auc_tr_mean_str,
auc_val_mean_str=auc_val_mean_str,
auc_tst_mean_str=auc_tst_mean_str
                )
```

3. Archivo "main_class_weigths_dnn_cros_entrop_validacion.py"

```
# cargar librerias

from __future__ import absolute_import

from __future__ import print_function
#forma de displayar en pantalla

from __future__ import division #division
con decimales

import numpy as np #manejo de matrices

from scipy.io import loadmat #libreria
cientifica, abrir archivos.mat

from keras.models import Sequential #
haer el modelo en keras y el sequential
empezar a realizar el modelo

from keras.layers.core import Dense,
Activation, Dropout#, Dense conecion de
neuronas una a una, activacion tipo de
funcion a activar

from keras.optimizers import RMSprop
#SGD #Adam # SGD #, #RMSprop desenso
por gradiente, forma automatica tasa de
aprendizaje

from keras.callbacks import
EarlyStopping #para cuando converge el
conj de validacion

import matplotlib.pyplot as plt  #de la
libreria matplot importa pyplot como plt

from functools import partial #aepte
variables que no estan en el programa

np.random.seed(1337)  # for reproducibility

#%% Funciones propias en extrafunctions y
bayesian_cost_functions

from extrafunctions import
calcula_tabla,calcula_auc_ps

#from extrafunctions import
preprocess_data

#%%

nb_classes = 2  #numero de clases

#%%

NN = 50   #numero de neuronas en la
capa oculta

d  = 0.25 #porcentaje de dropout [0.25
0.50 0.75]

nH = 2   #num. de capas ocultas [1 2 3]

nb_epoch = 100

#%%

alphas = np.linspace(0.1, 0.9, 9)

idx_05=alphas==0.5

n_alphas = len(alphas)

PFA_kfolds_val = np.zeros((n_alphas + 2,5))

PFA_kfolds_val[0,:] = 1;

PD_kfolds_val  = np.zeros((n_alphas + 2,5))

PD_kfolds_val[0,:] = 1;

PFA_kfolds_tr = np.zeros((n_alphas + 2,5))

PFA_kfolds_tr[0,:] = 1;

PD_kfolds_tr  = np.zeros((n_alphas + 2,5))

PD_kfolds_tr[0,:] = 1;



acc = np.zeros((n_alphas,1))

costes_matrix_tr = np.zeros((n_alphas,5))

costes_matrix = np.zeros((n_alphas,5))


PFA = np.zeros((n_alphas + 2,1))

PFA[0] = 1;


PD  = np.zeros((n_alphas + 2,1))

PD[0] = 1;



PFA_tr = np.zeros((n_alphas + 2,1))

PFA_tr[0] = 1;
```

```
PD_tr   = np.zeros((n_alphas + 2,1))
PD_tr[0] = 1;
#%% Cargar Datos
Kfolds = np.arange(5)
auc_5fold=np.zeros((1,5))
auc_5fold_tr=np.zeros((1,5))


for ik in Kfolds:
    k = ik + 1
    print('Num. de Fold = ', k)
    Datos_all =
loadmat('D:\TesisMel\datos\KDD_norm_tr_fol
d_' + str(k) + '.mat')

    #%% Formato adecuado "float32"
    X = Datos_all['x_tr'].astype("float32")
    Y = Datos_all['y_tr'].astype("float32")

    Xtst =
Datos_all['x_val'].astype("float32")
    Ytst =
Datos_all['y_val'].astype("float32")

    D = X.shape[1]
    N = len(X)
    Ntst = len(Xtst)

    #%%
    #print('All samples: ', N)
    #print('Test samples: ', Ntst)
    N_pos = len(Y[Y==+1])
    N_neg = len(Y[Y==-1])
    Ntst_pos = len(Ytst[Ytst==+1])
    Ntst_neg = len(Ytst[Ytst==-1])
    IR       = N_neg/N_pos
    IR_tst = Ntst_neg/Ntst_pos
    #print('N_pos: ', N_pos,' - N_neg:
',N_neg, ' - IR : ', IR)
    #print('Ntst_pos: ', Ntst_pos,' -
Ntst_neg: ', Ntst_neg, ' - IR_tst: ', IR_tst)

