UNIVERSITAT POLITÈCNICA DE VALÈNCIA

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

Implementación de un sistema de detección de intrusos en redes IP con Inteligencia Artificial

Mario Aragonés Lozano

Tutor: Alberto Albiol Colomer

Trabajo Fin de Grado presentado en la Escuela Técnica Superior de Ingenieros de Telecomunicación de la Universitat Politècnica de València, para la obtención del Título de Graduado en Ingeniería de Tecnologías y Servicios de Telecomunicación

Curso 2019-20

Valencia, 6 de Julio de 2020

Agradecimientos

Primeramente y en general, agradecer a todas las personas que alguna vez han coincidido y han compartido momentos conmigo puesto que toda experiencia, sea buena o mala, marca las futuras y, por tanto, marca la vida.

A mis amigos. Para empezar a los nuevos, los conocidos en este maravilloso grado. Todos los momentos que hemos pasado juntos. Por una parte los de sufrimiento. Aunque por otra, y más importante, los de alegría, aquellos en que las horas no pasaban y solamente había ganas de continuar. También a los de siempre, por estar a mi lado en todo momento fuese cual fuese la circunstancia.

A mis profesores, a todos y cada uno de ellos, por ofrecerme las herramientas que hoy me permiten poder entregar este trabajo. Mencionar al doctor Manuel Esteve por ofrecerse a ayudarme con sus conocimientos sobre ciberseguridad y recomendarme una pieza fundamental, los conjuntos de datos, de entre los cuales escogí el usado. Y sobre todo al doctor Alberto Albiol, no tengo forma de agradecerle todo lo que ha hecho por este proyecto. Desde un primer momento creyó en que yo estaba capacitado para realizarlo. Me ha ofrecido los recursos necesarios para desarrollar conocimiento sobre la materia y siempre ha tenido tiempo para resolver mis dudas aportando su experiencia en el *deep learning*. Sin él, este proyecto no hubiese sido posible. No solo eso, ha conseguido que el *deep learning* sea una pasión para mí y me ha permitido juntarla en este trabajo con mi otra gran pasión, las redes de comunicaciones.

Por último, pero no menos importante, a mi familia. Me gustaría destacar dos piezas clave de la misma. Pedro Miguel Romero, desde que era pequeño has convertido las telecomunicaciones en un sueño para mi. Siempre has estado dispuesto a resolver mis inquietudes, me has ofrecido recursos para que pudiese razonar el porque de las cosas y me has demostrado el funcionamiento de las telecomunicaciones en la práctica. Si no te hubiese conocido, no se si hoy estaría presentando este trabajo en esta escuela técnica superior de ingenieros. A mis padres, Manuel y Angelina, siempre me habéis apoyado en todas las decisiones que he ido tomando a lo largo de la vida y habéis tenido plena confianza en que estaba capacitado para llevarlas a cabo de forma satisfactoria. Me habéis ofrecido los recursos y la libertad para que pudiese realizar aquello que quisiese, no solo en materia de estudios, sino también de proyectos personales. Habéis estado junto a mi y me habéis animado cuando más lo necesitaba. Por celebrar los éxitos juntos. Pero sobre todo, y más importante, por inculcarme los valores que hoy me hacen ser la persona que soy, el valor del que más orgulloso estoy.

A todos ellos, GRACIAS.

Resumen

En unos tiempos donde cada vez se depende más de la tecnología en todos los aspectos (desde comprar hasta trabajar pasando por el contenido de ocio, consulta de información, …) la sociedad se enfrenta a un mundo donde a priori todo son facilidades y donde la falta de formación por la mayoría de sus usuarios es clara y abre un mundo a un nuevo tipo de delincuentes que no se tienen que mover de su domicilio para hacer daño, los ciberdelincuentes.

Hay gente que cree que estos son simples personas aburridas en su casa, no más lejos de la realidad, los más peligrosos viven de descubrir vulnerabilidades y diseñar nuevos ataques desconocidos por su víctima (los ataques de día cero), tienen un horario de trabajo normal y trabajan para empresas que se lo pueden permitir, con objetivos muy claros, como, por ejemplo, robar información sensible de la competencia para obtener beneficios.

Para protegerse, las empresas deben implementar servicios (hardware y software) que monitoricen el comportamiento de las comunicaciones y sean capaces de detectar un ataque e informar al departamento de seguridad para que éste pueda reaccionar cuanto antes. Hay distintos dispositivos que se deben tener en consideración, en este documento se detalla la implementación de un IDS (Sistema de detección de intrusos, de sus siglas en inglés). Primeramente, uno basado en firmas. Estos son capaces de detectar todos aquellos ataques cuyo comportamiento tienen previamente almacenado. A continuación, uno basado en anomalías. En este tipo de IDS específico, se modela cuál es el comportamiento normal del sistema y todo aquello que esté fuera de unos márgenes se considera un ataque. Con este tipo de IDS se pueden detectar ataques ya conocidos y, sobre todo, ataques de día cero, los más peligrosos y dañinos hoy en día.

Resum

En uns temps on cada volta es depèn més de la tecnologia en tots els aspectes (començant per anar a comprar i finalitzant per treballar passant pel contingut d'oci, consulta d'informació, ...) la societat s'enfronta a un món on a priori tot son facilitats i on la falta de formació per la majoria del seus usuaris es clara i obri on mon a un nou tipus de delinqüents que no s'han de moure del seu domicili per fer mal, els ciberdelinqüents.

Hi ha gent que creu que son simples persones avorrides en sa casa, no més lluny de la realitat, els mes perillosos viuen de descobrir vulnerabilitats i dissenyar nous atacs desconeguts per la seua víctima (el atacs de dia zero), tenen un horari de treball normal i treballen per a aquelles empreses que s'ho poden permetre, amb objectius com, per exemple, furtar informació de la competència.

Per a protegir-se, les empreses deuen implementar serveis (hardware i software) que realitzen la monitorització del comportament de les comunicacions i tinguen la capacitat de detectar un atac i informar al departament de seguretat per a que aquest puga reaccionar el més prompte possible. Hi ha distints dispositius que es deuen tindre en consideració, en aquest document es defineix la implementació d'un IDS (Sistema de detecció d'intrusos, de les seus sigles en anglès). En primer lloc, un basat en firmes. Aquestos son capaços de detectar tots aquells atacs que tinguen el comportament previament enmagatzemat. A continuació, un basat en anomalies. En aquest tipus d'IDS específic, es modela quin es el comportament normal del sistema i tot allò que es troba fora d'uns marges es considera un atac. Amb aquest tipus d'IDS es poden detectar atacs ja coneguts i, sobre tot, els atacs de dia zero, el més perillosos i els que més mal fan.

Abstract

In a time when we are increasingly dependent on technology in all aspects of life (from shopping to work to leisure to consulting data) society is facing a world where, a priori everything is easy, but where a lack of knowledge on the part of most users is clear and opens up the world to a new type of criminal who does not have to move from his home to do harm, the cybercriminal.

There are those who believe that the culprits are bored people at home, but this far from the truth. The most dangerous ones live from discovering vulnerabilities and designing new attacks which are unknown to their victims (zero-day attacks). They work nine to five for companies that can afford them. They have very clear objectives, like stealing sensitive information from competitors for profit.

To protect themselves, companies must implement services (hardware and software) that monitor communication behaviour and are able to detect an attack and inform the security department, which must then react as quickly as possible. There are a number of devices that must be taken into consideration. This document details the implementation of an IDS (Intrusion Detection System). Firstly, one based on signature. This type of device stores a limited amount of attack's behaviour which are used to decide whether the scanned traffic is a danger or not depending on matching both behavior. Then, one based on anomalies. In this specific type of IDS, the normal behaviour of the system is modelled and anything outside a certain range is considered to be an attack. With this type of IDS, it is possible to detect previously-known attacks and, above all, zero day attacks. These are the most dangerous nowadays, producing the most damage to the target if the attack is successfully completed.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

En los últimos años la sociedad está viviendo una revolución tecnológica sin precedentes. En poco menos de un par de décadas, se ha pasado de tener que consultar los números de teléfono en una guía telefónica a tener todos los números necesarios en cualquier bolsillo, las personas han pasado de estar prácticamente ilocalizables (a menos que se encontrasen en su lugar de trabajo o en su domicilio particular) a estar en todo momento pendientes de un dispositivo por si alguien quiere contactarles (es más, hay ciertas aplicaciones que hasta permiten a los usuarios compartir su ubicación en tiempo real), incluso se ha pasado de tener que pagar cantidades desorbitadas de dinero por una llamada transoceánica a que este tipo de llamadas se realicen con voz y video a través de una conexión a Internet y sin gastos adicionales.

Esta revolución ha implicado, también, un cambio en el estilo de vida en todas las edades. Se puede observar cómo los niños ya no disfrutan de un balón en un parque, prefieren estar jugando en frente de una pantalla. Se puede apreciar cómo gente mayor con cierta dificultad intenta hacer uso de las nuevas tecnologías, muchas veces sin éxito debido a las carencias de lectura y escritura por no haber ido a la escuela. Y sobre todo, cabe destacar cómo, anteriormente, era la gente mayor aquella que enseñaba a los jóvenes qué es aquello que debían hacer, en cambio, hoy en día cada vez se observa más cómo sucede al contrario, son esta gente de avanzada edad la que está siendo educada, tecnológicamente hablando, por la gente joven. No solo eso, hay muchos jóvenes que también ayudan a sus padres en temas de tecnología (por la falta de conocimientos de estos últimos) desde hacer uso de una *suite* ofimática hasta configurar un servicio de mensajería instantánea.

Que tantos usuarios dependan de gente sin formación para realizar las tareas cotidianas con la tecnología (debido a que los mismos no tienen idea alguna de como realizar dichos procedimientos y se fían en vez de aprender por expertos el como hacerlo) es un **gran problema**. Es un gran problema por el simple hecho de que esas mismas personas están creando una adicción a la tecnología, es decir, están creando una dependencia a algo que no saben como usar de forma correcta.

No solo ha cambiado la forma en que las personas viven, también la forma en que las personas **intentan hacer daño**. Hace un par de décadas, un individuo podía ser atracado por la calle, posiblemente la siguiente vez éste no iría por esa calle, o si fuese, lo haría con más cuidado almenos. Un individuo podía ser robado en casa, pero disponía de métodos como alarmas para dificultar la labor de los ladrones. Pero hoy en día, no es que esas acciones no continúen sucediendo, que suceden, lo que está pasando es que los delincuentes donde realmente hacen daño es de forma telemática, desde cualquier parte del mundo, usando aquellos sistemas donde se ha creado una

dependencia y siendo los más afectados aquellos que no tienen conocimiento del uso correcto de los mismos, ya que pueden ser atacados un número infinito de veces, por el hecho de que ni ellos saben cómo evitar el ataque, ni aquellos de los que han aprendido saben cómo evitarlo, por tanto, no pueden aplicar medidas para evitar que vuelva a suceder, permitiendo a los atacantes repetir la misma acción maligna con la certeza de que va a ser satisfactoria.

El problema no finaliza en un ciudadano y su domicilio particular, sino que se extiende a las empresas, ya que estas personas pueden ser trabajadores de las mismas y muchas veces tienen acceso a muchos más recursos de los necesarios para realizar su labor. Esto evidencia una falta de mecanismos de detección y prevención de ciberataques por parte del departamento de seguridad así como cursos de formación en buenas conductas tecnológicas a los trabajadores.

Este trabajo se elabora en medio de una pandemia mundial originada por el *SARS-CoV-2* (o *COVID-19*) que aún ha acentuado más este problema en las pequeñas, medianas y grandes empresas. Una de las medidas adoptadas por los gobiernos ha sido un confinamiento que ha *obligado* a las empresas a la implementación del teletrabajo para que los empleados pudiesen continuar ejerciendo la labor para la que han sido contratados sin tener que desplazarse a la empresa, es decir, desde sus propias casas. Muchas de estas empresas (por no decir la mayoría) no estaban preparadas para soportar tal reto tecnológico en cuanto a seguridad se refiere y ha propiciado un aumento en los ciber-ataques sobre éstas [1] [2] [3] [4]. Las grandes empresas se han dado cuenta de la necesidad de crear departamentos de ciberseguridad (o de crear una división dentro del departamento de seguridad) para prevenir y reaccionar de forma rápida ante uno de estos ataques. Las pequeñas y medianas empresas, por otro lado, han observado lo expuestas que están y con qué facilidad pueden perder toda la información almacenada (la cual solamente podrá ser recuperada pagando al ciberdelincuente).

Dentro de los dispositivos que deben ser implementados para poder detectar un ciberataque es un **Sistema de detección de intrusos** (o IDS de sus siglas en inglés *Intrusion Detection System*). Estos dispositivos se encuentran en los sistemas de seguridad de alta gama, es decir, tienen un precio elevado. Además, la correcta configuración de los mismos es compleja, lo que aún dificulta más la implementación de los mismos. Los gastos no finalizan en la compra del dispositivo, para poder tener las bases de datos al día y así poder detectar el mayor número de ataques posibles, se debe abonar una cuota mensual asociado al hardware adquirido (a mejor hardware, mayor coste mensual). La finalidad de éstos es la de informar al responsable de ciberseguridad sobre incidentes en el sistema de comunicaciones, con esta información, el mismo debe ser capaz de tomar decisiones sobre si es un ataque o no.

Existen distintos tipos de IDS. Los dos principales son:

- **IDS basado en firmas**: Son aquellos que buscan similitudes entre el comportamiento de los paquetes que están siendo analizados y el comportamiento de los ataques conocidos, etiquetados y almacenados en la base de datos (o firmas, de ahí su nombre).

- **IDS basado en anomalías**: Este tipo de IDS tiene una primera fase de modelado para tener conocimiento de cual es el comportamiento normal del sistema. A continuación, avisará de un ataque cuando el comportamiento que se está analizado exceda unos límites (configurados por el usuario) del comportamiento normal del sistema. Dependiendo del umbral establecido, avisará antes o después de un ataque. Son mejores por su capacidad de detección de nuevos ataques, pero pueden aportar falsos positivos.

1.1. Motivación

Una empresa, por pequeña que sea, envía y recibe una gran cantidad de paquetes por segundo de tráfico IP cuyo comportamiento debe ser procesado para poder detectar si este es el normal o es un ataque. Sin lugar a dudas, el *deep learning* está siendo cada vez más nombrado en publicaciones y congresos haciendo que su funcionamiento mejore día tras día. Estos modelos necesitan gran cantidad de datos para poder funcionar correctamente, coincidiendo con uno de los requisitos del problema a resolver, por tanto cabe preguntarse si se podría diseñar un sistema de detección de intrusos aplicando técnicas de inteligencia artificial.

Si fuese satisfactorio, se abriría un nuevo mundo en este tipo de dispositivos que, con *cloud computing* o incluso con hardware específico, se podría:

1. Abaratar costes
2. Facilitar el uso
3. Facilitar la implementación

Otra decisión a tomar mientras se planteaba el problema a resolver fue si trabajar a nivel de host (*layer* 7 en el modelo OSI) o trabajar a nivel de red (*layer* 4 en el modelo OSI). A pesar de ofrecer mayor dificultad a la hora de detectar cierto tipo de ataques (los cuales se caracterizan por modelar comportamientos normales y con la consecuencia de una posibilidad muy baja de detección) pero con la ventaja de reducir el número de elementos a añadir a la red, se ha decidido implementar un **Sistema de detección de intrusos a nivel de red** (NIDS de sus siglas en inglés *Network IDS*).

1.2. Objetivos

El objetivo de este proyecto es la implementación de un **sistema de detección de intrusos a nivel de red** para sistemas de comunicaciones que hacen uso del protocolo IP. Este se implementará haciendo uso de inteligencia artificial, más específicamente, *deep learning*.

Se busca conseguir un sistema fiable en la detección de intrusos. Este será implementado en infraestructuras de red.

Habrá dos implementaciones distintas acorde a los dos tipos anteriormente detallados.

Los datos de entrada al sistema encargado de procesarlos deberán cumplir

- Hasta nivel 4 en la capa OSI, es decir, hasta nivel de puerto de comunicaciones
- Preprocesados a nivel de conexión.
- Estarán o no etiquetados dependiendo del tipo de IDS a implementar.

1.3. Conjunto de datos

Tras realizar un análisis exhaustivo buscando el conjunto de datos que mejor permita obtener unos resultados fiables en cuanto a semejanza con un sistema real de comunicaciones, se decidió hacer uso de **CSE-CIC-IDS2018**[5], el cual ha sido generado por *Communications Security Establishment* y *Canadian Institute for Cybersecurity* en la plataforma *AWS*.

Se ha optado por este conjunto de datos debido a los motivos que se detallan a continuación:

- Es actual, año 2018. Los otros a valorar eran de años anteriores.
- Contiene una gran cantidad de datos, necesario para el *deep learning*.
- Los datos están muy bien etiquetados y son ofrecidos en plano y preprocesados. En este proyecto van a ser usados ya procesados.
- La cantidad de ataques distintos es muy amplia, ofreciendo casos que deberían ser detectados y casos que muy difícilmente van a poder serlo.

