

The logo consists of the lowercase letters "viu" in white, enclosed within a rounded orange circle.

viu

**Universidad  
Internacional  
de Valencia**

# **Modelo neuro-difuso para la identificación de negocios fachada**

Titulación:  
Master en Inteligencia  
Artificial  
Curso académico  
2022-2023

Alumno/a: Efraím Alberto  
Casas Herrera  
D.N.I: 8127080

Director/a de TFM: José  
Ángel Olivas

Convocatoria:  
Segunda

Es relativamente fácil conseguir que los ordenadores muestren capacidades similares a las de un humano adulto en un test de inteligencia o a la hora de jugar a las damas, y muy difícil lograr que adquieran las habilidades perceptivas y motoras de un bebé de un año

Hans Moravec



# Agradecimientos

- Al profesor José Angel Olivas por su dedicación, entusiasmo y sobre todo por mostrarnos otra visión de la inteligencia artificial, más allá de los datos.
- A todos los profesores de la maestría, por su esfuerzo y los conocimientos compartidos.
- A Pelé, mi fiel compañero de cuatro patas que estuvo a mi lado durante todas las clases de este programa, casi hasta el último día. Espero que algún día pueda utilizar el conocimiento adquirido para identificar y prevenir las causas de tu deceso.



# Índice general

Índice de figuras . . . . .	III
Índice de tablas . . . . .	V
Índice de algoritmos . . . . .	VI
Resumen . . . . .	1
1. Introducción . . . . .	3
1.1. Pasarelas de pago . . . . .	3
1.2. Negocios Fachada . . . . .	4
2. Objetivos . . . . .	5
3. Redes Neuronales Artificiales . . . . .	7
3.1. Historia . . . . .	7
3.2. Perceptrón Simple . . . . .	8
3.3. Perceptrón Multicapa (MLP) . . . . .	8
3.4. Redes Neuronales Convolucionales (CNN) . . . . .	9
3.5. Redes Neuronales Recurrentes (RNN) . . . . .	10
3.6. Redes Neuronales de Memoria a Corto y Largo Plazo (LSTM) . . . . .	11
3.7. Redes Neuronales Generativas Adversarias (GAN) . . . . .	11
3.8. Autoencoders . . . . .	12
4. Lógica Borrosa . . . . .	15
4.1. Conceptos Básicos y Definiciones . . . . .	15
4.2. Conjuntos Difusos . . . . .	15
4.3. Control Difuso . . . . .	16
4.4. Modelo Mamdani . . . . .	17
4.5. Modelo Takagi-Sugeno . . . . .	21
5. Metodología . . . . .	23



5.1. Diseño de la Investigación . . . . .	23
5.2. Recopilación de Datos . . . . .	23
5.3. Preprocesamiento de Datos . . . . .	23
5.3.1. Selección de atributos . . . . .	24
5.3.2. Imputación de valores atípicos . . . . .	25
5.3.3. Normalización de los atributos . . . . .	26
5.3.4. Generación de muestras sintéticas y balanceo de clases . . . . .	26
5.4. Construcción de Modelos . . . . .	27
5.4.1. Modelos de redes neuronales . . . . .	27
5.4.2. Modelo de Lógica Difusa . . . . .	33
5.5. Validación de Modelos . . . . .	39
5.5.1. Validación del Modelo de Redes Neuronales . . . . .	39
5.5.2. Validación del Modelo de Lógica Difusa . . . . .	43
5.6. Sistema Neuro-Difuso . . . . .	44
5.7. Herramientas Utilizadas . . . . .	44
6. Resultados y Discusión . . . . .	47
7. Conclusiones . . . . .	49
8. Limitaciones y Perspectivas de Futuro . . . . .	51
Lista de Acrónimos . . . . .	53
A. Código Redes Neuronales . . . . .	54
B. Código Lógica Difusa . . . . .	57
Bibliografía . . . . .	60