    #%% normalization

    #X, scaler      = preprocess_data(X)
    #Xtst,scaler    =
preprocess_data(Xtst,scaler)
    #%% Parametros Bayes

    #%%
    f = k;
    print('---'*10,'Fold ',f,'---'*10)
#para simular separacion en pantalla
    xtr = X
    ytr = Y
    xval= Xtst
    yval= Ytst

    Ntr_pos   = len(ytr[ytr==+1])
    Ntr_neg   = len(ytr[ytr==-1])

    Nval_pos = len(yval[yval==+1])
    Nval_neg = len(yval[yval==-1])

    Ntr = len(xtr)

    P0 = Ntr_neg/Ntr    #probabilidad de
la clase 0
    P1 = Ntr_pos/Ntr    #probabilidad de
la clase 1
```

```
        #%%

        alphas_idxs =  np.arange(0,n_alphas)

        optzer = RMSprop()

        earlyStopping =
EarlyStopping(monitor='val_loss',
patience=10, verbose=0, mode='min')

        t_tr = ytr > 0.5

        y_tr = t_tr.astype('float32')

        t_val = yval > 0.5

        y_val = t_val.astype('float32')

        t_tst = Ytst > 0.5

        y_tst = t_tst.astype('float32')



        for ialpha in alphas_idxs:

            np.random.seed(1337)  # for
reproducibility

            alpha_num = alphas[ialpha];

            print('alpha = ', alpha_num)

            C10 = alpha_num/P0
#penalizacion cuando se falla en la clase
"0"

            C01 = (1 - alpha_num)/P1
#penalizacion cuando se falla en la clase
"1"

            class_weight = {0:C10, 1:C01}
#[C01,C10]

            #print(class_weight)

            model = Sequential()

            model.add(Dense(output_dim = NN
, input_dim = D))

            model.add(Activation('tanh'))

            model.add(Dropout(d))

            if nH>1:

                model.add(Dense(output_dim =
NN , input_dim = NN))

                model.add(Activation('tanh'))

                model.add(Dropout(d))

            if nH>2:

                model.add(Dense(output_dim =
NN , input_dim = NN))

                model.add(Activation('tanh'))

                model.add(Dropout(d))


            model.add(Dense(output_dim = 1 ,
input_dim = NN))

            model.add(Activation('sigmoid'))

            model.compile(loss =
'binary_crossentropy', optimizer = optzer)

            coste = model.fit(xtr, y_tr,
batch_size=Ntr, nb_epoch=nb_epoch,
verbose=0, callbacks=[earlyStopping],
validation_data=(xval, y_val), class_weight =
class_weight)

            score = model.evaluate( xval,
y_val, verbose = 0 )

            print('Test score en fold     ' , f
,' = ', score)

            acc[ialpha] = score

            print('---fin fold---')

            #%%

            coste1 =  coste.history

            plt.figure(10+k)

            plt.subplot(3,3,ialpha+1)

            plt.plot(coste1['loss'])       # a
diferencia del matlab no es necesario
poner hold on para sobreponer otra
grafica

            plt.plot(coste1['val_loss'],'--')  # a
diferencia del matlab no es necesario
poner hold on para sobreponer otra
grafica

            plt.xlabel('epochs')

            plt.ylabel('H')

            plt.legend(('Tr','Val'))

            plt.title( r'$\alpha $')
```

```
            plt.title( alpha_num, loc='right')