1.3.1. Tipos de ataques del conjunto de datos

Este conjunto de datos contiene los siguientes ataques:

1.3.1.1. Bruteforce

Ataques por fuerza bruta.

Consisten en probar todas las combinaciones posibles de un cierto recurso hasta conseguir la correcta.

Ejemplo: Ataque por fuerza bruta a contraseña: se comprueban todas las contraseñas (del juego de caracteres válido) hasta conseguir la correcta. Los servicios de seguridad deberían implementar contraseñas de doble factor y bloqueo de la cuenta al cabo de un cierto número de intentos para solucionar este problema.

1.3.1.2. DoS

Denegación de servicio.

Un atacante busca agotar los recursos de un cierto servicio.

Ejemplo: Realizar peticiones mal formadas a un cierto recurso para que solamente pueda atender dicha petición.

1.3.1.3. DDoS

Denegación de servicio distribuido.

Varios atacantes desde distintas localizaciones buscan agotar los recursos de un cierto servicio.

Ejemplo: Ataque por peticiones DNS del tipo TXT, donde la solicitud al servidor DNS tiene como campo de origen la del objetivo, consiguiendo saturar el ancho de banda del mismo.

1.3.1.4. Web

Ataques web.

Buscan vulnerabilidades en los códigos web para infectar al objetivo. Puede ser en el propio código fuente o en el conjunto de recursos solicitados desde el mismo.

Ejemplo: Ataques por sitio cruzado (o XSS) que permiten inyectar *scripts* al solicitar dichas páginas.

1.3.1.5. Infiltración

Ataques de infiltración.

El virus consigue infectar un terminal. A partir de dicho momento, busca monitorizar la red, encontrar los distintos dispositivos que la conforman e ir distribuyéndose por la misma hasta tenerla toda infectada. Una vez sus algoritmos le indican que ha conseguido dicho objetivo, actúa. En la mayoría de los casos, una vez ha finalizado su misión se autodestruye.

Es uno de los ataques que difícilmente se podrán detectar con certeza ya que están programados para:

1. Modelar el comportamiento normal
2. Usar recursos de *layer* 7 para ocultarse

1.3.1.6. Botnet

Ataques con redes de bots.

Se denomina bot a todo aquel dispositivo que ha sido infectado pero en vez de realizar un ataque esperan órdenes de un controlador. Estos dispositivos permanecen inactivos hasta que el servidor de control les envía el *script* a ejecutar. Son muy versátiles ya que pueden ejecutar cualquier comando y, si estos no consumen muchos recursos, pueden permanecer activos sin que el usuario note su existencia.

El objetivo más común son los dispositivos IoT (Internet of Things).

Ejemplo: Pueden ser usados para realizar ataques DDoS.

1.3.2. Otros ataques

Uno de los diseños a implementar, el basado en anomalías, debe ser capaz de alertar sobre ataques nuevos que no estén incluidos en el conjunto de datos. Si esto se cumple, se detectarán aquellos ataques clasificados como **Ataques de día cero** (o **Zero-day attacks** en inglés). Estos son los ataques más peligrosos hoy en día, realizados muchas veces a imagen y semejanza de la red que se pretende atacar.

Si se consigue que el IDS funcione correctamente, no solo se detectarán otros ataques desconocidos hasta la fecha, sino que además se podrá crear un modelo de los mismos, almacenarlos y compartirlos, de forma que los sistemas que trabajan con firmas también sean capaces de detectarlos.

1.4. Revisión del estado del arte

A continuación se define cuál es el estado actual del arte para el problema planteado y, más específicamente, para el conjunto de datos con el que trabajar escogido.

Tiendo en consideración la antigüedad de este conjunto de datos, existen distintos artículos donde se ha implementando un sistema de detección de intrusos y se ha hecho uso del mismo. A pesar de ello, son pobres en contenido y los métodos de trabajo y evaluación son distintos a los planteados en este proyecto.

Algunos de estos artículos podrían ser:

- [6]: Realizan una metodología de trabajo similar a la seguida en este proyecto, pero usan entrenamiento supervisado y profundizan en otro tipo de modelo. Además, ofrecen los resultados finales sin las gráficas de evolución de los mismos (debido a las herramientas empleadas).
- [7]: Se realiza un entrenamiento supervisado y los resultados no son exhaustivos debido a las herramientas empleadas para el análisis.

Como resultado del estado actual del arte, parece razonable ofrecer una alternativa en metodología de trabajo, tipo de entrenamiento y modelo de red empleado.

Capítulo 2

Deep Learning

2.1. Evolución

2.1.1. Inteligencia artificial

En las últimas décadas se ha estado explotando un campo de las ciencias de la computación conocido como Inteligencia Artificial. Este no es un campo que haya sido descubierto en la actualidad, según John McCarthy [8] a partir de 1950 ya se empezó a usar la expresión inteligencia artificial, aunque ha sido en estos últimos años cuando ésta ha tenido su auge gracias a ciertos factores que serán detallados a continuación.

La inteligencia artificial se podría definir como un conjunto de algoritmos que tienen como finalidad que un ordenador sea capaz de aprender patrones. Constan de unos datos de entrada y, si fuese necesario, unos datos de salida que definen a los de entrada.

Se podrían diferenciar tres tipos de inteligencia artificial,

1. Inteligencia artificial débil: Es aquella en la que se aprende una única tarea y su funcionamiento se restringe a la misma de forma que cualquier cosa fuera de ella no sabrá interpretarla correctamente. Un ejemplo sería el piloto automático de un avión.
2. Inteligencia artificial fuerte: Un ordenador seria capaz de comportarse como un humano, es decir, aprender, pensar y actuar en consecuencia. Podrían ser capaces de crear nuevas soluciones como si de un humano se tratase
3. Inteligencia artificial extrema: Un computador llegaría a ser más inteligente que un humano. Es hipotética.

A día de hoy, en cuanto a inteligencia artificial se refiere, solamente existe la inteligencia artificial débil. ¿Quiere decir que los ordenadores solamente pueden aprender una única tarea? No.

2.1.2. Machine learning

A partir del año 1980 y con la reinvención del método de propagación hacía atrás (o *backpropagation*), detallado en la sección 2.2, nace un subconjunto de la inteligencia artificial llamado *Machine learning* [9]. En este avance se hace uso de algoritmos de aprendizaje estadístico para que el sistema no solo sea capaz de repetir aquello que aprendió mientras se producía su entrenamiento, sino que además tenga la capacidad de modelar el comportamiento de los mismos a partir de un **conjunto de variables definidas por el diseñador** y cuyos valores son establecidos por los algoritmos en el entrenamiento del sistema.

El *machine learning*, por tanto, se caracteriza por tener una gran cantidad de parámetros a aprender. Para ser efectivo necesita una cantidad de datos mayor a la requerida por la Inteligencia artificial débil (con un conjunto de datos pequeño no llegaría a establecer correctamente los valores de las variables).

El *machine learning* trabaja en muchos casos con algoritmos de regresión, es decir, predice el valor de una variable continua a partir de un conjunto de variables de entrada. Un ejemplo sería la regresión lineal, donde se sitúan los datos en el espacio y se buscará trazar una linea que se aproxime al máximo a cada punto. Si se implementase este algoritmo con *machine learning*, se definiría el error del sistema para cada punto como la mínima distancia entre la linea y el punto. La intención es que la linea obtenida por el sistema atraviese (o se acerce el máximo posible a) todo los puntos, para ello se irá modificando la dirección de la linea hasta conseguir el objetivo.

Este modelo se centra en intentar imitar a las personas cuando realizan una toma de decisiones, más específicamente, intenta imitar el cerebro humano. Para ello, se hace uso de las llamadas **redes neuronales** para la implementación del mismo.

Las redes neuronales se basan en un elemento (o nodo) denominado neurona. Éste es agrupado en capas. Un conjunto de capas de neuronas forman una red neuronal.

El funcionamiento de este modelo se detalla a continuación. En cada neurona se almacena un valor. Cada neurona, en general, tiene conexiones con todas las neuronas de la capa anterior. Cada conexión entre dos neuronas tiene asociado un valor (o peso). Para obtener el valor almacenado en una neurona, se realiza el sumatorio de, el valor de la neurona de la capa anterior multiplicado por el peso de la conexión entre la neurona de la capa anterior y la neurona a la que calcular el valor, para todas las neuronas de la capa anterior (2.1). Una vez se ha definido el valor de la neurona, este será utilizado como entrada para las neuronas de la capa siguiente.

Cuando se está realizando el entrenamiento, el valor que se actualiza es el peso de las conexiones. Estos modifican su valor a partir de la información obtenida del método de propagación hacía atrás.

$$x_{(c,m)} = \sum_{n=0}^{N-1} x_{(c-1,n)} \cdot W_{(c-1,n)(c,m)} \tag{2.1}$$

Nota: Siendo x el valor almacenado en la neurona

Nota: Siendo W el peso de la conexión

Nota: Siendo (*Posición de la capa*,*Posición de la neurona*)

Nota: Siendo N el total de neuronas de una capa

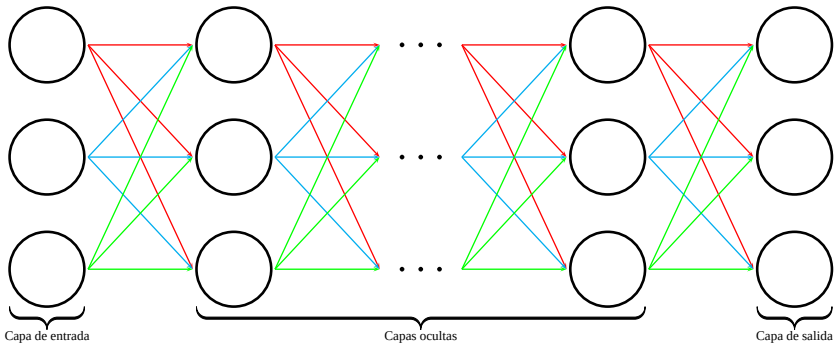


Figura 2.1: Red Neuronal: Clasificación de las capas

Las redes neuronales se pueden dividir en tres conjuntos de capas (ver figura 2.1):

1. Capa de entrada: Contiene los datos de entrada al sistema.
2. Capas ocultas: Capas intermedias donde se conforma la lógica encargada de aprender los patrones de comportamiento.
3. Capa de salida: Resultado del sistema a devolver.

2.1.3. Deep learning

En el año 2009, Fei-Fei Li (profesora de la universidad de Stanford) creó el conjunto de datos que revolucionaría el mundo de la inteligencia artificial, **Image Net**. Este enorme conjunto de datos permitió modificar el modelo de sistema de aprendizaje de los sistemas creando un nuevo subconjunto dentro del *machine learning* conocido como *deep learning*. En la actualidad se hace uso de *deep learning* para la solución de todo tipo de problemas.

Este modelo, a diferencia del *machine learning* tradicional, permite incrementar el número de capas que conforman la red, consiguiendo así trabajar con información de más alto nivel. Para obtener un resultado óptimo, hay dos requisitos que se deben cumplir

1. Gran cantidad de datos. Muy superior a la necesaria en *machine learning* tradicional
2. Capacidad de cómputo muy elevada.

Ambos problemas citados anteriormente no son un inconveniente para este proyecto por los siguientes motivos

- **Gran cantidad de datos**: El problema a solucionar parte de la premisa de que hay muchos datos por segundo a analizar.
- **Capacidad de cómputo**: Hoy en día se dispone de una gran capacidad de procesamiento en cualquier dispositivo. Incluso no teniéndola en local, se puede hacer uso de *cloud computing*.

2.2. Método de propagación hacía atrás

Anteriormente ha sido nombrado el método de propagación hacía atrás como el encargado de aportar la información para actualizar los pesos de las conexiones entre neuronas.

Dicho valor está formado a su vez por un conjunto de variables.

- Gradiente del error: Indica cuál es la variación del error obtenido respecto a los pesos.
- *Learning rate* o Tasa de aprendizaje. Indica la proporción del gradiente del error que se debe sustraer. Es un parámetro definido por el diseñador de la red neuronal.

La operación matemática a realizar sería la definida en la ecuación (2.2)

$$W = W_{\text{actual}} - (\text{Learning rate}) \cdot (\text{Gradiente del error}) \tag{2.2}$$

El **gradiente** del **error** con **respecto** a los **pesos actuales** de la red neuronal se encuentra definido en la ecuación (2.3).

$$\text{Gradiente del error} = \frac{\partial \text{Error}}{\partial W_{\text{actual}}} \tag{2.3}$$

Un símil de las actualizaciones de los pesos podría ser el proceso de actualización de los filtros adaptativos.

La elección de un valor óptimo de *learning rate* es una parte crucial a la hora de actualizar los pesos. Si este no es óptimo, se pueden dar los siguiente casos

- A mayor *learning rate*, más rápido convergerá la red a los valores óptimos, pero puede ser que esta se salte el valor óptimo y se quede oscilando, con la consecuencia de que no convergerá nunca.
- A menor *learning rate*, más lento convergerá, pero será más difícil que la red se salte el valor óptimo del peso. Si el valor es muy pequeño, la red no convergerá nunca.

Por tanto, hay que buscar cuál es el valor óptimo del *learning rate* y, en la mayoría de los casos, éste debe ir evolucionando a medida que se va entrenando la red.

A continuación se define con qué funciones se obtiene el error y cuál es el procedimiento para realizar dichos gradientes.

2.2.1. Obtención del error

Para obtener el error entre el valor devuelto por la red neuronal y el valor esperado, se hace uso de las llamadas funciones de coste (*loss function* en inglés). Dependiendo del tipo de variables del que se desea obtener el error, existen funciones de coste distintas.

2.2.1.1. Variables numéricas

2.2.1.1.1. MSE Error cuadrático medio.

Es la más conocida. Consiste en la suma de la distancia al cuadrado del resultado obtenido por la red y el esperado.

$$\text{MSE} = \frac{\sum_{m=1}^{n} (y_m - y_m^{predicted})^2}{n} \tag{2.4}$$

2.2.1.1.2. MAE Error absoluto medio.

Reside en la suma de las diferencias absolutas entre el resultado obtenido por la red y el esperado.

$$\text{MAE} = \frac{\sum_{m=1}^{n} |y_m - y_m^{predicted}|}{n} \tag{2.5}$$

2.2.1.1.3. Diferencias entre MSE y MAE No solamente hay que tener conocimiento de la existencia de dichas funciones, sino también de cuando se debe usar cada una.

Si solamente se analizase cuál es la función que tiene menor coste computacional, la elección sería el **MSE** ya que cuando se desease actualizar el valor de los pesos, si se trabajase con el MAE se debería realizar el gradiente de una función con valor absoluto.

Por otro lado, si el **conjunto** de datos con el que trabajar tiene una **gran cantidad de datos atípicos** (o *outliers*), es más recomendable hacer uso del **MAE**, ya que el MSE reaccionaría ofreciendo un peso mayor a los *outliers* cuando estos se actualicen, es decir, se centrará en reducir dicho caso atípico en vez de los ejemplos comunes, resultando en una reducción del rendimiento en general.

2.2.1.2. Variables booleanas

2.2.1.2.1. BCE Binary Cross Entropy Loss.

En castellano traducido como pérdida por incertidumbre cruzada binaria. Se calcula a partir de la siguiente ecuación

$$\text{BCE} = -\frac{1}{n} \sum_{m=1}^{n} y_m \cdot \log\left(p(y_m)\right) + (1 - y_m) \cdot \log\left(1 - p(y_m)\right) \tag{2.6}$$

Siendo $p(\alpha)$ la **probabilidad** predicha por la red de **obtener** un valor **uno** (o positivo).

2.2.2. Gradientes

Tal y como se ha definido anteriormente, para actualizar el valor de los pesos se hace uso de los gradientes del error con respecto a los pesos. Estos se podrían calcular de entrada a salida o viceversa. Debido a que el nombre de la sección es **método de propagación hacía atrás** cabe suponer que los gradientes se calculan de salida a entrada, la pregunta sería, ¿por qué?

Sabiendo que el valor de una cierta neurona solamente depende de los valores de la capa anterior, si se realiza el gradiente de salida a entrada, cuando se obtiene un cierto error, este se irá repartiendo por el conjunto de neuronas que han influido en dicho valor, ya que se propagará el error desde la última capa hasta la primera.

En consecuencia, se empieza por la capa de salida, la primera capa después de calcular el error, y se calculan los gradientes. A continuación se hará uso de la regla de la cadena. La siguiente capa contendrá los gradientes de la función de error más los gradientes de la última capa. Se continua así sucesivamente hasta llegar a la primera capa que contendrá los gradientes de la función de error más los gradientes de todas las capas ocultas. Una vez llegado a este punto, ya se ha calculado el gradiente de todas las neuronas y por tanto el error en todas ellas.

Cada vez que se calcula el gradiente, se obtiene cual es el valor que se debe sustraer al peso, es decir, en qué dirección se debe actualizar el peso de manera óptima. Cabe recordar que este no será finalmente el valor que será sustraído al peso actual, ya que viene condicionado por el *learning rate*.

2.3. Tareas de aprendizaje supervisado

Como se ha descrito anteriormente, el objetivo de las redes neuronales es aprender, ahora bien, ¿qué deben aprender? Según el problema a resolver se pueden diferenciar distintos tipos de tareas.