# Índice de figuras

3.1. Perceptrón simple . . . . .	9
3.2. Perceptrón multicapa . . . . .	9
3.3. Redes neuronales convolucionales (CNN) . . . . .	10
3.4. Redes neuronales recurrentes (RNN) . . . . .	11
3.5. Redes neuronales de memoria a corto y largo plazo (LSTM) . . . . .	12
3.6. Redes neuronales generativas adversarias (GAN) . . . . .	12
3.7. Autoencoder . . . . .	13
4.1. Conjunto borroso comercios riesgo . . . . .	16
4.2. Estructura controlador difuso . . . . .	17
4.3. Funciones de pertenencia para calidad y servicio . . . . .	18
4.4. Funciones de pertenencia para la propina . . . . .	18
4.5. Evaluación Regla 1 . . . . .	19
4.6. Evaluación Regla 2 . . . . .	19
4.7. Salida de las reglas . . . . .	20
5.1. Capacidad discriminativa de los atributos . . . . .	24
5.2. Arquitectura MLP. . . . .	28
5.3. Arquitectura autoencoders comercios. . . . .	30
5.4. Arquitectura de nuestro autoencoder. . . . .	31
5.5. Conjunto borroso comercios riesgo . . . . .	33
5.6. Función de membresía atributo V1 . . . . .	34
5.7. Llamado a motor de inferencia difuso . . . . .	38
5.8. Resultado llamado a motor de inferencia difuso . . . . .	38
5.9. MLP: Historial de entrenamiento experimento 010. . . . .	40
5.10. MLP: Matriz de confusión experimento 010. . . . .	40
5.11. MLP: Curva ROC experimento 010. . . . .	41
5.12. MLP: Curva lift experimento 010. . . . .	41
5.13. MLP: Curva de calibración experimento 010. . . . .	42
5.14. Fuzzy: Matriz de confusión modelo lógica difusa. . . . .	43
5.15. Fuzzy: Curva ROC modelo de lógica difusa. . . . .	43

## ÍNDICE DE FIGURAS

---

5.16. Modelo Neuro-difuso. . . . .	44
6.1. Sistemas guiados por ingeniería del conocimiento. . . . .	48

# Índice de tablas

4.1. Grado de pertenencia a conjuntos borrosos . . . . .	19
5.1. Distribución inicial de muestras . . . . .	27
5.2. Distribución final de muestras . . . . .	27
5.3. Reglas motor de inferencia difuso . . . . .	35
5.4. Reglas motor de inferencia difuso . . . . .	36

# Índice de algoritmos

1.	Llamado <i>entrenamiento red neurona MLP</i> . . . . .	29
2.	Llamado <i>entrenamiento red neuronal AE</i> . . . . .	32
3.	Algoritmo <i>Imputación valores áтипicos</i> . . . . .	54
4.	Algoritmo <i>Construcción de la arquitectura MLP</i> . . . . .	54
5.	Algoritmo <i>Entrenamiento modelo MLP</i> . . . . .	55
6.	Algoritmo <i>Construcción de la arquitectura AE</i> . . . . .	56
7.	Algoritmo <i>Importar librerías SKfuzzy</i> . . . . .	57
8.	Algoritmo <i>Definición conjuntos difusos</i> . . . . .	57
9.	Algoritmo <i>Reglas del motor de inferencia</i> . . . . .	58
10.	Algoritmo <i>Simulación del motor de inferencia</i> . . . . .	59

# Resumen

En este trabajo de grado, se detalla la creación de un sistema neuro-difuso para la detección de comercios fachada en el contexto de una pasarela de pagos. La solución de este desafío se sitúa en el campo de la inteligencia artificial, donde se combinan técnicas de aprendizaje basado en datos con modelos de ingeniería del conocimiento. Además se emplean tecnologías más 'cognitivas' y menos 'numéricas', como el Soft Computing [Olivas et al. \(2023\)](#). Esta combinación permite aprovechar tanto los datos masivos como el conocimiento experto, lo cual resulta fundamental en la detección de fraudes. Dado que la naturaleza del fraude es variable y cambia constantemente, es esencial aprender de las mutaciones en los patrones de fraude y al mismo tiempo que es crucial anticiparse a estos.

Esta investigación fue diseñada considerando dos facetas fundamentales dentro del campo de la inteligencia artificial: el aprendizaje automático y el razonamiento aproximado. Estas dos líneas de estudio representan pilares esenciales en la evolución y aplicación de la inteligencia artificial en una amplia variedad de campos.

Dentro del enfoque en el **aprendizaje automático**, se llevó a cabo una evaluación exhaustiva que se centró en la creación de modelos de aprendizaje supervisado. Este enfoque implica el uso de un conjunto de datos previamente recopilado que contiene información sobre los comportamientos de los comercios que habían sido previamente identificados como comercios fachada. La premisa fundamental de este proceso es utilizar la información histórica para entrenar modelos que puedan identificar futuros casos de comercios fachada con precisión.