            #%%

            ye_tst = model.predict(Xtst,
batch_size=Ntst)

            te_tst = ye_tst > 0.5

            Ye_tst =
2.0*(te_tst.astype('float32')) - 1.0

            #%%

            ye_tr = model.predict(xtr,
batch_size=Ntr)

            te_tr = ye_tr > 0.5

            Ye_tr = 2.0*(te_tr.astype('float32'))
- 1.0

            w00, w10, w01, w11 =
calcula_tabla(ytr,Ye_tr)

            costes_matrix_tr[ialpha,:] =
np.array([w00, w10, w01, w11,
alpha_num*w10 + (1. - alpha_num)* w01])




            ye_tst = model.predict(Xtst,
batch_size=Ntst)

            te_tst = ye_tst > 0.5

            Ye_tst =
2.0*(te_tst.astype('float32')) - 1.0

            w00, w10, w01, w11 =
calcula_tabla(Ytst,Ye_tst)

            costes_matrix[ialpha,:] =
np.array([w00, w10, w01, w11,
alpha_num*w10 + (1. - alpha_num)* w01])

        #%%


        PFA[1:n_alphas+1] =
np.reshape(costes_matrix[:,1],(n_alphas,1))

        PD[1:n_alphas+1]  =
np.reshape(costes_matrix[:,3],(n_alphas,1))

        auc,ps = calcula_auc_ps(PFA,PD,idx_05)

        auc_5fold[0,ik]=auc


        PFA_tr[1:n_alphas+1] =
np.reshape(costes_matrix_tr[:,1],(n_alphas,1
))

        PD_tr[1:n_alphas+1]  =
np.reshape(costes_matrix_tr[:,3],(n_alphas,1
))

        auc_tr,ps_tr =
calcula_auc_ps(PFA_tr,PD_tr,idx_05)

        auc_5fold_tr[0,ik]=auc_tr


        PFA_kfolds_val[:,ik] = PFA[:,0];

        PD_kfolds_val[:,ik]  = PD[:,0];

        PFA_kfolds_tr[:,ik]  = PFA_tr[:,0];

        PD_kfolds_tr[:,ik]   = PD_tr[:,0];



        plt.figure(k)

        plt.plot(PFA_tr,PD_tr)

        plt.plot(PFA,PD,'--')       # a
diferencia del matlab no es necesario
poner hold on para sobreponer otra
grafica

        plt.legend(('Tr.','Val.'))

        plt.xlabel('PFA')

        plt.ylabel('PD')


#%%


print('Número de capas', nH,'porcentaje de
dropout', d )

auc_sim_tr = np.mean(auc_5fold_tr[0])

print('auc tr. para los 5 folds: ',
auc_5fold_tr)

print('auc tr. media de los 5 folds: ',
auc_sim_tr, 'para NN = ' , NN)


auc_sim = np.mean(auc_5fold[0])

print('auc val. para los 5 folds: ',
auc_5fold)
```

```
print('auc val. media de los 5 folds: ',
auc_sim, 'para NN = ' , NN)


#%%

file_name =
'resultados_mats/DNN_norm_NH_'+ str(nH)
+ '_NN_' + str(NN) +'_d_' + str(d) +
'_epochs_'+ str(nb_epoch)+
'_validacion.npz'

np.savez(file_name, auc_sim_tr =
auc_sim_tr, auc_sim = auc_sim,
auc_5fold_tr=auc_5fold_tr,
auc_5fold=auc_5fold,

          PFA_kfolds_val= PFA_kfolds_val,

          PD_kfolds_val = PD_kfolds_val,

          PFA_kfolds_tr = PFA_kfolds_tr,

          PD_kfolds_tr  = PD_kfolds_tr)


#%%

PFA_mean_kfolds_tr =
np.mean(PFA_kfolds_tr,axis=1)

PFA_std_kfolds_tr =
np.std(PFA_kfolds_tr,axis=1)


PD_mean_kfolds_tr =
np.mean(PD_kfolds_tr,axis=1)

PD_std_kfolds_tr =
np.std(PD_kfolds_tr,axis=1)



PFA_mean_kfolds_val =
np.mean(PFA_kfolds_val,axis=1)

PFA_std_kfolds_val =
np.std(PFA_kfolds_val,axis=1)


PD_mean_kfolds_val =
np.mean(PD_kfolds_val,axis=1)