2.3.1. Clasificadores

Se denomina clasificadores a aquellos tipos de redes neuronales en los cuales ésta recibe unos ciertos datos como parámetros de entrada y una etiqueta que describe a los mismos.

La finalidad de este tipo de redes es posicionar los conjuntos en un espacio y ser capaz de ubicar en éste las fronteras entre las distintas etiquetas. Si es capaz de posicionar las fronteras de forma correcta, al llegar nuevos datos, la red los posicionará en el espacio y será capaz de etiquetarlos automáticamente.

NOTA: La **etiqueta** suele ser un valor numérico entero.

Una de las funciones de error que se hace uso en los clasificadores es la **Cross Entropy Loss**.

2.3.2. Regresión

La otra tarea de aprendizaje de las redes neuronales por tradición es la de calcular regresiones. Se denomina regresión al hecho de encontrar relaciones entre ciertas variables de entrada para encontrar una salida que satisface las mismas.

Ejemplo: Regresión por cuadrados mínimos.

La filosofía en las redes neuronales no es diferente. A partir de un conjunto de valores de entrada, la red debe ser capaz de encontrar las correlaciones para obtener un valor de salida, a priori, conocido en entrenamiento. Una vez la misma esté entrenada de forma correcta, cuando se le introduzca un valor que no forme parte del entrenamiento a la entrada de la red, esta devolverá un valor de salida que satisfaga las correlaciones de los valores con los que ha sido entrenada.

Un ejemplo de regresión aplicada a la redes neuronales podría ser el cálculo del precio óptimo de una vivienda a partir de los valores del contexto oportunos.

Algunas de las funciones de error que se pueden usar en la detección por regresión son **MSE** y **MAE**.

2.4. Métodos avanzados de deep learning

El conjunto de métodos que a continuación se detallan, se implementan con tareas de aprendizaje no supervisadas.

2.4.1. Autoencoder

Un *autoencoder* es una estructura de red neuronal que se basa en obtener una representación de menor dimensionalidad (es decir, más compacta) de los datos introducidos a la entrada y que después debe ser capaz, a partir de esta representación obtenida, volver a generar la entrada introducida. Tiene la restricción de que solo será capaz de regenerar aquellos conjuntos que se encuentren dentro del espacio que conforman los datos de entrenamiento (de forma que todos los datos que estén fuera obtendrán un error de reconstrucción muy elevado).

Este tipo de estructuras se basan en un efecto llamado **cuello de botella**. Siguiendo el símil de una botella cualquiera, el cuerpo de la misma tiene un diámetro mayor que el cuello (por donde sale el líquido almacenado dentro de la misma hacía fuera). En las redes neuronales se implementa un modelo que sigue el mismo comportamiento, a partir de una entrada con un gran número de dimensiones, se busca comprimirlas en unas pocas intentando no perder información (o almenos, perder el mínimo posible). Este proceso es llevado a cabo por los conocidos *encoders* (o codificadores). Para lograrlo, la red neuronal que forma el *encoder* debe aprender cuáles son las correlaciones de las variables de entrada a medida que avanzan por la misma y, éstas, quedarán almacenadas en el conjunto de parámetros aprendidos por la red neuronal. Una vez se ha obtenido esta representación intermedia con un número reducido de dimensiones, la red debe ser capaz de obtener otra vez la entrada original. Para ello, se implementará una estructura nombrada como decodificador (o *decoder*) cuya distribución de capas es un espejo del *encoder*, es decir, pasa de pocas dimensiones a muchas. Un esquema gráfico se encuentra en la figura 2.2.

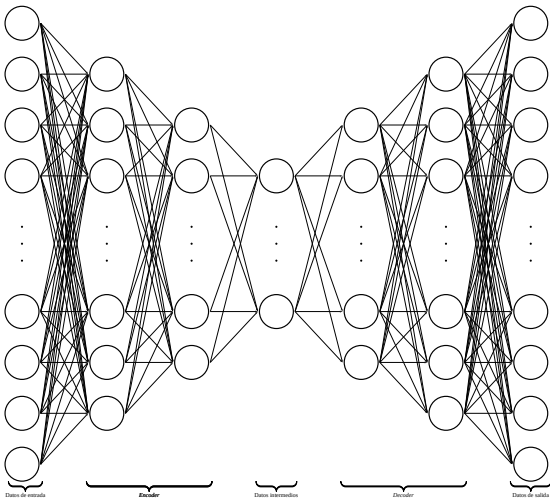


Figura 2.2: Autoencoder: Estructura

La importancia de este tipo de estructuras es que una vez ha sido preentrenada la red, introduciendo al *decoder* una entrada cuya distribución de los datos sea similar a la distribución de la salida del *encoder*, este debería ser capaz de generar un dato válido.

2.4.2. GAN

El modelo de red GAN (*Generative adversarial network* o Red generativa antagónica en castellano) pretende que una red consiga generar modelos parecidos a la entrada como pasa en el *autoencoder*, aunque la filosofía de entrenamiento es distinta.

En las redes de tipo GAN se busca crear una competición entre dos jugadores, en este caso dos redes neuronales, donde una de ellas debe ser capaz de generar datos con una distribución estadística lo más parecida a la que tiene la partición de entrenamiento a partir de ruido blanco gaussiano con media cero y varianza uno (la red llamada como generador) y la segunda red debe ser capaz de detectar si el dato procede del conjunto conocido o si este proviene de la red generadora de datos (esta red es llamada discriminador). Por tanto, la red del discriminador recibe entradas tanto del conjunto conocido como de la red del generador.

En consecuencia, se pueden diferenciar tres grandes bloques

1. Conjunto de datos conocido
2. Discriminador
3. Generador de datos

Una representación gráfica de la estructura del sistema se encuentra en la figura 2.3.

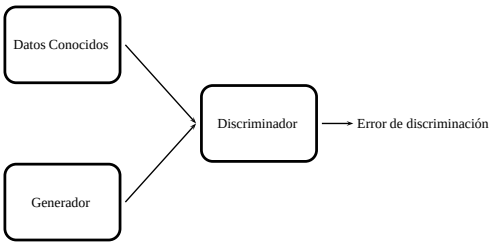


Figura 2.3: DGAN: Estructura

Con un correcto funcionamiento del sistema de redes generado, a medida que va avanzado el entrenamiento, la red del generador obtendrá como salida datos más parecidos a los de entrada, esto implica que la red del discriminador se deberá esforzar más para diferenciar entre datos válidos (los procedentes del conjunto de datos conocidos) y no válidos (procedentes del generador), es decir, se consiguen dos objetivos de forma simultánea:

- Obtener una red que sea capaz de generar datos con una distribución estadística lo más parecida a la que tiene la partición de entrenamiento, comportamiento similar al del *autoencoder*. Red del generador.
- Obtener una red que sea capaz de distinguir, de manera muy fina, si un dato pertenece al conjunto de entrenamiento o no. Red del discriminador

NOTA: Con un funcionamiento normal, la red del discriminador ofrecerá mayor error cuando el dato no sea del conjunto conocido.

2.4.2.1. Problemas

Debido a que se están actualizando dos redes neuronales de forma simultánea, se debe tener en cuenta que no tienen por qué converger ambas redes a la vez. Si esto pasase, implica que los gradientes de una red convergen más rápidos que los de la otra red, por tanto las actualizaciones de los pesos se vuelven inestables y el sistema diverge.

2.5. Otros tipos

A parte de los mencionados anteriormente, existen otros objetivos de aprendizaje para las redes neuronales.

2.5.1. Eliminación de ruido

En la grabación de un concierto se busca obtener el sonido más nítido de los instrumentos posible, una red neuronal podría eliminar todo aquello que se considerase ruido.

2.5.2. Procesamiento del lenguaje natural

El nuevo auge de las redes neuronales. Este tipo de redes es capaz de generar un fichero de texto a partir de unas palabras clave. También es capaz de relacionar conceptos, como si de un humano se tratase, a partir de la experiencia con la que ha sido entrenado, un comportamiento no esperado pero muy útil para futuras mejoras en este campo.

Capítulo 3

Conjuntos de datos y métricas de evaluación

3.1. Conjuntos de datos

Hay que realizar un paso previo con los datos antes de poder empezar a entrenar, para así, poder detectar (entre otras cosas) el llamado *overfit*. Este paso previo se podría nombrar como **creación de las particiones de datos**.

La creación de las particiones de datos consiste en la división del conjunto de datos conocido en dos bloques:

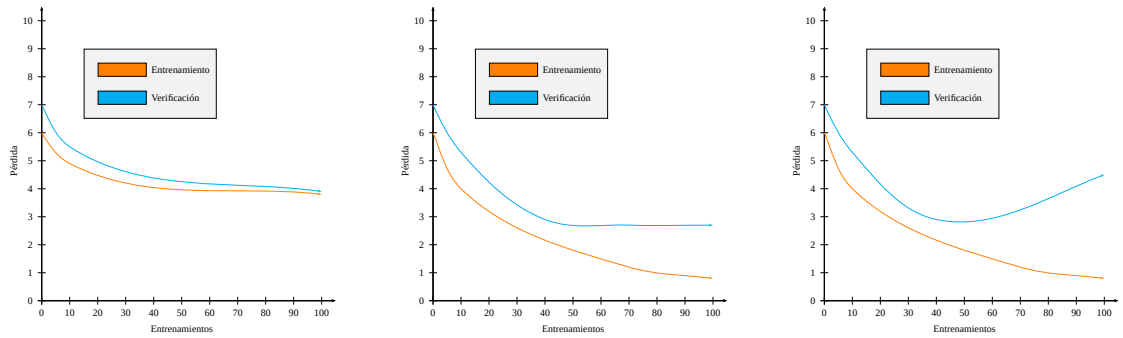
- Entrenamiento
- Verificación

A continuación se detallará cuáles son los criterios para realizar dicha división y por qué es necesario realizarla.

3.1.1. Entrenamiento

Los datos de entrenamiento serán los usados para, como se puede suponer, **entrenar a la red neuronal**. Cuando se esté usando este conjunto de datos, se realizará el método de propagación hacía atrás, es decir, se calcularán los gradientes y se actualizará el peso de las conexiones entre neuronas, por tanto, es una **etapa crítica**.

Es importante recordar que uno de los requisitos del *deep learning* es que necesita una **gran cantidad de datos para entrenar**, por tanto, este conjunto de datos debe ser extenso. Debido a este requerimiento, en la mayoría de los casos se destina el **80 %** del total del conjunto de datos conocido.



(a) *Underfitting* **(b) Correcto** **(c) *Overfitting***

Figura 3.1: Conjuntos de datos: Verificación: Evolución de la pérdida

3.1.2. Verificación

Para poder verificar que la red se está entrenando de forma satisfactoria, se debe destinar una parte del conjunto de datos conocido a este fin. Cuando se introducen estos datos, no se actualizan los pesos de las conexiones, solamente se obtiene cuál es el resultado de la red, por tanto, se está simulando cuál es el uso que se le da a una red neuronal ya entrenada y en producción.

Este conjunto está formado por aquellos datos que no han sido usados para el entrenamiento, en consecuencia, datos nuevos para la red.

Esta agrupación de datos tiene como finalidad detectar si la red se está entrenando de forma correcta o por el contrario hay algún problema, es decir, hay *underfitting* o *overfitting*. Para ello, se siguen las reglas cuya representacion gráfica se encuentra en la figura 3.1 y que a continuación se detallan,

- *Underfitting*: A medida que avanzan los entrenamientos, la **evolución de la pérdida** en la fase de **entrenamiento** y en la fase de **verificación** tienden a **decrementarse**.
- Comportamiento normal: A medida que avanzan los entrenamientos, la **evolución de la pérdida** en la fase de **entrenamiento** tiende a **decrementarse** y en la fase de **verificación**, a **estabilizarse**.
- *Overfitting*: A medida que avanzan los entrenamientos, la **evolución de la pérdida** en la fase de **entrenamiento** tiende a **decrementarse** y en la fase de **verificación**, a **incrementarse**.

3.1.3. Validación cruzada

Como se ha indicado anteriormente, el número de datos escogido para verificación es inferior al escogido para entrenamiento. Se podría dar el caso de que la red entrenase muy bien el comportamiento de los datos escogidos para verificación y que los datos con distinto comportamiento se quedasen todos en entrenamiento. Si esto sucediese, se podría llegar a considerar que una red es perfecta cuando en realidad los datos más difíciles se encuentran enmascarados dentro del conjunto de entrenamiento.

Para corregirlo, se deberían realizar tantos entrenamientos distintos como particiones pueda haber, de forma que todos los datos hayan sido usados en verificación almenos una vez (de hecho, deberían ser usados solamente una vez para así poder detectar mejor cual es el conjunto más complejo).

Si se hiciese una partición donde el 80 % de los datos se usan en entrenamiento, habría una quinta parte de los datos usados en verificación, por tanto, se debería repetir el entrenamiento cinco veces, teniendo en consideración que cada vez se usan unos datos distintos en verificación (así se ha conseguido que el 100 % de los datos se hayan usado una única vez en verificación).

3.2. Early Stop

Las fases de un entrenamiento no son estáticas, es decir, a medida que un entrenamiento progresa puede alternar entre distintas fases (en la práctica suele suceder así en la mayoría de los casos). Es importante detectar en qué punto sucede dicho cambio para poder tomar una decisión sobre si continuar el entrenamiento o finalizarlo. El ejemplo más claro es que un entrenamiento esté avanzando de forma correcta y llegue un punto donde pase a tener *overfitting*, en dicho punto se debería detener el entrenamiento.

Una técnica para la detección de dicho punto es el llamado *early stop* (o detención temprana). Tal y como se puede observar en la figura 3.1, el entrenamiento pasa a tener *overfitting* cuando la curva de la pérdida en verificación tiende a incrementarse. Siguiendo este hecho, esta técnica consiste en analizar la evolución de dicha curva y, cuando esta tienda a acrecentarse durante un número seguido de entrenamientos (también conocidos como paciencia), se detendrá el entrenamiento. Además, se debería devolver al usuario los pesos de la red neuronal en el punto donde se ha producido el cambio de comportamiento (es decir, en el punto óptimo), no los pesos finales (ya que estos ya tienen *overfitting*).

3.3. Tipos de aprendizaje

3.3.1. Aprendizaje Supervisado

Se define el aprendizaje supervisado como aquel en el que el conjunto de datos contiene tanto la información de entrada a la red neuronal como la información de salida esperada.

Cuando se realiza un entrenamiento con esta técnica, la red es capaz de aprender con certeza cuáles son las correlaciones entre los datos del mismo tipo, de esta forma, al introducir nuevos datos, se obtiene una gran precisión. Ofrece cuál es el límite de entrenamiento que se puede obtener para un determinado problema, a priori.

En contrapartida, no es capaz de detectar datos cuyas correlaciones no hayan sido aprendidas.

3.3.2. Aprendizaje No Supervisado

Se puede denominar aprendizaje no supervisado aquel en el que se conoce cuáles son los datos de entrada a la red pero no se dispone de ninguna información adicional sobre las categorías de los mismos. En este caso, se busca aprender cuál es la función de densidad de probabilidad de esta información introducida a la red neuronal.

Por tanto, para cada entrada, la red será capaz de obtener con que probabilidad ésta (la entrada) forma parte del conjunto de datos con el que ha sido entrenada, obteniendo mayor probabilidad cuando la entrada forme parte de la información aprendida y menor cuando no.

3.4. Técnicas de evaluación del entrenamiento

En primer lugar se debe asegurar que el entrenamiento de la red está mostrando un comportamiento satisfactorio.

3.4.1. Curva de pérdida

Cuando en secciones anteriores se explica cómo se comprueba si la red está evolucionando de forma correcta o hay algún problema en el entrenamiento (es decir, hay *underfitting* o *overfitting*), se hace referencia a la evolución de la pérdida a lo largo del entrenamiento (ver figura 3.1). La curva que se ha detallado y mostrado gráficamente es la llamada **curva de coste** o curva de pérdida.

Por tanto, se deduce que la finalidad de esta curva es poder analizar visualmente si el entrenamiento está progresando como se espera.

En estas curvas se representa cuál es el error medio obtenido en cada entrenamiento para las distintas fases (entrenamiento y verificación). Al ser el error medio, ofrece una idea general de cómo se comporta la red que se está entrenando, pero no es precisa a nivel de dato.

Para calcular el error medio, se realiza el sumatorio de todos los valores obtenidos por la función coste (detalladas en la sección 2.2.1) y se divide entre el número total de datos.

Precaución: Si hay una pequeña cantidad de datos en el entrenamiento que forman un comportamiento complejo, éste no será detectado con las curvas de pérdida, ya que al realizar la media se enmascarará (se ocultará) en el resto de datos que sí evolucionan de forma correcta.

3.5. Técnicas de evaluación de los resultados

Existen distintas técnicas para evaluar los resultados obtenidos, es decir, para poder comparar el resultado esperado (el conocido) con el que devuelve la red (el predicho). En los próximos puntos estas serán detalladas.