PD_std_kfolds_val =
np.std(PD_kfolds_val,axis=1)
```

```
plt.figure(200)

plt.plot(PFA_mean_kfolds_tr,
PD_mean_kfolds_tr, 'b')

plt.plot(PFA_mean_kfolds_val,
PD_mean_kfolds_val, '--g')

plt.legend(('Tr.','Val.'))

plt.xlabel('$P_{FA}$')

plt.ylabel('$P_D$')



plt.figure(201)

plt.errorbar(PFA_mean_kfolds_tr,
PD_mean_kfolds_tr,
yerr=[PD_std_kfolds_tr, PD_std_kfolds_tr],
xerr=[PFA_std_kfolds_tr,
PFA_std_kfolds_tr], fmt='b')

plt.errorbar(PFA_mean_kfolds_val,
PD_mean_kfolds_val,
yerr=[PD_std_kfolds_val,
PD_std_kfolds_val],
xerr=[PFA_std_kfolds_val,
PFA_std_kfolds_val], fmt='--g')

plt.legend(('Tr.','Val.'))

plt.xlabel('$P_{FA}$')

plt.ylabel('$P_D$')

plt.figure(202)

plt.errorbar(PFA_mean_kfolds_tr,
PD_mean_kfolds_tr,
yerr=[PD_std_kfolds_tr, PD_std_kfolds_tr],
xerr=[PFA_std_kfolds_tr,
PFA_std_kfolds_tr], fmt='b')

plt.errorbar(PFA_mean_kfolds_val,
PD_mean_kfolds_val,
yerr=[PD_std_kfolds_val,
PD_std_kfolds_val],
xerr=[PFA_std_kfolds_val,
PFA_std_kfolds_val], fmt='--g')

plt.legend(('Tr.','Val.'))

plt.xlabel('$P_{FA}$')

plt.ylabel('$P_D$')

plt.axis([0, 0.05, 0.985, 1]) #ìzoom
```

4. Archivo "main_class_weigths_dnn_cros_entrop_clasificador_final_NN.py"

```
# cargar librerias

from __future__ import absolute_import

from __future__ import print_function
#forma de displayar en pantalla

from __future__ import division #division
con decimales

import numpy as np #manejo de matrices

from scipy.io import loadmat #libreria
cientifica, abrir archivos.mat

from keras.models import Sequential #
haer el modelo en keras y el sequential
empezar a realizar el modelo

from keras.layers.core import Dense,
Activation, Dropout#, Dense conecion de
neuronas una a una, activacion tipo de
funcion a activar

from keras.optimizers import RMSprop
#SGD #Adam # SGD #, #RMSprop desenso
por gradiente, forma automatica tasa de
aprendizaje

from keras.callbacks import
EarlyStopping #para cuando converge el
conj de validacion

import matplotlib.pyplot as plt #de la
libreria matplot importa pyplot como plt

#from functools import partial #aepte
variables que no estan en el programa


#%% Funciones propias en extrafunctions y
bayesian_cost_functions

from extrafunctions import
calcula_tabla,calcula_auc_ps

#from extrafunctions import
preprocess_data

#%%

nb_classes = 2  #numero de clases

#%%

NN = 50   #mejor numero de neuronas en
la capa oculta

d  = 0.25 #mejor porcentaje de dropout
[0.25 0.50 0.75]

nH = 2    #mejor num. de capas ocultas
[1 2 3]

nb_epoch = 20

R = 10

#%% Cargar Datos

Datos_all =
loadmat('D:\TesisMel\datos\KDD_norm_tr_fin
al' + '.mat')

#%% Formato adecuado "float32"

xtr = Datos_all['x_tr'].astype("float32")

ytr = Datos_all['y_tr'].astype("float32")

xval = Datos_all['x_val'].astype("float32")

yval = Datos_all['y_val'].astype("float32")

Datos_all =
loadmat('D:\TesisMel\datos\KDD_norm_tst_fi
nal' + '.mat')

xtst = Datos_all['x_tst'].astype("float32")

ytst = Datos_all['y_tst'].astype("float32")


D = xtr.shape[1]

N = len(xtr)

Nval = len(xval)

Ntst = len(xtst)


#%%

#print('All samples: ', N)

#print('Test samples: ', Ntst)

N_pos = len(ytr[ytr==+1])

N_neg = len(ytr[ytr==-1])

Ntst_pos = len(ytst[ytst==+1])

Ntst_neg = len(ytst[ytst==-1])

IR     = N_neg/N_pos
```

```
IR_tst = Ntst_neg/Ntst_pos

#print('N_pos: ', N_pos,' - N_neg: ',N_neg,
' - IR : ', IR)

#print('Ntst_pos: ', Ntst_pos,' - Ntst_neg: ',
Ntst_neg, ' - IR_tst: ', IR_tst)


#%% normalization


#X, scaler      = preprocess_data(X)

#xtst,scaler    = preprocess_data(xtst,scaler)


  #%% Parametros Bayes


alphas = np.linspace(0.1, 0.9, 9)

idx_05=alphas==0.5

n_alphas = len(alphas)


repetitions = np.arange(R)

auc_tr_repts=np.zeros((1,R))

auc_val_repts=np.zeros((1,R))

auc_tst_repts=np.zeros((1,R))


#%%

PFA_tr = np.zeros((n_alphas + 2,1))

PFA_tr[0] = 1;


PD_tr   = np.zeros((n_alphas + 2,1))

PD_tr[0] = 1;


PFAs_tr = np.zeros((n_alphas+2,R))

PFAs_tr[0,:] =1

PDs_tr = np.zeros((n_alphas+2,R))

PDs_tr[0,:] =1


#%%

PFA_val = np.zeros((n_alphas + 2,1))

PFA_val[0] = 1;


PD_val   = np.zeros((n_alphas + 2,1))

PD_val[0] = 1;


PFAs_val = np.zeros((n_alphas+2,R))

PFAs_val[0,:] =1

PDs_val = np.zeros((n_alphas+2,R))

PDs_val[0,:] =1


#%%

PFA_tst = np.zeros((n_alphas + 2,1))

PFA_tst[0] = 1;


PD_tst   = np.zeros((n_alphas + 2,1))

PD_tst[0] = 1;


PFAs_tst = np.zeros((n_alphas+2,R))

PFAs_tst[0,:] =1

PDs_tst = np.zeros((n_alphas+2,R))

PDs_tst[0,:] =1


#%%


Ntr_pos   = len(ytr[ytr==+1])

Ntr_neg   = len(ytr[ytr==-1])


Nval_pos = len(ytst[ytst==+1])

Nval_neg = len(ytst[ytst==-1])


Ntr = len(xtr)
```

```
P0 = Ntr_neg/Ntr    #probabilidad de la clase 0
P1 = Ntr_pos/Ntr    #probabilidad de la clase 1

acc = np.zeros((n_alphas,1))

costes_matrix_tr = np.zeros((n_alphas,5))
costes_matrix_val = np.zeros((n_alphas,5))
costes_matrix_tst = np.zeros((n_alphas,5))
#%%
alphas_idxs =  np.arange(0,n_alphas)
optzer = RMSprop()
earlyStopping = EarlyStopping(monitor='val_loss', patience=50, verbose=0, mode='min')

t_tr = ytr > 0.5
y_tr = t_tr.astype('float32')
t_val = yval > 0.5
y_val = t_val.astype('float32')
t_tst = ytst > 0.5
y_tst = t_tst.astype('float32')

for ik in repetitions:
    k = ik + 1
    print('Num. de Repeticion = ', k)