3.5.1. Precision

Podemos definir el *precision* o precisión **para un valor de umbral específico** como

$$\text{Precision} = \frac{\text{Verdaderos positivos}}{\text{Positivos predichos}} = \frac{\text{Verdaderos positivos}}{(\text{Verdaderos positivos}) + (\text{Falsos positivos})} \tag{3.1}$$

3.5.2. Accuracy

Podemos especificar el *accuracy* o exactitud **para un valor de umbral específico** como

$$\text{Exactitud} = \frac{\text{Número de predicciones correctas}}{\text{Número de predicciones totales}} \tag{3.2}$$

Es un valor que informa sobre cuántas predicciones se han realizado de forma satisfactoria sobre el total de las mismas, es decir, cuándo se cumple,

- Es positivo y se ha detectado positivo
- Es negativo y se ha detectado negativo

El opuesto, por tanto, sería cuántas predicciones han sido erróneas

$$1 - \text{Exactitud} = \frac{\text{Número de predicciones erróneas}}{\text{Número de predicciones totales}} \tag{3.3}$$

Es decir, cuándo se cumple,

- Es positivo y se ha detectado negativo
- Es negativo y se ha detectado positivo

3.5.3. F1 Score

El *F1 Score* o la puntuación F1 es una medida que ofrece información acerca de cuánto de preciso (del total de positivos **predichos**, cuántos son verdaderos) y exhaustivo (del total de positivos del **conjunto de datos**, cuántos han sido detectados) es un clasificador binario y **para un valor de umbral específico**.

Se puede detallar analíticamente como

$$\text{F1 Score} = 2 \cdot \frac{\text{Precision} \cdot \text{TPR}}{\text{Precision} + \text{TPR}} \tag{3.4}$$

NOTA: Siendo la exhaustividad representada por la TPR (Tasa de verdaderos positivos) y estando definida en la ecuación (3.5).

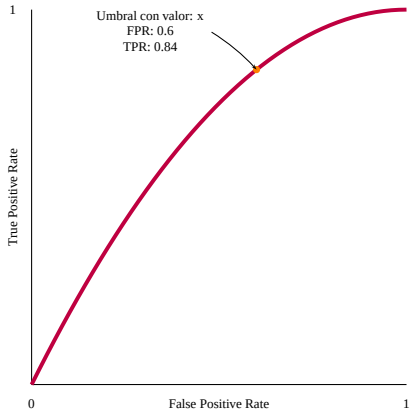


Figura 3.2: Curva ROC: Ejemplo

3.5.4. ROC

La curva ROC (o curva de características operativas del receptor) es una gráfica que permite evaluar cuál es el rendimiento de un clasificador binario. Ésta contiene los valores de la tasa de falsos positivos (FPR, definida en la ecuación (3.6)) en el eje X y los de la tasa de verdaderos positivos (TPR, definida en la ecuación (3.5)) en el eje Y.

A continuación se calculan un conjunto de umbrales empezando por un FPR y TPR igual a cero y finalizando en un FPR y TPR igual a uno. Para proseguir, estos son representados gráficamente, de forma que cada punto de la gráfica es un valor de umbral de clasificación que tiene asociado un FPR y TPR (valores que determinan su posición en la misma).

Un ejemplo de curva roc se puede observar en la figura 3.2

$$\text{TPR} = \frac{\text{Verdaderos positivos}}{\text{Total de positivos}} = \frac{\text{Verdaderos positivos}}{(\text{Verdaderos positivos}) + (\text{Falsos negativos})} \tag{3.5}$$

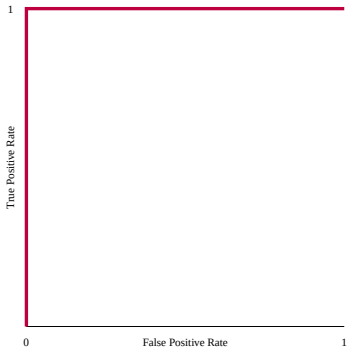
$$\text{FPR} = \frac{\text{Falsos positivos}}{\text{Total de negativos}} = \frac{\text{Falsos positivos}}{(\text{Falsos positivos}) + (\text{Verdaderos negativos})} \tag{3.6}$$

3.5.4.1. Información de la curva ROC

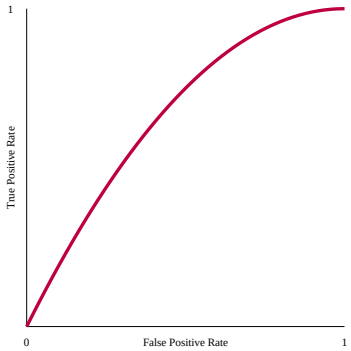
La curva ROC no solo informa si el rendimiento del clasificador binario es óptimo o debe ser mejorado, también es capaz de alertar sobre un diseño erróneo del mismo.

Debido a las definiciones de **TPR** y **FPR**, es fácil deducir que los valores inmejorables de los mismos serían:

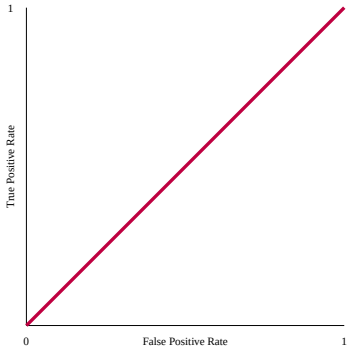
1. **Tasa de Verdaderos Positivos** (o TPR) = 1
2. **Tasa de Falsos Positivos** (o FPR) = 0



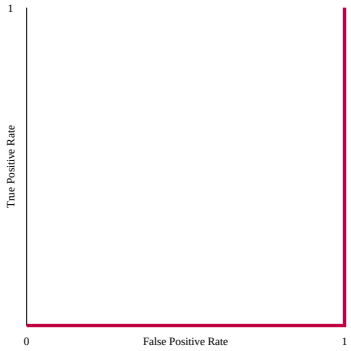
(a) Clasificador perfecto



(b) Clasificador funcionando correctamente



(c) Clasificador sin capacidad de clasificacion



(d) Clasificador con error en el diseño

Figura 3.3: Evaluación de la curva ROC: Resultados posibles

Tal y como se observa en la figura 3.3, se puede distinguir cuatro comportamientos distintos

- **Clasificador perfecto** (figura 3.3a). Esta curva ROC correspondería a un clasificador perfecto, es decir, a aquel que es capaz de etiquetar perfectamente el 100 % de los datos. Es un caso ideal, en la práctica si hay que tener en cuenta una cierta pérdida.
- **Clasificador normal** (figura 3.3b). Cuando se está realizando un entrenamiento, se espera una curva parecida a esta. Se deben realizar modificaciones en el diseño para conseguir que se aproxime todo lo posible a la curva del clasificador perfecto.
- **Clasificador malo** (figura 3.3c). Un clasificador con dicha curva ROC no tiene capacidad de clasificación. Esta curva informa de que hay la misma posibilidad de obtener un valor correcto que la de uno erróneo. Este clasificador, por tanto, no tendría validez alguna.
- **Error en el diseño** (figura 3.3d). Cuando el clasificador binario ha sido diseñado, se han cometido errores, ya que este clasifica los valores de forma inversa a la esperada. Habría que revisar el diseño de clasificador y corregirlo para que clasifique los valores de forma correcta.

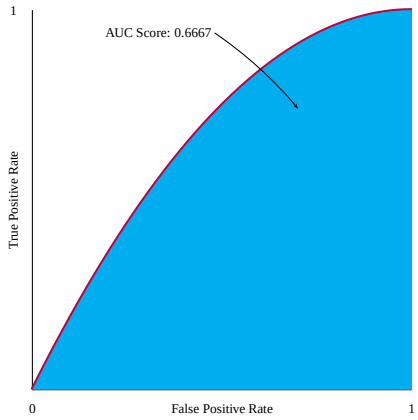


Figura 3.4: Evaluación de la curva ROC: AUC (Área bajo la curva)

3.5.4.2. Evaluación de la curva ROC

La curva ROC no solamente ofrece una forma visual de análisis del comportamiento del clasificador binario a analizar, también puede ser usada para definir dicho comportamiento con valores numéricos.

Como se puede apreciar, al ubicar el conjunto de umbrales sobre la gráfica llegan a formar una curva. Por tanto, la forma de evaluar numéricamente la curva ROC es con el conocido *AUC Score*, es decir, el valor del área bajo la curva. Una representación gráfica de dicho valor se puede observar en la figura 3.4.

Los ejes de la curva ROC comprenden entre el valor cero y uno (ambos inclusive), en consecuencia, el valor mínimo del *AUC Score* será cero y el máximo, uno.

Al igual que anteriormente se han definido una serie de casos de interés, en cuanto a posibles curvas ROC se refiere (figura 3.3), estos también tienen valores numéricos asociados, los cuales son,

- **Clasificador perfecto** (figura 3.3a): 1
- **Clasificador normal** (figura 3.3b): Un valor entre 0.5 y 1
- **Clasificador malo** (figura 3.3c): 0.5
- **Error en el diseño** (figura 3.3d): Un valor entre 0 y 0.5

	Descripción	AUC Score
Clasificador perfecto	Esta curva indica que el clasificador es capaz de distinguir a la perfección todas las etiquetas.	1
Clasificador normal	Una curva esperada cuando se realiza una evaluación.	Entre 0.5 y 1
Clasificador malo	El clasificador no tiene capacidad de clasificación.	0.5
Error en el diseño	Hay un error al diseñar el clasificador, devuelve el valor de los resultados al revés.	Entre 0 y 0.5

Tabla 3.1: Curva ROC: Resumen

3.5.4.3. Resumen

Resumiendo, la curva ROC es una herramienta para analizar los valores obtenidos por un clasificador binario de forma visual y numérica. Informa no solo de si este está funcionando como se esperaba, sino que también de si hay un posible error en el diseño. En la tabla 3.1 se encuentran los valores a destacar de la misma relacionados con la figura 3.3.

Capítulo 4

Herramientas

4.1. IDE

4.1.1. Microsoft Visual Studio Code

Para realizar la programación de los distintos códigos que van a formar las redes neuronales y el conjunto de instrucciones para entrenar y supervisar, se ha usado el entorno de desarrollo integrado creado por Microsoft llamado **Microsoft Visual Studio Code**.

Este software permite trabajar con gran variedad de lenguajes, entre ellos Python. Se ha elegido este IDE por

- Uso gratuito
- Integración con Git
- Integración con Notebooks de Python
- Permite la depuración del código en Python

4.2. Lenguaje de programación

4.2.1. Python

Entre los distintos lenguajes de programación disponibles a usar, se ha escogido Python. Es un lenguaje de programación gratuito, multiplataforma, compila en ejecución y ofrece una gran variedad de paquetes (o clases) programadas por terceros para trabajar con *deep learning*

4.3. Paquetes

4.3.1. Conda

Cuando se trabaja con una gran cantidad de paquetes no es viable instalarlos todos uno a uno cada vez que se cambia de computador, en vez de eso, se ha optado por hacer uso de Conda. Con este sistema, se crea un entorno donde se instalan los paquetes a usar, por tanto, si en algún momento se debe cambiar de computador donde realizar la ejecución de los códigos, basta con exportar el entorno y cargarlo en el nuevo computador, Conda automáticamente descargará todos los paquetes y dependencias necesarios para que todo funcione de forma correcta.

Se ha hecho uso de Conda en vez de Docker ya que Conda solamente instala paquetes mientras que Docker virtualiza un sistema operativo entero, algo no disponible en muchos *clusters* por seguridad.

4.3.2. Numpy

Numpy es un objeto (o clase) de Python que facilita el trabajo con vectores y matrices permitiendo, entre otros, realizar concatenaciones en dimensiones específicas, calcular medias, etc.

4.3.3. Pandas

Pandas es un objeto (o clase) de Python que permite trabajar como si de una base de datos se tratase. Permite, entre otras cosas, realizar búsquedas y filtros por columnas, contar cantidad de elementos, etc.

Además, contiene un método que permite crear un nuevo objeto leyendo los datos de un archivo de datos separados por coma (o CSV).

4.3.4. Scikit-learn

Scikit-learn es un paquete que contiene distintos objetos (o clases) de Python. En este proyecto ha sido usado para

1. Verificar el estado del arte
2. Procesar resultados de las redes neuronales para así poder evaluarlos

4.3.5. Pytorch

Pytorch es una librería de código abierto desarrollada por el departamento de Inteligencia Artificial de Facebook. Ofrece los recursos necesarios para trabjar con *Machine Learning* y *Deep Learning*.

Hay una amplia variedad dependiendo del objetivo buscado. Algunos de estos recursos ofrecen,

- Módulos para realizar la Propagación hacía atrás, es decir, calcular los gradientes y actualizar los pesos
- Módulos con gran cantidad de funciones de error preprogramadas. Además, permiten crear funciones de error propias al usuario.
- Módulos para crear redes neuronales
- Módulos para alimentar las redes. Hay opciones para permitir mezclar los datos, separarlos en conjuntos más pequeños, etc.

Además, también ofrece trabajar con aceleración por GPU si el sistema contiene componentes compatibles con CUDA.

4.4. Control de versiones

4.4.1. Git

Con la intención de tener un histórico de las distintas versiones por las que han ido evolucionando los archivos, se ha visto conveniente hacer uso de un sistema de control de versiones.

En este caso, se ha hecho uso de Git. Fue diseñado por Linus Torvalds (creador del kernel original de Linux) y es gratuito y de código abierto.

Con este sistema, se pueden revisar archivos de versiones pasadas por si se necesitase recuperar información eliminada.

Igualmente, puede ser usado haciendo uso de un terminal o con una gran cantidad de software que tiene integrado dicho sistema, entre ellos, el IDE empleado.

Capítulo 5

Métodos supervisados para la detección de intrusiones

El objetivo de la red neuronal a generar en este proyecto es identificar si los datos de entrada a la misma forman parte del comportamiento normal o son un ataque. En otras palabras, a partir de una entrada, la salida debe ser una etiqueta que identifique la entrada.

En primer lugar, se va a implementar un sistema de detección de intrusos basado en firmas. En este sistema en particular, se conoce el conjunto de posibles ataques a detectar, más específicamente, su comportamiento (o firma). Dado que se conoce cual es la entrada y también la salida, la mejor forma de implementarlo es con un entrenamiento supervisado.

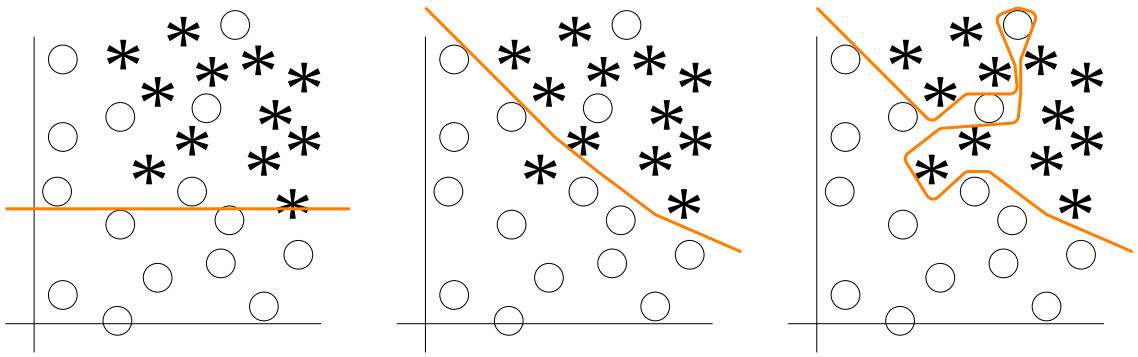
Como se ha descrito anteriormente, la tarea de aprendizaje supervisada que tiene este objetivo es la definida con los llamados clasificadores.

En esta tarea de aprendizaje, los datos se sitúan en un espacio y las redes intentan aprender dónde se encuentran las fronteras (en el propio espacio) entre los datos con distintas etiquetas. Para los próximos desarrollos, se va a suponer que el espacio es de dos dimensiones, aunque no tendría por qué ser así.

Un **clasificador perfecto** sería aquel que es capaz de **separar correctamente el 100 %** de los casos. Raramente eso sucede, siempre suele haber algún caso que no se consigue etiquetar de forma correcta. El **objetivo** final de una red bien diseñada es que la cantidad de **datos clasificados erróneamente** sea lo más cercano al **0 %** posible.

5.1. Posibles problemas

Cuando un clasificador está siendo entrenado, éste se enfrenta a dos principales problemas que deben ser detectados por el diseñador de la red neuronal, el *underfitting* y el *overfitting* (o sobre entrenamiento).



(a) *Underfitting* **(b) Correcto** **(c) *Overfitting***

Figura 5.1: Frontera aprendida por la red neuronal de un clasificador en distintas casos

En la figura 5.1 se encuentra una representación gráfica de como se situarían los datos en el espacio y las fronteras entre los mismos aprendidas por la red neuronal. Hay dos tipos de datos distintos, unos representados con círculos y los otros con asteriscos. Se representan las fronteras para los tres posibles casos,

1. *underfitting*: Figura 5.1a
2. Comportamiento esperado de una red bien diseñada: Figura 5.1b
3. *overfitting*: Figura 5.1c

5.1.1. Underfitting

El *underfitting* se produce cuando la red neuronal implementada converge pero no tiene la capacidad suficiente para aprender las correlaciones de los datos. Es decir, la red es demasiado simple para el problema que debe aprender.