    #%%
    print('---'*10,'Rept.  ',k,'---'*10) #para simular separacion en pantalla

    for ialpha in alphas_idxs:
        #%%
        np.random.seed(ik)  # for reproducibility
        alpha_num = alphas[ialpha];
        print('alpha = ', alpha_num)
        C10 = alpha_num/P0 #penalizacion cuando se falla en la clase "0"
        C01 = (1 - alpha_num)/P1 #penalizacion cuando se falla en la clase "1"
        class_weight = {0:C10, 1:C01} #[C01,C10]
        #print(class_weight)
        model = Sequential()
        model.add(Dense(output_dim = NN , input_dim = D))
        model.add(Activation('tanh'))
        model.add(Dropout(d))
        if nH>1:
            model.add(Dense(output_dim = NN , input_dim = NN))
            model.add(Activation('tanh'))
            model.add(Dropout(d))
        if nH>2:
            model.add(Dense(output_dim = NN , input_dim = NN))
            model.add(Activation('tanh'))
            model.add(Dropout(d))

        model.add(Dense(output_dim = 1 , input_dim = NN))
        model.add(Activation('sigmoid'))
        model.compile(loss = 'binary_crossentropy', optimizer = optzer)
        coste = model.fit(xtr, y_tr, batch_size=1000, nb_epoch=nb_epoch, verbose=0, callbacks=[earlyStopping], validation_data=(xval, y_val), class_weight = class_weight)
```

```
        #score = model.evaluate( xval,
y_val, show_accuracy = True, verbose = 0
)

        score = model.evaluate( xtst,
y_tst, verbose = 0 )

        print('Test score en rept.    ' , k
,' = ', score)

        acc[ialpha] = score

        print('---fin rept.---')

        #%%

        coste1 =  coste.history

        plt.figure(10+k)

        plt.subplot(3,3,ialpha+1)

        plt.plot(coste1['loss'])      # a
diferencia del matlab no es necesario
poner hold on para sobreponer otra
grafica

        plt.plot(coste1['val_loss'])  # a
diferencia del matlab no es necesario
poner hold on para sobreponer otra
grafica

        plt.xlabel('epochs')

        plt.ylabel('R')

        plt.legend(('Tr','Val'))

        plt.title( r'$\alpha $')

        plt.title( alpha_num, loc='right')


        #%%

        ye_tr = model.predict(xtr,
batch_size=Ntr)

        te_tr = ye_tr > 0.5

        Ye_tr = 2.0*(te_tr.astype('float32'))
- 1.0

        #

        w00, w10, w01, w11 =
calcula_tabla(ytr,Ye_tr)

        costes_matrix_tr[ialpha,:] =
np.array([w00, w10, w01, w11,
alpha_num*w10 + (1. - alpha_num)* w01])

        #%%

        ye_val = model.predict(xval,
batch_size=Nval)

        te_val = ye_val > 0.5

        Ye_val =
2.0*(te_val.astype('float32')) - 1.0

        #

        w00, w10, w01, w11 =
calcula_tabla(yval,Ye_val)

        costes_matrix_val[ialpha,:] =
np.array([w00, w10, w01, w11,
alpha_num*w10 + (1. - alpha_num)* w01])

        #%%

        ye_tst = model.predict(xtst,
batch_size=Ntst)

        te_tst = ye_tst > 0.5

        Ye_tst =
2.0*(te_tst.astype('float32')) - 1.0

        #

        w00, w10, w01, w11 =
calcula_tabla(ytst,Ye_tst)

        costes_matrix_tst[ialpha,:] =
np.array([w00, w10, w01, w11,
alpha_num*w10 + (1. - alpha_num)* w01])

    #%%

    PFA_tr[1:n_alphas+1] =
np.reshape(costes_matrix_tr[:,1],(n_alphas,1
))

    PD_tr[1:n_alphas+1]  =
np.reshape(costes_matrix_tr[:,3],(n_alphas,1
))

    auc_tr,ps_tr =
calcula_auc_ps(PFA_tr,PD_tr,idx_05)

    auc_tr_repts[0,ik]=auc_tr


    PFAs_tr[:,ik] = np.reshape(PFA_tr[:],11);

    PDs_tr[:,ik]  = np.reshape(PD_tr[:],11);


    PFA_val[1:n_alphas+1] =
np.reshape(costes_matrix_val[:,1],(n_alphas,1
))
```

```
    PD_val[1:n_alphas+1]  =
np.reshape(costes_matrix_val[:,3],(n_alphas,1
))

    auc_val,ps_val =
calcula_auc_ps(PFA_val,PD_val,idx_05)

    auc_val_repts[0,ik]=auc_val



    PFAs_val[:,ik] =
np.reshape(PFA_val[:],11);

    PDs_val[:,ik]  =
np.reshape(PD_val[:],11);



    PFA_tst[1:n_alphas+1] =
np.reshape(costes_matrix_tst[:,1],(n_alphas,1
))

    PD_tst[1:n_alphas+1]  =
np.reshape(costes_matrix_tst[:,3],(n_alphas,1
))

    auc_tst,ps_tst =
calcula_auc_ps(PFA_tst,PD_tst,idx_05)

    auc_tst_repts[0,ik]=auc_tst



    PFAs_tst[:,ik] =
np.reshape(PFA_tst[:],11);

    PDs_tst[:,ik]  =
np.reshape(PD_tst[:],11);



    #%%

    plt.figure(k)

    plt.plot(PFA_tr,PD_tr)        # a
diferencia del matlab no es necesario
poner hold on para sobreponer otra
grafica

    plt.plot(PFA_val,PD_val,'r')

    plt.plot(PFA_tst,PD_tst,'g')

    plt.xlabel('PFA')

    plt.ylabel('PD')

    #%%

auc_tr_rep = np.mean(auc_tr_repts[0])

auc_val_rep = np.mean(auc_val_repts[0])

auc_tst_rep = np.mean(auc_tst_repts[0])

print('batch_size ' , batch_size)

print('auc tr para los s repeticiones: ',
auc_tr_repts)

print('auc tr media de s repeticiones: ',
auc_tr_rep, 'para NN = ' , NN)


print('auc val para los s repeticiones: ',
auc_val_repts)

print('auc val media de s repeticiones: ',
auc_val_rep, 'para NN = ' , NN)


print('auc tst para los s repeticiones: ',
auc_tst_repts)

print('auc tst media de s repeticiones: ',
auc_tst_rep, 'para NN = ' , NN)

#%%


PFA_tr_mean = np.mean(PFAs_tr,axis=1)

PD_tr_mean = np.mean(PDs_tr,axis=1)

auc_tr_mean,ps_tst_mean =
calcula_auc_ps(PFA_tr_mean,PD_tr_mean,id
x_05)


PFA_val_mean = np.mean(PFAs_val,axis=1)

PD_val_mean = np.mean(PDs_val,axis=1)

auc_val_mean,ps_val_mean =
calcula_auc_ps(PFA_val_mean,PD_val_mean,i
dx_05)


PFA_tst_mean = np.mean(PFAs_tst,axis=1)

PD_tst_mean = np.mean(PDs_tst,axis=1)

auc_tst_mean,ps_tst_mean =
calcula_auc_ps(PFA_tst_mean,PD_tst_mean,i
dx_05)

#%%
```

```
auc_tr_mean_str = str(auc_tr_mean);

auc_val_mean_str = str(auc_val_mean);

auc_tst_mean_str = str(auc_tst_mean);


legends=('Tr: ' + auc_tr_mean_str[0:5]
,'Val: ' + auc_val_mean_str[0:5] ,'Tst: ' +
auc_tst_mean_str[0:5] )

plt.figure(101)

plt.plot(PFA_tr_mean,PD_tr_mean)

plt.plot(PFA_val_mean,PD_val_mean,':r')

plt.plot(PFA_tst_mean,PD_tst_mean,'--g')

plt.xlabel(r'$P_{FA{$'), plt.ylabel(r'$P_{D{$'),
# plt.axis([np.min(NNs), np.max(NNs), 98,
100])

plt.legend(legends)