La solución para aportar más capacidad a la red (es decir, incrementar la flexibilidad de la misma) consiste en aumentar el número de parámetros que la conforman. Para conseguirlo, se puede incrementar el número de capas, el número de neuronas por capa o ambas.

Otra solución, si se pudiese, residiría en reducir la complejidad de los datos.

5.1.2. Overfitting

El *overfitting* o sobreentrenamiento se produce cuando la red consigue aprenderse el conjunto de datos en lugar de las correlaciones entre los mismos. Al producirse este efecto, el error obtenido para dicho conjunto será muy pequeño, pero al introducir nuevos datos, la red no será capaz de clarificarlos, por tanto, no se cumple el objetivo buscado al implementar dicha red (ya que se busca que la red sea capaz de etiquetar correctamente datos completamente nuevos).

Para detectar este problema. se debe realizar una preparación previa de los datos antes de empezar a entrenar la red, tal y cómo se ha detallado anteriormente.

5.2. Etapa previa

Tal y como se ha descrito en la sección 3.1, la primera etapa (previa a poder entrenar) consiste en preparar el conjunto de datos, es decir, realizar la división en conjunto de entrenamiento y de verificación.

El conjunto de datos escogido en este proyecto [5] está formado por una agrupación de ficheros de texto con valores separados por comas (o CSV). En cada uno de estos ficheros, se puede encontrar datos con

- Comportamiento normal del sistema.
- Un tipo de ataque distinto.

Al tener los ataques separados por ficheros, permite de forma rápida y sencilla la evaluación del clasificador implementado para un tipo de ataque específico, ya que basta con cargar el fichero correspondiente al tipo de ataque a analizar.

Cabe recordar que este proyecto busca implementar un clasificador que sea capaz de distinguir cualquier tipo de ataque, conocido o desconocido, en consecuencia, se debe trabajar con el conjunto entero de los datos sin diferenciar tipos de ataques, es decir, entrenar con todos los ficheros a la vez.

Además, tal y como se ha mencionado anteriormente, cada fichero contiene un tipo de ataque distinto. Si se cargasen todos los ficheros en una única matriz y se realizase la división en entrenamiento y verificación, con una posibilidad muy elevada, habría ficheros enteros que solamente se encontrasen en el conjunto de entrenamiento y otros solamente en el conjunto de verificación, por tanto, los resultados obtenidos no serían válidos. Este hecho implica tener que separar el contenido de los ficheros cuando son cargados en el sistema y, en vez de crear una gran matriz con todos los datos, introducir directamente los mismos en el conjunto correspondiente (entrenamiento o verificación). No solo eso, cabe recordar el funcionamiento de la validación cruzada (subsección 3.1.3). Si se cargasen los datos tal y como se ha mencionado anteriormente, podría haber ciertos conjuntos de verificación donde hubiese tipos de ataque que no apareciesen porque están todos en entrenamiento, por tanto, se debería asegurar que siempre hay una cantidad (aunque pequeña) de todos los ataques en verificación. Para ello, se debe realizar la separación de los datos al ser cargados, pero además, teniendo en cuenta que se deben cumplir los requisitos de la validación cruzada. Por tanto, se conseguiría finalmente que en todos los conjuntos de verificación exista información de todos los ataques (en mayor o menor medida).

Nota: Debido a que no va a haber siempre la misma cantidad de datos sobre un ataque en particular en todos los conjuntos de verificación, es posible que ciertos análisis puedan indicar peores resultados debido a que la cantidad de comportamientos complejos es mayor en verificación y menor en entrenamiento.

En este proyecto se ha decidido realizar una separación de los datos consistente en un **80 % para entrenamiento** y un **20 % para verificación**. Además, como hay una quinta parte de los datos destinados a verificación, para cada modelo, se han realizado cinco entrenamientos con datos de verificación distintos, consiguiendo que el 100 % de los datos hayan estado una vez en verificación.

Además, para dificultar la tarea de entrenamiento y como no se ha tenido en consideración la dimensión temporal de los datos, se ha realizado un mezclado de los mismos, es decir, no siguen el orden que presentan en el fichero de datos original.

La preparación de los datos no finaliza en la separación de los mismos, en este proyecto. Al analizar las columnas que conforman el conjunto de datos original, se ha decidido realizar modificaciones en ciertas de ellas para así conseguir un resultado mejor y más lógico.

Primeramente, existe un columna definida como *Dst. Port* que hace referencia al puerto destino de la conexión. La información de la misma está representada con **valores enteros**. Si se introdujese dicha información a la red neuronal tal y como se representa en su formato original, ésta aprendería los valores estadísticos de la misma (es decir, su media, varianza, etc.), una información no válida ya que un puerto destino tiene un valor definido por el protocolo o por la aplicación usada. En vez de ello, se ha decidido separar dicha información en un conjunto de columnas con **variables booleanas** que indican la existencia, o no, del puerto, o conjunto de puertos, a los que hacen referencia. Se ha realizado una selección de los mismo los cuales son detallados a continuación

- Port 0
- Port 80: Puerto HTTP
- Port 53: Puerto DNS
- Port 443: Puerto SSL
- Port 3389: Puerto RDP (Windows Remote Desktop Protocol)
- Port 21: Puerto Telnet
- Port 8080: Puerto HTTP
- Port 445: Puerto del Active Directory de Windows
- Port 22: Puerto del SSH
- Port 3306: Puerto del MySQL
- Port 5060: Protocolo SIP (Protocolo de inicio de sesion usado por VoIP)
- Port Common: Cualquier valor de puerto menor de 1024 y no citado anteriormente
- Port Not Common: Cualquier valor de puerto mayor o igual de 1024 y no citado anteriormente

Asimismo, otra columna que ha sido modificada es la nombrada como *Labels* que detalla cuál es la etiqueta asociada a una conexión. Esta viene definida con **variables de tipo texto**, un tipo no válido para una red neuronal. Dado que en este conjunto de datos hay una cantidad finita de las mismas, se ha asociado un **valor entero** a cada uno de los textos, cuya relación se detalla a continuación

0. Benign
1. FTP-BruteForce
2. SSH-Bruteforce
3. DoS attacks-GoldenEye
4. DoS attacks-Slowloris
5. DoS attacks-SlowHTTPTest
6. DoS attacks-Hulk
7. DDOS attack-LOIC-UDP
8. DDOS attack-HOIC
9. Brute Force -Web
10. Brute Force -XSS
11. SQL Injection
12. Infilteration
13. Bot

En el punto actual ya se ha divido el total de los datos conocidos en dos conjuntos, entrenamiento y verificación, además, se han modificado aquellos tipos de datos que se ha considerado oportunos para mejorar el funcionamiento, a pesar de ello, todavía no se ha finalizado el proceso de preparación de los datos.

Para incrementar la dificultad de los datos, el clasificador no debe tener conocimiento del tipo de ataque, solamente de si es un ataque o no lo es. Por tanto, el paso final de la preparación consistiría en agrupar todos los ataques en una única etiqueta.

Cuando se trabaja con la librería Pytorch, ésta ofrece un objeto encargado de gestionar el conjunto de datos, llamado *torch.utils.Dataset*. Para tener el control de la gestión de los datos un diseñador en vez de los valores definidos en la librería, éste debe crear un objeto propio que herede [1] del anteriormente mencionado.

[1] En programación se llama herencia cuando se desea que un determinado objeto (el que hereda) obtenga todos los métodos ya programados de otro objeto (el heredado). Si los métodos que contiene el objeto original no se encuentran en el nuevo objeto, se usarán los ya existentes. Si hay un método del objeto original en el nuevo objeto, se usará el del nuevo objeto.

Uno de los métodos que contiene dicho objeto está definido como *__getitem__* y es el encargado de devolver al programa de entrenamiento los datos de las posiciones que se solicitan como parámetro de entrada. Este ha sido el método que se necesitaba sobreescribir. Cuando se solicite un cierto dato, el método devolverá un diccionario [2] que contiene la siguiente información

- **data_bool**: Conjunto de valores booleanos solicitados
- **data_numeric**: Conjunto de valores numéricos solicitados
- **labels**: Etiquetas con clasificación binaria
    0. Comportamiento normal
    1. Ataque
- **labels_class**: Etiquetas con clasificación entera (contiene la información de los tipos de ataques).

Alguien podría preguntarse por qué se realiza este procedimiento cuando se solicitan los datos por parte del programa de entrenamiento y no cuando se transforman estos datos (las etiquetas) de un valor de texto a uno entero. No se realizan en dicho punto debido a que esa información conviene almacenarla por si se desea aislar el comportamiento del clasificador para un tipo de ataque específico o un conjunto de los mismos. Por tanto, en el propio programa de entrenamiento y verificación se hace uso de todas las variables anteriormente detalladas.

- **labels**: Para realizar el entrenamiento.
- **labels_class**: Para almacenar los resultados.

Asimismo, es apreciable cómo se separan las variables por tipo, es decir, en booleanos y en enteros. Como se detalla en la subsección 2.2.1, dependiendo del tipo de variable se deben usar unas funciones de coste u otras, por tanto, realizando dicha separación previa se ahorra coste computacional después. Además, al generar las redes neuronales encargadas de la clasificación, también se hará uso de dicha división.

5.3. Verificación del estado del arte

Al plantear el procedimiento a seguir para la realización del proyecto, se consideró oportuno empezar por el análisis del estado del arte. Realizando esta primera etapa se consigue validar los datos reportados en dichos artículos y permite asegurar que las transformaciones realizadas al conjunto de datos han sido satisfactorias y no han perjudicado a los resultados.

En este paso, a pesar de usar *Scikit-learn* en vez de *Pytorch*, también se han aplicado las técnicas de transformación de los datos detalladas anteriormente y se ha realizado una clasificación binaria (siendo un valor de **comportamiento normal** etiquetado con el valor **cero** y uno de **ataque**, con el valor **uno**).

[2] En Python, un diccionario es un tipo de variable que contiene un conjunto de claves (o identificadores) y, para cada una de ellas, una valor asociado

Se han seguido las técnicas detalladas en el propio estado del arte, usando el conjunto de métodos que ofrece *Scikit-learn* para el manejo de los estados, el entrenamiento de distintos modelos y el análisis de los resultados.

También se ha entrenado siguiendo los requisitos de la validación cruzada.

5.3.1. Resultados

Primeramente se ha entrenado cada ataque por separado y, a continuación, se han agrupado todos para realizar un entrenamiento no supervisado.

Cada uno de estos entrenamientos, se ha realizado para los modelos proporcionados por *Scikit-learn* que se detallan a continuación

- Perceptron: Clasificador Perceptron mono capa. Es un clasificador lineal de tipo percetron optimizado por SGD (Descenso por Gradiente Estocástico, es decir, no determinista).
- SGD: Clasificador lineal tipo máquina vector de soporte optimizado por SGD.
- MLP: Clasificador Perceptron multi capa

Una vez ha finalizado el entrenamiento, cada uno de esos modelos ha sido analizado a nivel de dato por las funciones coste disponibles en *Scikit-learn* que se enumeran a continuación

- **Acc.**: Accuracy: Exactitud. Número de predicciones correctas respecto del total.
- **F1 We**: F1 Score - Weighted: Se calculan las métricas para cada etiqueta y se obtiene la media balanceada dependiendo del número de positivos para cada una de ellas.
- **F1 Ma**: F1 Score - Macro: Se calcula la métrica para cada etiqueta y se almacena su media no ponderada.
- **F1 Bi**: F1 Score - Binary: Se calcula la métrica binaria.

Finalmente, se ha realizado un promedio de los resultados obtenidos por los cinco entrenamientos que forman parte de la validación cruzada. Se han resumido los resultados obtenidos en la tabla 5.1, donde la primera parte representa los ataques de forma individualizada y la segunda, una simulación de entrenamiento no supervisado.

Al analizar dichos resultados, es fácil deducir que el ataque más difícil de detectar es el de *infiltration*, un hecho esperado por los motivos detallados anteriormente.

	Perceptron				SGD				MLP			
	Acc.	F1 We	F1 Ma	F1 Bi	Acc.	F1 We	F1 Ma	F1 Bi	Acc.	F1 We	F1 Ma	F1 Bi
BruteForce	1.0	1.0	0.91	1.0	1.0	1.0	0.78	1.0	0.99	0.99	0.83	0.99
Dos	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.99	0.99	0.90	0.99
DDoS	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.93	0.90	0.87	0.79
Infiltration	0.65	0.67	0.485	0.76	0.81	0.74	0.46	0.89	0.83	0.76	0.47	0.91
Bot	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.99	0.99	0.99	1.0
All	0.95	0.95	0.94	0.97	0.97	0.97	0.96	0.98	0.83	0.81	0.74	0.89

Tabla 5.1: Métodos Supervisados: Verificación del estado del arte: Resultados

5.4. Clasificador

Se va a hacer uso de la libreria Pytorch y el conjunto de pasos para la preparación previa de los datos.

En esta etapa, se va a incluir en el **conjunto de entrenamiento** datos con **comportamiento normal** y datos con comportamiento de **ataques** (estos últimos no van a estar clasificados, es decir, la red no va a tener información de qué tipo de ataque es, solamente conocerá que es un ataque).

La red neuronal a diseñar va a ser identificada como **Discriminador**. La función de esta red va a ser que, a partir de una entrada, decidir si los datos forman parte del comportamiento normal del sistema o, por el contrario, son un ataque. Es decir, como su nombre indica, es la encargada de discriminar el tipo de comportamiento que tiene la entrada.

Para poder conseguir un diseño óptimo, se deben realizar distintas pruebas e ir ajustando el modelo a partir de los resultados obtenidos en cada una de éstas. En cada una de las correcciones, se ha modificado el número de neuronas por capa, número de capas, funciones de coste, parámetros de dichas funciones, el valor del *learning rate* así como se ha introducido o no las normalizaciones de batch en distintos puntos de las redes neuronales y se han introducido o no distintos tipos de funciones de activación.

Se han realizado más de 80 entrenamientos donde cada uno tiene una duración mínima de dos días por la cantidad de datos que debe procesar y puede llegar a durar semanas. No es solo la duración del entrenamiento, a continuación se debe generar el conjunto de gráficas para poder examinar los resultados y analizar las mismas para ver si los datos obtenidos son los esperados o por el contrario hay que realizar modificaciones en el diseño así como detectar cual es el error en éste y aportar una solución para corregirlo.

El fruto de este conjunto de ensayos no ha sido uno, sino dos diseños, los cuales, se detallan a continuación explicando el por qué pueden ser válidos.

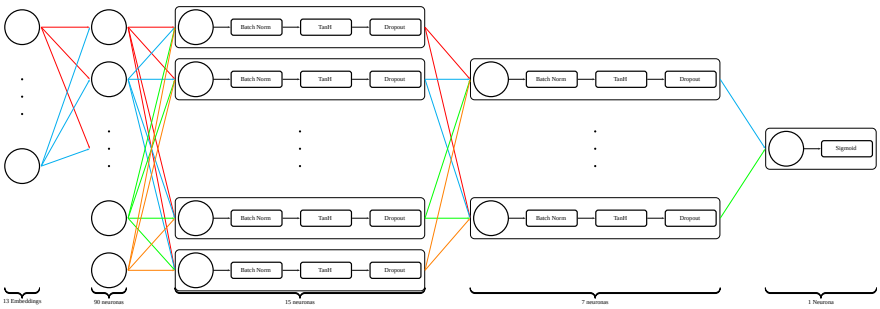


Figura 5.2: Clasificador: Menor error en pérdida: Red neuronal

5.4.1. Menor error en pérdida

El primer diseño tiene un error pérdida cuyas órdenes de magnitud son muy cercanas a cero, por tanto, presenta una validez relacionada con dicho hecho.

Los elementos que forman dicha red neuronal se pueden encontrar representados gráficamente en la figura 5.2 y son detallados de entrada a salida en la siguiente secuencia,

1. *Embedding layer*: Permite reducir los booleanos de dos dimensiones a una, consiguiendo optimizar los cálculos futuros.
2. *Linear 90-15*: Indica una interconexión total de las neuronas entre dos capas. En este caso, la primera de 90 neuronas y la segunda de 15.
3. *Batch norm*: Normalización de los datos de entrada para que estos tengan media cero y desviación estándar (es decir, la raíz cuadrada de la varianza) uno.
4. TanH: Los datos se introducen en una función de tipo tangente hiperbólica de forma que la salida comprende entre los valores -1 y 1
5. *Dropout*: Esta capa sirve para desechar el dato que tiene a la entrada con una probabilidad previamente determinada. En este caso, está establecida (la probabilidad de desecho) al 45 %.
6. *Linear 15-7*
7. *Batch norm*
8. TanH
9. *Dropout*
10. *Linear 7-1*
11. Sigmoid: Los datos se introducen en una función de tipo sigmoide de forma que la salida comprende entre los valores 0 y 1.

Debido a que solamente hay un tipo de valor a la salida y es booleano, como función de coste se hará uso de **Binary Cross Entropy Loss**. Además, el valor de *learning rate* que se ha considerado óptimo es **0.0001**.

5.4.1.1. Resultados

El resultado obtenido para este diseño será analizado a continuación.

Primeramente se evaluará cuál ha sido el comportamiento general del entrenamiento. Para ello, se hará uso de la curva pérdida. Este entrenamiento ha consistido en un total de 175 *epochs* [3]. En la figura 5.3 se puede analizar visualmente cual ha sido la evolución de la pérdida a lo largo de estos *epochs*.

La pérdida ha sido evaluada en tres situaciones distintas para cada uno de los *epochs*. En todas las gráficas se muestran el resultado de los cinco entrenamientos correspondientes a la validación cruzada.

- **Entrenamiento**. Del conjunto de datos de entrenamiento, aquellos con comportamiento normal y ataques (no clasificados). Únicamente se actualiza el valor de lo pesos en esta fase.
- **Verificación - Solo comportamiento normal**. Del conjunto de datos de verificación, solamente se han procesado aquellos que tiene comportamiento normal.
- **Verificación - Solo ataques**. Del conjunto de datos de verificación, solamente se han procesado aquellos que corresponden a ataques.

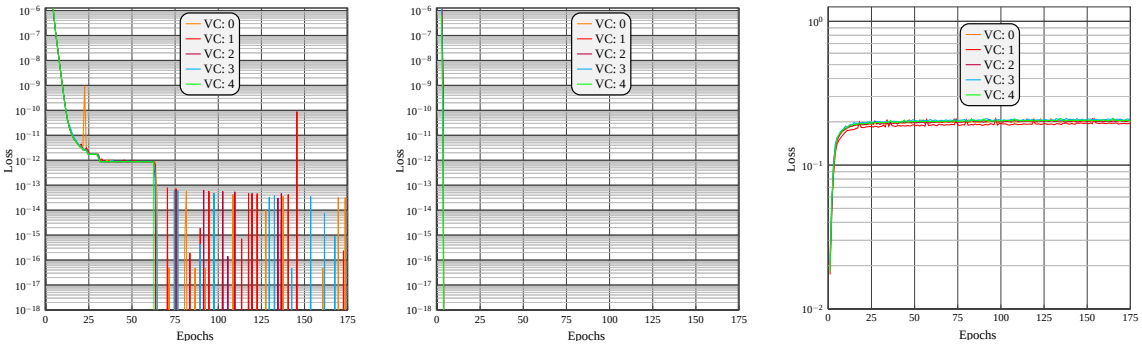
A medida que avanzan los entrenamientos, la red está aprendiendo qué es un comportamiento normal y qué es un ataque. Ahora bien, se centra en reducir el error del comportamiento normal ya que así se ha configurado la función coste, por tanto, sería lógico pensar que conseguirá reducir la pérdida del comportamiento normal (en lo que se centra) y aumentará cuando se introduzca un comportamiento de ataque (ya que no los aprende, solamente son usados para saber que no son el normal).

Debido a los dos casos a analizar se ha visto oportuno representar los mismos en diferentes gráficas y así poder examinar de forma correcta la evolucón.

Los resultados tras el análisis han sido

- **Entrenamiento**. Entre en el *epoch* 1 y 60, se puede observar una curva decreciente típica de un aprendizaje primerizo donde la red todavía no ha encontrado cuál es el patrón que rigen los datos de entrada. A partir del *epoch* 61, se puede encontrar un descenso brusco ocasionado porque la red ha aprendido el patrón de los datos. Es tal dicho descenso, que ni con escala logarítmica se puede llegar a apreciar cuál es su nivel mínimo, debido a lo cercano a cero que se encuentra.
- **Verificación - Solo comportamiento normal**. La red es capaz de detectar dicho comportamiento desde un momento inicial. Se puede apreciar cómo su pérdida es cercana a cero desde el momento inicial del entrenamiento.
- **Verificación - Solo ataques**. A diferencia del caso anterior, la curva tiene una pendiente ascendiente durante todo el entrenamiento. Al analizar los ejes, es importante destacar cómo este valor de pérdida se encuentra muchas órdenes de magnitud por encima de los valores anteriormente analizados.

[3] 1 *epoch* consiste en que la red procese completamente el conjunto de datos de entrenamiento y verificación, es decir, introducir todos los valores conocidos.



(a) Entrenamiento: Ataques no clasificados

(b) Verificación: Solo comportamiento normal

(c) Verificación: Solo comportamiento de ataques

Figura 5.3: Clasificador: Menor error en pérdida: Curva de pérdida

Se podría finalizar el análisis del coste a lo largo de los *epochs* con la verificación de lo esperado aplicando la lógica. En entrenamiento y comportamiento normal tiende a decrecer y en comportamiento de ataques, tiende a crecer.

Una vez se ha evaluado cuál ha sido el comportamiento general del entrenamiento, el siguiente paso es evaluar los resultados para el mejor *epoch*, es decir, para los resultados devueltos por el *early stop*.

Para realizar dicho examen se aplicará el análisis por curva ROC debido a la cantidad de información que ésta aporta. Para este diseño, el conjunto de curvas ROC obtenidas siguiendo los requisitos de la validación cruzada se encuentran en la figura 5.4.

Al analizar la misma, es fácil diferenciar que los cinco casos presentan una gráfica cercana a la de un clasificador perfecto. A pesar de ello, todavía ofrecen una cierta mejora.

Se ha obtenido un **valor de AUC medio** igual a **0.96044**.

Si se analiza la curva ROC obtenida con mayor valor de AUC, se puede obtener para el mejor umbral los siguientes valores de caracterización

- Tasa de falsos positivos = 0
- Tasa de verdaderos positivos = 0.96

Sería válido preguntarse el por qué de que si hay tanta diferencia entre el comportamiento normal y el de ataques, este hecho no se refleja en la curva ROC con una gráfica equivalente a la de un discriminador binario perfecto. Tal y como se detalla anteriormente, el análisis a nivel de entrenamiento presenta datos generales, en media, mientras que la evaluación de los resultados se centra únicamente en el mejor *epoch*. Aquellos datos que han tenido un comportamiento el cual no ha podido ser aprendido de forma óptima por la red neuronal, se encuentran enmascarados en la curva de pérdida (debido a la media) pero son mostrados en el análisis de la curva ROC.

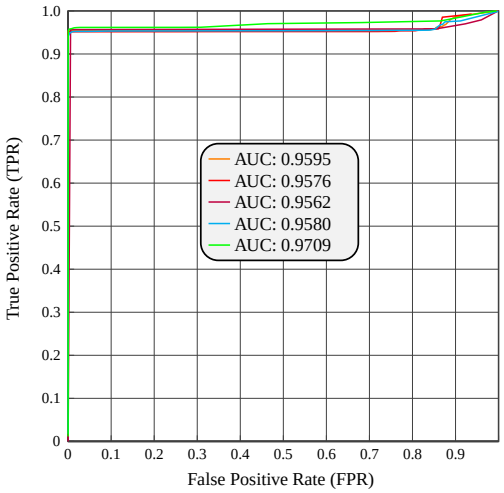


Figura 5.4: Clasificador: Menor error en pérdida: Curva ROC

5.4.2. Mejor TPR en ROC

El segundo diseño presenta unos valores de tasa de positivos correctos cercanos a la unidad, es decir, con dicho umbral casi todos los paquetes etiquetados como ataque lo serían realmente. Es interesante ya que con una tasa baja de falsos positivos, se conseguiría unos resultados casi óptimos por parte del clasificador implementado.

Los elementos que forman dicha red neuronal se pueden encontrar representados gráficamente en la figura 5.5 y son detallados de entrada a salida en la siguiente secuencia,

1. *Embedding layer*: Permite reducir los booleanos de dos dimensiones a una, consiguiendo optimizar los cálculos futuros.
2. *Linear 90-7*: Indica una interconexión total de las neuronas entre dos capas. En este caso, la primera de 90 neuronas y la segunda de 7.
3. *Batch norm*: Normalización de los datos de entrada para que estos tengan media cero y desviación estándar (es decir, la raíz cuadrada de la varianza) uno.
4. TanH: Los datos se introducen en una función de tipo tangente hiperbólica de forma que la salida comprende entre los valores -1 y 1
5. *Dropout*: Esta capa sirve para desechar el dato que tiene a la entrada con una probabilidad previamente determinada. En este caso, está establecida (la probabilidad de desecho) al 45 %.
6. *Linear 7-1*
7. Sigmoid: Los datos se introducen en una función de tipo sigmoide de forma que la salida comprende entre los valores 0 y 1.

A diferencia del modelo anteriormente especificado, en este se puede apreciar cómo se ha eliminado la capa intermedia que contenía un total de quince neuronas. Con esta modificación, se

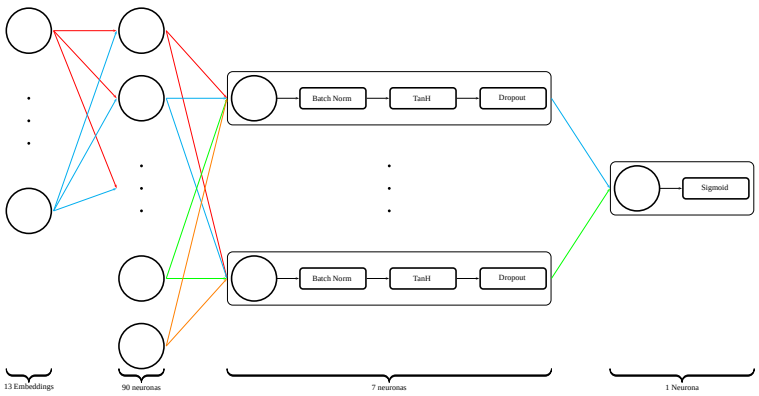


Figura 5.5: Clasificador: Mejor TPR en ROC: Red neuronal

ha conseguido reducir la complejidad de la red. Esta ha sido alguna de las diferencias que se han realizado entre los distintos modelos para poder encontrar cuál es el diseño óptimo.

Igual que anteriormente y debido a que solamente hay un tipo de valor a la salida, y es booleano, como función de coste se hará uso de **Binary Cross Entropy Loss**. Además, el valor de *learning rate* que se ha considerado óptimo es **0.0001**.

5.4.2.1. Resultados

Igual que en el diseño anterior, en primer lugar serán analizados los resultados del comportamiento general del entrenamiento. Se ha completado todo el entrenamiento de 175 *epochs*, o sea, no ha habido *early stop* como pasaba anteriormente, tal y como se puede ver en la figura 5.6.

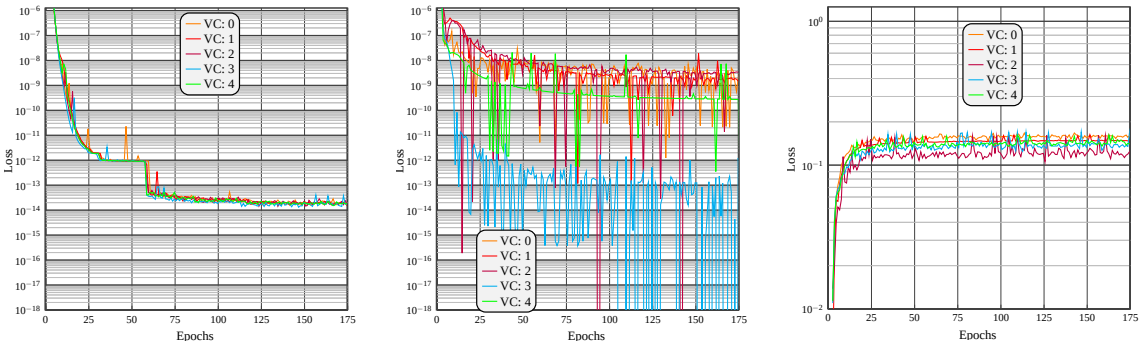
La técnica de entrenamiento ha sido idéntica a la anterior y, en consecuencia, los resultados esperados los mismos.

El detalle del análisis sería

- **Entrenamiento**. Entre en el *epoch* 1 y 50, se puede observar una curva decreciente típica de un aprendizaje primerizo donde la red todavía no ha encontrado cual es el patrón que rigen los datos de entrada. A partir del *epoch* 51, se puede encontrar un descenso ocasionado porque la red ha aprendido el patrón de los datos. Esta vez, el descenso no es tan brusco como anteriormente y se puede observar como tiende a valores de **2e-14**.

- **Verificación - Solo comportamiento normal**. Entre el *epoch* 1 y el 15, es observable cómo la red está aprendiendo el comportamiento de los datos. Una vez se ha llegado a dicho punto, la curva empieza a reducir el valor de la pendiente llegando a estar cada vez más estable. En el mejor de los casos, presenta un valor de error que tiende a **1e-13** mientras que en el peor de ellos, **3.5e-9**. En media, el valor de error al que se aproximaría es **1.5e-9**.

- **Verificación - Solo ataques**. Igual que en el diseño de red anterior, la curva tiene una pendiente ascendiente durante todo el entrenamiento. Al analizar los ejes, es importante destacar como este valor de pérdida se encuentra algunas órdenes de magnitud por encima de los valores del comportamiento normal, aunque dicha diferencia no es tan evidente como antes.



(a) Entrenamiento: Ataques no clasificados

(b) Verificación: Solo comportamiento normal

(c) Verificación: Solo comportamiento de ataques

Figura 5.6: Clasificador: Mejor TPR en ROC: Curva de pérdida

En resumen, se aprecia como igual que en el diseño anterior se cumple aquello que se esperaba, aunque de forma no tan obvia. Debido a la diferencia entre ambas gráficas de verificación, estos datos también se podrían considerar buenos.

Una vez se ha evaluado cuál ha sido el comportamiento general del entrenamiento, el siguiente paso es evaluar los resultados para el mejor *epoch*, es decir, para los resultados devueltos por el *early stop*.

Para realizar dicho examen se aplicará el análisis por curva ROC debido a la cantidad de información que ésta aporta. Para este diseño, el conjunto de curvas ROC obtenidas siguiendo los requisitos de la validación cruzada se encuentran en la figura 5.7.

Cuando se analiza, es fácil apreciar cómo en los cinco casos de la validación cruzada, las curvas se llegan a aproximar a las de un clasificador binario perfecto.

El análisis visual viene respaldado por el análisis analítico donde la media de los entrenamientos ofrece un AUC con valor **0.94874**

Cabe destacar cuáles serían las caracterizaciones para el mejor umbral del mejor y peor entrenamiento de la validación cruzada

- Mejor partición
  - Tasa de falsos positivos = 0.008
  - Tasa de verdaderos positivos = 0.995
- Peor partición
  - Tasa de falsos positivos = 0.084
  - Tasa de verdaderos positivos = 0.986

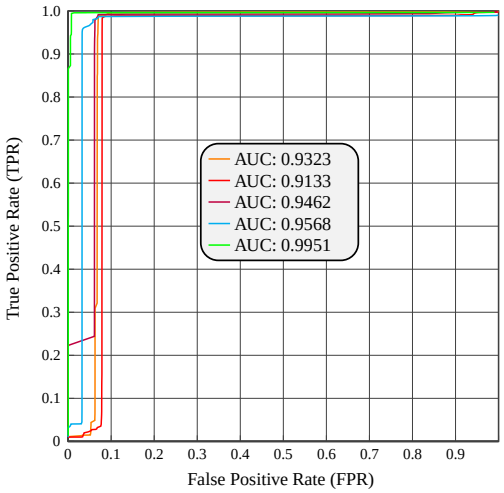


Figura 5.7: Clasificador: Mejor TPR en ROC: Curva ROC

5.4.3. Comparativa

Una vez detallados los dos posibles diseños, se debe escoger uno de ambos.

Debido a que ambos diseños tienen un buen comportamiento en general, para realizar la elección se debe tener en consideración cómo de bueno es el clasificador a la hora de discriminar los resultados.

Ambos modelos entrenados tiene valores cercanos de AUC en media tal y como se especifica a continuación.

- Menor error en pérdida: 0.96
- Mejor TPR en ROC: 0.95

En cambio, si analizan las gráficas, para el umbral óptimo se obtiene que

- Menor error en pérdida: Tiene una menor tasa de falsos positivos (FPR) a costa de no obtener un valor unitario de tasas de positivos correctos, aunque si que muy cercana a la perfección (0.96)
- Mejor TPR en ROC: La tasa de positivos correctos es casi unitaria (0.99) aunque es evidente una cierta varianza en los falsos positivos que comprende entre 0 y 0.8 para el mejor y peor valor respectivamente.

5.4.3.1. Elección

Tras este análisis, el diseño escogido ha sido el segundo (mejor TPR en ROC) ya que

1. El comportamiento de las gráficas de pérdida es normal en un entrenamiento correcto
2. Los resultados de la curva ROC indican un porcentaje de positivos correctos cercanos a la unidad a pesar de tener una varianza en los falsos positivos. Se ha considerado que es mejor un falso positivo que no un positivos no identificado.

Capítulo 6

Métodos no supervisados para la detección de anomalías

6.1. Etapa previa

Al igual que en el entrenamiento con método supervisado, el primer paso a realizar consiste en crear en conjuntos de datos a emplear en el entrenamiento. Para ello, se siguen los mismos pasos que antes (sección 5.2) pero con cambios detallados a continuación.