#%%

batch_size = 1000

file_name =
'resultados_mats/DNN_norm_NH_'+ str(nH)
+ '_NN_' + str(NN) +'_d_' + str(d) +
'_epochs_'+ str(nb_epoch)+
'_BS_'+str(batch_size) + '_final.npz'


np.savez(file_name,auc_tr_mean_str=auc_tr
_mean_str,

          auc_val_mean_str =
auc_val_mean_str,

          auc_tst_mean_str =
auc_tst_mean_str,

          PFAs_tr=PFAs_tr,PDs_tr=PDs_tr,


PFAs_val=PFAs_val,PDs_val=PDs_val,


PFAs_tst=PFAs_tst,PDs_tst=PDs_tst)



print('Número de capas', nH,'porcentaje de
dropout', d, ' y NN = ' , NN )

print('auc media de las S sims: Tr ',
auc_tr_mean_str)

print('auc media de las S sims: Val ',
auc_val_mean_str)

print('auc media de las S sims: Tst ',
auc_tst_mean_str)


#%%

PFA_mean_reps_tr =
np.mean(PFAs_tr,axis=1)

PFA_std_reps_tr = np.std(PFAs_tr,axis=1)


PD_mean_reps_tr = np.mean(PDs_tr,axis=1)

PD_std_reps_tr = np.std(PDs_tr,axis=1)


PFA_mean_reps_val =
np.mean(PFAs_val,axis=1)

PFA_std_reps_val = np.std(PFAs_val,axis=1)


PD_mean_reps_val =
np.mean(PDs_val,axis=1)

PD_std_reps_val = np.std(PDs_val,axis=1)


PFA_mean_reps_tst =
np.mean(PFAs_tst,axis=1)

PFA_std_reps_tst = np.std(PFAs_tst,axis=1)


PD_mean_reps_tst =
np.mean(PDs_tst,axis=1)

PD_std_reps_tst = np.std(PDs_tst,axis=1)


plt.figure(200)

plt.plot(PFA_mean_reps_tr,
PD_mean_reps_tr, 'b')

plt.plot(PFA_mean_reps_val,
PD_mean_reps_val, '--g')

plt.plot(PFA_mean_reps_tst,
PD_mean_reps_tst, '.-r')

plt.legend(('Tr','Val','Tst'))
```

```
plt.xlabel('$P_{FA}$')

plt.ylabel('$P_D$')


plt.figure(201)

plt.errorbar(PFA_mean_reps_tr,
PD_mean_reps_tr, yerr=[PD_std_reps_tr,
PD_std_reps_tr], xerr=[PFA_std_reps_tr,
PFA_std_reps_tr], fmt='b')

plt.errorbar(PFA_mean_reps_val,
PD_mean_reps_val, yerr=[PD_std_reps_val,
PD_std_reps_val], xerr=[PFA_std_reps_val,
PFA_std_reps_val], fmt='--g')

plt.errorbar(PFA_mean_reps_tst,
PD_mean_reps_tst, yerr=[PD_std_reps_tst,
PD_std_reps_tst], xerr=[PFA_std_reps_tst,
PFA_std_reps_tst], fmt='--g')

plt.legend(('Tr','Val','Tst'))

plt.xlabel('$P_{FA}$')

plt.ylabel('$P_D$')


plt.figure(202)

plt.errorbar(PFA_mean_reps_tr,
PD_mean_reps_tr, yerr=[PD_std_reps_tr,
PD_std_reps_tr], xerr=[PFA_std_reps_tr,
PFA_std_reps_tr], fmt='b')

plt.errorbar(PFA_mean_reps_val,
PD_mean_reps_val, yerr=[PD_std_reps_val,
PD_std_reps_val], xerr=[PFA_std_reps_val,
PFA_std_reps_val], fmt='--g')

plt.errorbar(PFA_mean_reps_tst,
PD_mean_reps_tst, yerr=[PD_std_reps_tst,
PD_std_reps_tst], xerr=[PFA_std_reps_tst,
PFA_std_reps_tst], fmt='--g')

plt.legend(('Tr','Val','Tst'))

plt.xlabel('$P_{FA}$')

plt.ylabel('$P_D$')

plt.axis([0, 0.05, 0.985, 1]) #ìzoom
```