6.1.1. Conjunto de datos de entrenamiento

Dado que se va a hacer uso de un método no supervisado, la red solamente podrá disponer del comportamiento normal del sistema en el entrenamiento.

Este es un dato que se conoce en todos los sistemas de comunicaciones ya que se obtiene a partir de analizar los paquetes que circulan por el medio.

6.1.2. Datos usados para el entrenamiento

En el entrenamiento supervisado se hacía uso de la variable *labels* como etiqueta. En este caso, no hay etiquetas, por tanto tampoco hará falta dicha variable en entrenamiento. Las etiquetas que se usarán en la función de coste se definen en las siguientes secciones dependiendo del diseño a implementar.

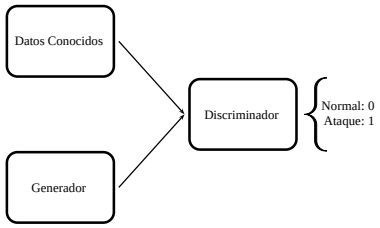


Figura 6.1: Detector de anomalías: Estructura de componentes del modelo DGAN

6.2. Detector de anomalías

El segundo tipo de sistema de detección a implementar es uno basado en anomalías. Para la correcta implementación de este sistema, se entiende que el entrenamiento debe ser no supervisado, ya que así la red aprenderá cuál es el comportamiento normal del sistema y clasificará como ataque (o anomalía) todo aquello que exceda unos ciertos márgenes establecidos por el diseñador o usuario final del sistema.

Para la implementación de este modelo, se ha optado por un diseño del tipo DGAN (Deep Generative Adversarial Network). Este se basa en las redes **GAN** detalladas en la subsección 2.4.2. Como el sistema que se desea implementar es un clasificador binario, esta función la realizará la red neuronal del discriminador con una modificación, para así, conseguir que esta (la red del clasificador) devuelva dichos valores (cero o uno) y no el error obtenido al discriminar. Por tanto, la estructura de bloques resultante con la modificación de la salida del discriminador se podría visualizar gráficamente en la figura 6.1.

El procedimiento seguido para entrenar una red tipo GAN es el siguiente:

1. Se introduce el conjunto de datos de entrenamiento conocido al discriminador. Como solo hay comportamiento normal, con etiqueta de comportamiento normal.
2. Se obtiene el coste del discriminador.
3. Se generan datos en el generador y se usan para alimentar al discriminador con etiqueta de ataque.
4. Se obtiene el coste del discriminador.
5. Se suman ambos costes y se actualizan los pesos del discriminador.
6. Se vuelven a introducir los mismos datos anteriormente generados por el generador al discriminador, pero esta vez con etiqueta de comportamiento normal.
7. Se obtiene el coste del discriminador.
8. Se actualizan los pesos del generador.

Figura 6.2: Detector de anomalías: DGAN: Generador: Estructura de entrenamiento

Ya ha sido definido cuál es el modelo escogido como red neuronal y cual será el procedimiento a seguir para realizar el entrenamiento. Para finalizar, falta determinar cuales serán los elementos que conformarán dicha red neuronal.

1. **Conjunto de datos conocido**
   - Entrenamiento
     - Comportamiento normal del sistema
   - Verificación
     - Comportamiento normal del sistema
     - Ataques
2. **Discriminador**: La red que se ha usado en secciones anteriores para implementar el entrenamiento supervisado y se ha considerado que es la mejor, será la usada en esta etapa para realizar la función de discriminador. De hecho, cuando se ha nombrado la misma ya se le ha definido dicho nombre no de forma trivial.
3. **Generador**: A continuación será definida la red neuronal que se encargará de generar los datos.

6.2.1. Generador

La red del generador es la encargada de producir datos para alimentar al discriminador a partir de una entrada formada por unas variables aleatorias de una distribución a determinar. Esta red debe llegar a originar datos lo más parecidos al conjunto de entrenamiento posible.

El diseño del generador se ha basado en una estructura del tipo **autoencoder** (detallada en la subsección 2.4.1). En primera instancia se podría llegar a pensar que no tiene lógica hacer uso de este tipo de redes debido a que no se van a tener datos válidos a la entrada (ya que la entrada está formada por una distribución aleatoria de datos), aunque tal y como se describe en la definición de dicha estructura, introduciendo valores de una distribución de datos correcta al *decoder*, éste debería ser capaz de generar datos válidos. Por tanto, la estructura que resume la red del generador se podría observar en la figura 6.2.

A pesar de ello todavía existe un problema, tal y como se ha definido anteriormente el *autoencoder*, la distribución de los datos a la salida del codificador no es conocida.

La solución consiste en una capa que ya ha sido usada con anterioridad, el *Batch norm*. Esta capa se encarga de asegurarse que los datos a la salida de la misma cumplen una media cero y

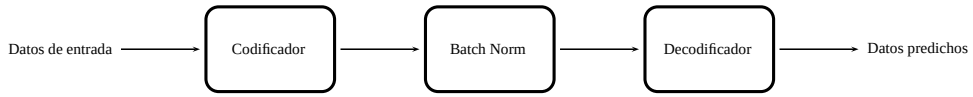


Figura 6.3: Detector de anomalías: DGAN: Generador: Estructura de entrenamiento previo

varianza uno (se puede observar una representación gráfica del *autoencoder* con la capa del *Batch norm* en la figura 6.3). Una vez ya se conoce que distribución existirá como entrada del *decoder*, solamente falta generar dicho datos. Para generarlos, se ha hecho uso de un método de la librería Pytorch que devuelve una matriz de las dimensiones solicitadas cuyos valores cumplen media cero y varianza uno.

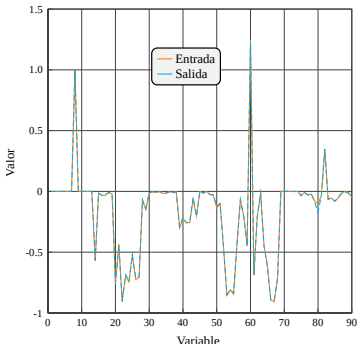
Debido a la rápida convergencia del discriminador, cabe la posibilidad de una inestabilidad en la red del DGAN. De hecho, así ha sido y, por tanto, se ha decidido preentrenar la red del generador y cargar los valores óptimos de esta red en el DGAN. Debido a que no se conoce cual va a ser el diseño óptimo del generador, se ha optado por no definir el mismo en esta fase, sino cuando este el DGAN implementado. Para poder conseguir ambos requerimientos, se ha preentrenado la red del generador con una variedad de diseños, los cuales, serán posteriormente evaluados en el DGAN para obtener cual es el óptimo.

Alguien podría preguntarse el porque usar el DGAN para implementar dicho clasificador binario y no emplear únicamente un *autoencoder*. De hecho, es lógico pensar que si solamente se enseña a la red del *autoencoder* el comportamiento normal del sistema, al introducir un dato que forme parte de un ataque, este tendría dificultad para reproducirlo ya que no se debería encontrar en el espacio junto a los datos de comportamiento normal y, en consecuencia, el error de discriminación sería muy grande.

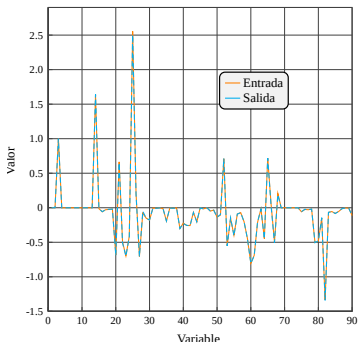
Para realizar dicha comprobación, en los entrenamientos realizados con el *autoencoder* se han seleccionado aleatoriamente muestras de entrada y salida del mismo para las distintas fases (entrenamiento con comportamiento normal, verificación con comportamiento normal y verificación con ataques) y se han mostrado gráficamente.

El resultado se encuentra en la figura 6.4 para uno de los distintos diseños del generador y, por extraño que pueda llegar a parecer, la red es capaz de regenerar tanto muestras de comportamiento normal como de ataques (cabe hacer memoria de que nunca había visto ningún dato de ataques). Aunque parezca extraño, este fenómeno no está aislado, cuando se detallan los otros tipos de aprendizaje posibles se comentan las redes NLP. Estas redes, son capaces de generar pensamiento a partir del contexto, sin haber sido entrenadas para dicho propósito.

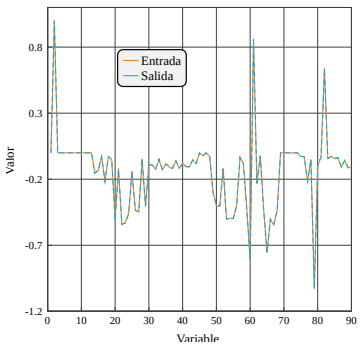
Recapitulando, el hecho por el que se debe implementar una red DGAN y no solo un *autoencoder* es debido a que esta red es capaz de regenerar tanto datos con comportamiento normal como datos correspondientes a ataques.



(a) Entrenamiento: Comportamiento normal



(b) Verificación: Solo comportamiento normal



(c) Verificación: Solo comportamiento de ataques

Figura 6.4: Detector de anomalías: Autoencoder: Comparativa datos introducidos y predichos

6.2.2. DGAN: Generador fijo

Teniendo todos los componentes que implementan el DGAN definidos, llega el momento de realizar el entrenamiento.

Tal y como se detalla en la subsección 2.4.2, hay que tener en cuenta la inestabilidad de este tipo de diseño. Cuando se empezó a realizar el entrenamiento de la forma detallada anteriormente, no se consiguió que la red funcionase como se esperaba.

La solución planteada a este inconveniente consiste en solamente actualizar los pesos de la red del discriminador en el DGAN. Para que la red del generador origine cada vez mejores resultados, y no perder los objetivos que tiene el diseño de tipo GAN, se decide realizar el entrenamiento de dicha red de forma paralela con los datos del comportamiento normal del sistema y, regularmente, introducir el valor de los nuevos pesos a la red (del generador) del DGAN.

Resumiendo, se van a realizar dos entrenamientos en paralelo detallados a continuación

1. Se entrenará la red del generador solamente con comportamiento normal
2. Se entrenará la red del DGAN siguiendo el siguiente procedimiento
    a) Se introduce el conjunto de datos de entrenamiento conocido al discriminador. Como solo hay comportamiento normal, con etiqueta de comportamiento normal.
    b) Se obtiene el coste del discriminador.
    c) Se generan datos en el generador y se usan para alimentar al discriminador con etiqueta de ataque.
    d) Se obtiene el coste del discriminador.
    e) Se suman ambos costes y se actualizan los pesos del discriminador.

Se han realizado un conjunto de entrenamientos en los cuales se ha mantenido siempre el mismo discriminador pero se ha modificado el modelo del generador así como el valor de *learning rate* inicial para finalmente obtener una red basada en DGAN óptima.

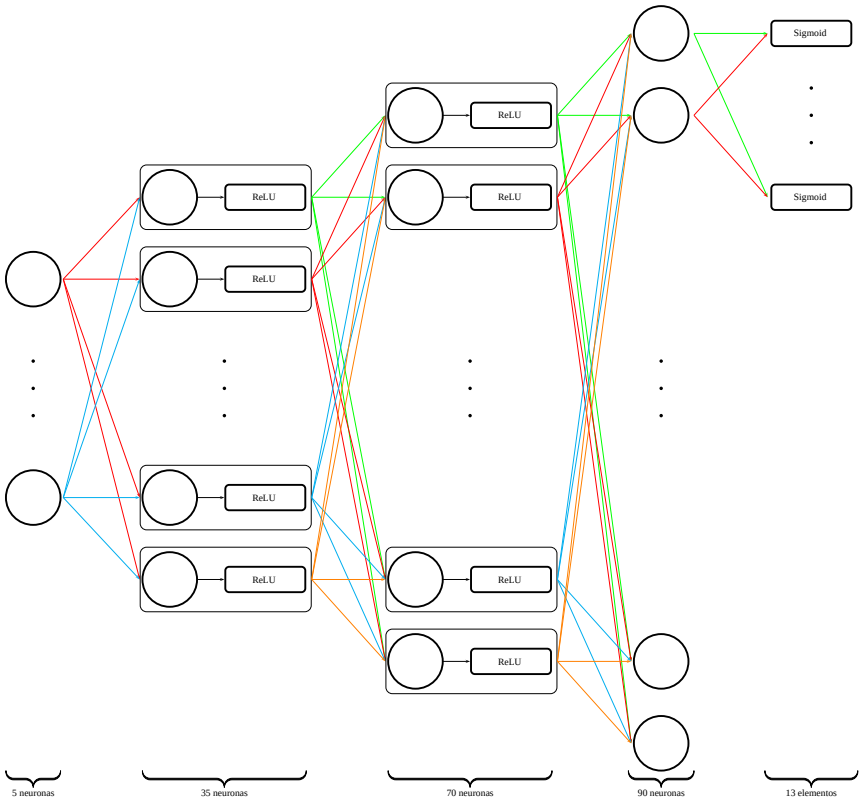
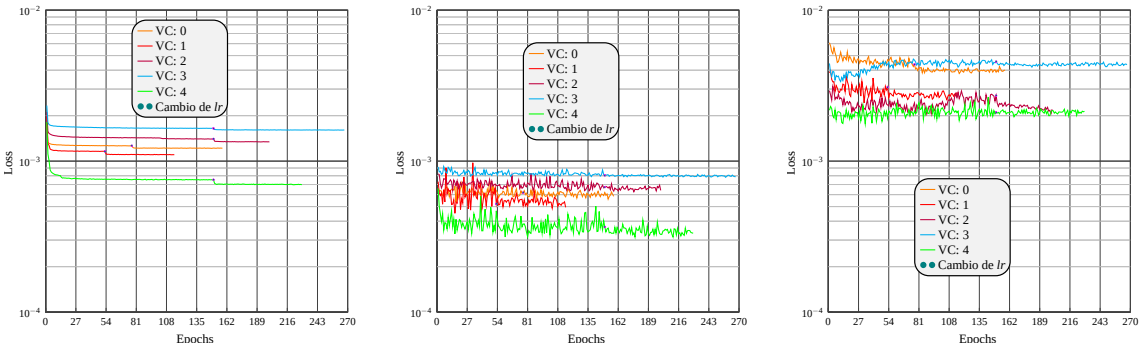


Figura 6.5: Detector de anomalías: DGAN: Generador fijo: Red generador

El diseño que ha obtenido mejor evaluación, ha sido entrenado primeramente con un *learning rate* cuyo valor era **0.0001** y ha proseguido el entrenamiento modificando el *learning rate* a valor **0.00001**. El generador (más específicamente, el *decoder*) que finalmente ha resultado ideal, se encuentra esquematizado en la figura 6.5 y se detalla a continuación

- *Linear 5-35*: Indica una interconexión total de las neuronas entre dos capas. En este caso, la primera de 5 neuronas y la segunda de 35.
- *ReLU*: Función que devuelve el valor de entrada si este es mayor que cero y cero en otro caso
- *Linear 35-70*
- *ReLU*
- *Linear 70-90*
- Sigmoid: Los datos se introducen en una función de tipo sigmoide de forma que la salida comprende entre los valores 0 y 1.



(a) Entrenamiento: Comportamiento normal

(b) Verificación: Solo comportamiento normal

(c) Verificación: Solo comportamiento de ataques

Figura 6.6: Detector de anomalías: DGAN: Generador fijo: Curva de pérdida

6.2.2.1. Resultados

A continuación se detallarán cuales han sido los **resultados del discriminador**, ya que es el encargado de indicar que tipo de comportamiento tiene a la entrada. Igual que anteriormente y debido a que solamente hay un tipo de valor a la salida, y es booleano, como función de coste se hará uso de **Binary Cross Entropy Loss**.

NOTA: Si se desease implementar el DGAN completo, para entrenar la red del generador, como función de coste se debería hacer uso del **MSE** para los valores numéricos y del **Binary Cross Entropy Loss** para los booleanos.

En primer lugar serán analizados los resultados del comportamiento general. Se muestran ambos entrenamientos separándolos con un punto sobre la gráfica en el instante en el que se ha decrementado el *learning rate*. Las gráficas resultantes se pueden ver en la figura 6.6. A diferencia de en los casos anteriores, se aprecia como cada uno de los casos de la validación cruzada finalizan en un *epoch* distinto, de hecho, ninguno de ellos llega a completar los 300 *epochs* que se había establecido como máximo. El hecho anterior es debido a que el mecanismo de *early stop* ha detectado *overfiting* y ha finalizado el entrenamiento. Los resultados mostrados en dichas gráficas finalizan en el punto óptimo, no en el que se ha finalizado el entrenamiento.

El detalle del análisis sería

- **Entrenamiento**. Se observa una bajada muy brusca en los primeros *epochs* y que en seguida se transforma en una pendiente muy poco pronunciada descendente. Si que es apreciable como en los cambios de *learning rate* vuelve a tener un descenso. En media, tiende a valores cercanos a **1e-3**
- **Verificación - Solo comportamiento normal**. Desde el principio muy estable aunque con una varianza muy elevada en el primer entrenamiento la cuál se reduce en el segundo. En media, tiende a valores próximos a **6e-4**
- **Verificación - Solo ataques**. En media, tiende a valores semejantes a **3e-3**

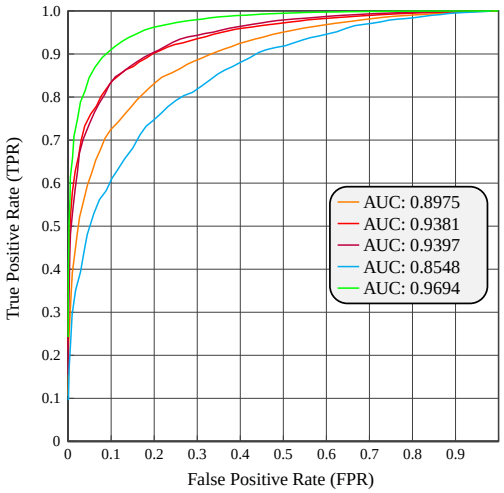


Figura 6.7: Detector de anomalías: DGAN: Generador fijo: Curva ROC

A continuación, se analizan cuales son los resultados obtenidos para el mejor *epoch*. Igual que anteriormente, se hará uso de la curva ROC.

La gráfica resultante del calculo de la curva ROC para cada unos de los casos de la validación cruzada se encuentra en la figura 6.7.

Realizando un análisis visual, es fácil deducir que los resultados no son tan buenos como en el caso supervisado, un hecho que se esperaba. A pesar de ello, todas las curvas tienden a valores de un clasificador binario perfecto, lo que indica que el funcionamiento no es el óptimo pero si es correcto.

Si se realiza un análisis numérico con el valor del AUC de las curvas, se observa como el mejor de los casos tiene una puntuación de **0.9694**, mientras que el peor de **0.8548**. Todos ellos, por tanto, están por encima de 0.85, un valor muy bueno teniendo en consideración el tipo de entrenamiento. De hecho, la media de las puntuaciones es **0.92**.

Teniendo en consideración el mejor caso (aquel con AUC score con valor 0.9694), podríamos caracterizar el umbral óptimo con los valores

- Tasa de falsos positivos = 0.103
- Tasa de verdaderos positivos = 0.915

En este último análisis, se evidencia como la tasa de falsos positivos tiene un valor muy elevado en comparación con casos anteriores. Un hecho también esperado, ya que se partía con el conocimiento de que el entrenamiento no supervisado iba a tener resultados no tan buenos como en el supervisado.

Capítulo 7

Comparativa entre los resultados del entrenamiento supervisado y del no supervisado

Una vez obtenidos todos los resultados del proyecto, se realiza una comparativa de los mismos.

En primer lugar, es recomendable el recuerdo de las premisas previas. En un entrenamiento supervisado, los resultados deberían ser mejores que en un uno no supervisado, ya que la red tiene conocimiento de todos los datos que debe clasificar. Por otro lado, si se realiza un entrenamiento no supervisado, la red es capaz de clasificar correctamente no solo ataques ya conocidos, sino también aquellos desconocidos y detectará más o menos de éstos según el error que defina el diseñador de la red (o el usuario final).

En la tabla 7.1 se encuentra una comparativa de los resultados obtenidos.

Tal y como se puede analizar, los resultados del entrenamiento supervisado son mejores que los del entrenamiento no supervisado.

Examinando los valores del entrenamiento supervisado, ambos diseños obtienen valores de AUC muy parejos en media. A pesar de ello, el primero de éstos no llega a obtener una tasa de positivos correctos con valor unitario (aunque tiene un valor muy cercano), teniendo a su favor una tasa de falsos positivos muy próxima a cero. Por otro lado, el segundo diseño implementado obtiene un valor de falsos positivos superior al anteriormente mencionado, no obstante, su tasa de verdaderos positivos es cercana a la unidad (0.99).

Por otro lado, está el resultado del entrenamiento no supervisado. Se obtiene un valor de AUC muy cercano a los del entrenamiento supervisado, indicando de que el funcionamiento de este clasificador binario se podría considerar bueno. A pesar de ello, para un valor pequeño de falsos positivos (FPR), solamente se consigue una tasa de positivos correctos (TPR) del 0.75.

En resumen, el diseño del entrenamiento no supervisado necesita mejoras para conseguir reducir la tasa de falsos positivos (FPR) y así lograr resultados más cercanos a los del diseño supervisado, a pesar de ello, los valores de AUC calculados indican que el diseño no es malo (0.92 frente a 0.95).

	ROC		
	TPR	FPR	AUC
Supervisado: Menor Pérdida	$0{,}96 \pm 0{,}01$	0	$0{,}96 \pm 0{,}01$
Supervisado: Mejor TPR	0.99	$0{,}4 \pm 0{,}4$	$0{,}95 \pm 0{,}04$
No Supervisado	0.75	$0{,}1 \pm 0{,}15$	$0{,}92 \pm 0{,}05$

Tabla 7.1: Comparativa de los resultados obtenidos: ROC

Capítulo 8

Propuesta de implementación

El objetivo final de este proyecto es la creación de un sistema de detección de intrusos basado en anomalías, como ha sido detallado en seccione anteriores. Este dispositivos es el encargado de informar al responsable de ciberseguridad sobre un posible ataque dentro de la red que está monitorizando. Es importante tener conocimiento que este dispositivo no va a impedir ataques, solamente informar sobre la posibilidad de los mismos.

Se busca que este sistema funcione de la mejor forma posible, por tanto, tras estudiar todos los resultados obtenidos, a continuación se detalla cual es la propuesta de implementación del sistema.

A causa de que los resultados del entrenamiento no supervisado no son óptimos, se considera que este sistema no se debería implementar con un único modelo sino con una combinación de distintos modelos, con el objetivo de que los resultados mostrados al usuario final sean lo más veraces posibles.

El diseño pensado intenta aprovechar las virtudes de ambos tipos de entrenamiento, es decir, la **capacidad de detección de nuevos ataques** propia del diseño con el modelo **no supervisado** y la **precisión en la detección** del modelo supervisado.

El procedimiento manifestado para informar al usuario si el comportamiento es normal o es un ataque se puede observar en la figura 8.1 y seguiría los siguientes pasos

1. El analizador de redes realizar la captura de un paquete
2. El conjunto de paquetes que forman una conexión sería procesado para obtener los valores más significativos que serán usados como entrada para la red neuronal
3. Los datos serían evaluados por la red neuronal con entrenamiento **no supervisado**
4. Si no fuese un ataque, finalizaría la secuencia e informaría al usuario que con un 0 % de probabilidad hay ataque, en caso contrario, continuaría.
5. Los datos serían evaluados por la red neuronal con entrenamiento **supervisado**
6. Si fuese un ataque, finalizaría la secuencia e informaría al usuario que con un 100 % de probabilidad hay ataque, en caso contrario, continuaría.
7. Con todos los datos obtenidos por ambas redes, se calcularía con que probabilidad hay un ataque y se informaría al usuario de esta.

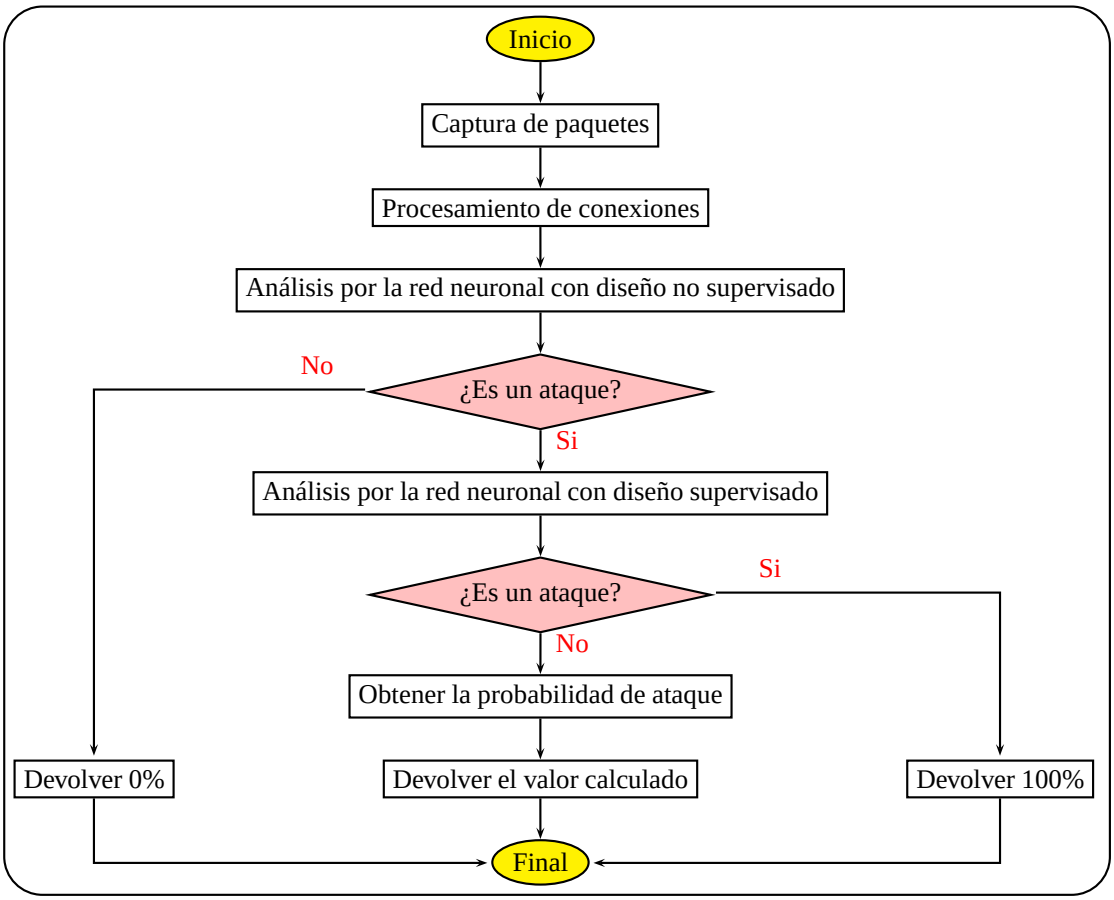


Figura 8.1: Propuesta implementación: Pasos para la detección de un ataque

Con esta implementación, el **usuario final** sería el que decidiese la **restricción** del sistema de detección de intrusos de forma que a **menor error**, **más ataques nuevos** serían detectados pero también debería analizar más posibles conexiones con un probabilidad baja de ataque que finalmente no resultasen serlo.

El diseño anteriormente detallado también implica mantener la base de datos actualizada para un funcionamiento óptimo, a pesar de ello, permite modelar y etiquetar ataques nuevos cuando son descubiertos.

Capítulo 9

Conclusiones y futuros trabajos

9.1. Conclusiones

La finalidad de este proyecto ha sido implementar un sistema de detección de intrusos basado en anomalías haciendo uso de inteligencia artificial.

Para conseguir dicho objetivo, primero se ha debido de aprender en que se basa la inteligencia artificial, distintos tipos de implementación, como implementar cada uno de ellos, el conjunto de funciones de coste, etc. para finalmente poder elegir con criterio cual es en cada uno de los apartados anteriormente listados, la mejor opción para el problema a resolver entre todas las disponibles.

Una vez realizada dicha tarea, se ha debido realizar la selección que se ha considerado óptima. A continuación, implementar todos los diseños y entrenarlos. Una vez entrenados, verificar que ha funcionado como se esperaba, y si no ha sido así, analizar el error obtenido para realizar los cambios oportunos acorde al problema de diseño.

Según los datos obtenidos, se ha verificado que en la mayoría de los casos, los resultados que en teoría se esperaban, se han confirmado. Entre otras, y tal y como se ha detallado anteriormente, se ha demostrado como al realizar un entrenamiento supervisado se obtienen mejores resultados que con un entrenamiento no supervisado.

Por otro lado, se han obtenido resultados que no se esperaban. Este es, por ejemplo, el resultado del *autoencoder* al introducir un dato de un ataque cuyo comportamiento no había procesado con anterioridad. Se esperaba que el error obtenido fuese muy elevado y, por el contrario, la red ha sido capaz de devolver a la salida el mismo dato que a la entrada.

Finalmente, se ha elaborado una propuesta de diseño que incluye la parte a destacar de cada uno de los sistemas de aprendizaje de forma que el sistema no es solo capaz de tener una gran exactitud a la hora de detectar ataques ya conocidos, sino que además tiene competencia para detectar nuevos ataques e informar al responsable de ciberseguridad. Si finalmente este decide que es un ataque, se podría modelar, etiquetar y almacenar su comportamiento en el conjunto ya conocido para que en futuras ocasiones sea detectado. Asimismo, teniendo una base común de conocimiento, éste se podría compartir.

9.2. Futuros trabajos

Este trabajo ofrece un amplio abanico de posibilidades para continuar profundizando, comparando y mejorando los resultados obtenidos.

Primeramente, es importante destacar que prácticamente cada semana se puede encontrar un virus nuevo o una modificación de uno existente que puede dificultar la tarea de estos dispositivos. Por tanto, implica una constante investigación del comportamiento de los nuevos ataques, incorporarlos al conjunto de datos conocido e ir mejorando el sistema con la retroalimentación de esta información.

Para continuar, la red del DGAN todavía tiene un comportamiento que ofrece una tasa de falsos positivos muy elevada. Se debería profundizar en dicho diseño e ir consultando las nuevas técnicas de *deep learning* por si alguno de los avances publicados pudiese aportar beneficios a la hora de detectar ataques con el modelo de entrenamiento no supervisado.

Por otro lado, se ha propuesto una solución dual para evitar los problemas de inestabilidad presentados por las redes de tipo GAN. Una posible rama en la mejora consistiría en diseñar un sistema que solamente hiciese uso de la red del DGAN, para ello, se debería implementar un diseño en el generador y en el discriminador cuyos pesos evolucionasen a velocidades similares, no solo eso, se debería implementar un sistema que avisase si una red convergiese más rápida que la otra para finalizar el entrenamiento y alertar al diseñador de dicho problema.

Finalmente, realizar un mecanismo que fuese capaz de procesar el conjunto de datos obtenidos por uno o varios sistemas de detección de intrusos y mostrar dicha información de forma simple al usuario final (ya sea creando gráficas interactivas, mostrando los incidentes de ciberseguridad más preocupantes en ciertos sectores de la red, lanzando avisos personalizados,...) para así poder crear **ciberconciencia situacional**, es decir, mostrar de manera sencilla toda la información de interés proveniente de estos sistemas al responsable de ciberseguridad.

Bibliografía

[1] Cyber Threat Intelligence centre. *COVID-19 : CYBER THREAT ASSESSMENT*. External Report. Thales group, mar. de 2020. URL: https://www.thalesgroup.com/sites/default/files/database/document/2020-04/2020-03-24_COVID-19_CYBER_THREAT_ASSESSMENT_%28ENG%29_0.pdf.

[2] Cyber Threat Intelligence centre. *REMOTE WORKING IN TIMES OF CRISIS: CYBER THREAT ASSESSMENT*. External Report. Thales group, mar. de 2020. URL: https://www.thalesgroup.com/sites/default/files/database/document/2020-04/2020-04-03_COVID-19_THREAT_ON_REMOTE_WORKING_%28ENG%29%20%283%29.pdf.

[3] Check Point Software Technologies. "Aumentan los ciberataques desde que empezó la crisis del coronavirus". En: *redseguridad.com* (abr. de 2020). URL: https://www.redseguridad.com/actualidad/aumentan-los-ciberataques-desde-que-empezo-la-crisis-del-coronavirus_20200413.html.

[4] David Marchal. "El coronavirus pone a prueba la ciberseguridad de todo el mundo". En: *seguritecnia.es* (mayo de 2020). URL: https://www.seguritecnia.es/tecnologias-y-servicios/ciberseguridad/el-coronavirus-pone-a-prueba-la-ciberseguridad-de-todo-el-mundo_20200521.html.

[5] Communications Security Establishment (CSE) y Canadian Institute for Cybersecurity (CIC). *A Realistic Cyber Defense Dataset (CSE-CIC-IDS2018)*. 2019. URL: https://registry.opendata.aws/cse-cic-ids2018.

[6] Qianru Zhou y Dimitrios Pezaros. "Evaluation of Machine Learning Classifiers for Zero-Day Intrusion Detection - An Analysis on CIC-AWS-2018 dataset". En: *CoRR* abs/1905.03685 (2019). arXiv: 1905.03685. URL: http://arxiv.org/abs/1905.03685.

[7] V. Kanimozhi y T. Prem Jacob. "Artificial Intelligence based Network Intrusion Detection with hyper-parameter optimization tuning on the realistic cyber dataset CSE-CIC-IDS2018 using cloud computing". En: *ICT Express* 5.3 (2019), págs. 211-214. ISSN: 2405-9595. DOI: https://doi.org/10.1016/j.icte.2019.03.003. URL: http://www.sciencedirect.com/science/article/pii/S2405959518305976.

[8] John McCarthy. "What is artificial intelligence?" En: (1998). URL: http://www-formal.stanford.edu/jmc/whatisai/whatisai.html.

[9] Stuart J Russell, Peter Norvig y Peter Norvig. "Artificial intelligence: Prentice Hall series in artificial intelligence". En: (2003).