En esta fase inicial, se puso gran énfasis en la evaluación de atributos. Lo anterior implica una cuidadosa revisión de los datos disponibles, donde se examinó la calidad de los datos para garantizar su integridad y confiabilidad. Además, se exploró la dependencia entre diferentes variables, analizando cómo influyen unas sobre otras. La correlación entre estas variables también fue objeto de un análisis, lo que permitió comprender mejor las relaciones subyacentes en los datos.

Para la construcción de nuestros modelos, optamos por utilizar varios algoritmos de aprendizaje automático. Entre ellos, destacan las redes neuronales de perceptrón multicapa, que son conocidas por su capacidad para modelar relaciones complejas en los datos. También exploramos técnicas de aprendizaje no supervisado *autocoders* que se caracterizan por su eficacia

en la identificación de patrones latentes y la generación de representaciones comprimidas de los datos.

Dentro del enfoque de **razonamiento aproximado** se realizó la creación de un modelo **lógica borrosa** en el cual decidimos centrarnos en seis atributos clave que hemos denominado por confidencialidad y de manera abreviada como V1, V2, ... V6 a lo largo de esta investigación. Estos atributos son esenciales para el análisis y representan tasas que relacionan dos variables relevantes en este contexto. Cabe destacar que la definición de estos atributos se basó en el criterio de expertos en el campo, lo que aseguró que estuvieran alineados con las necesidades y particularidades de la investigación.

Finalmente, se llevó a cabo una evaluación exhaustiva de ambos modelos, arrojando resultados altamente satisfactorios. La fusión de estos modelos en uno solo marcó un hito importante, ya que permitió aprovechar al máximo las ventajas que ambos escenarios ofrecían. Por un lado, se pudo capitalizar el conocimiento experto en detección de fraudes, modelando y aplicando reglas de lógica difusa para una toma de decisiones inteligente. Por otro lado, se logró sacar partido de la capacidad de aprendizaje y adaptación de los modelos de redes neuronales, los cuales explotaron la información contenida en los datos para identificar patrones de comportamiento de los comercios en un entorno de constante cambio.

# Introducción

# 1

En el presente período de constante crecimiento en digitalización y el comercio electrónico, las transacciones financieras en línea han experimentado un aumento exponencial. Sin embargo, este aumento en la actividad comercial en línea también ha dado lugar a la proliferación de actividades fraudulentas, como el establecimiento de **negocios fachada**, que buscan explotar las pasarelas de pago para actividades ilícitas. Junto con las transacciones fraudulentas, los negocios fachada representan un desafío importante para la seguridad financiera y la integridad del comercio electrónico.

Con el fin de abordar este tema crítico, el presente trabajo presenta un enfoque innovador y efectivo, basado en la combinación de dos poderosas técnicas de inteligencia artificial: la lógica difusa y las redes neuronales. La lógica difusa ofrece la capacidad de modelar y manejar la incertidumbre inherente en la identificación de patrones sutiles y cambiantes, mientras que las redes neuronales han demostrado su destreza en el procesamiento y reconocimiento de patrones complejos en grandes conjuntos de datos.

En esa medida, el objetivo fundamental de esta investigación es desarrollar un sistema avanzado de detección de negocios fachada en pasarelas de pago mediante la fusión de la precisión adaptativa de la lógica difusa y la capacidad de aprendizaje profundo de las redes neuronales. Al combinar estas dos metodologías, se pretende mejorar significativamente la precisión y la eficacia de la identificación de actividades fraudulentas, brindando así una mayor confianza y seguridad en el entorno de las transacciones en línea.

Finalmente, esta investigación busca contribuir al campo de la inteligencia artificial aplicada a la seguridad financiera, al proporcionar un enfoque innovador y efectivo para la identificación de negocios fachada en pasarelas de pago. La combinación de lógica difusa y redes neuronales tiene el potencial de revolucionar la forma en que se abordan los desafíos de seguridad en el comercio electrónico, y este trabajo se propone explorar y demostrar ese potencial.

## 1.1. Pasarelas de pago

Una **pasarela de pago** es una plataforma de tecnología diseñada para facilitar y gestionar transacciones financieras en línea, estas permiten a los comercios electrónicos captar pagos por diferentes medios de forma segura y eficiente; para esto, la pasarela se encarga de todo el

proceso referente a la transacción, como lo es el uso de procesadores de pago, encriptación de los datos y uso como lo son motores de detección de fraudes, todo con el fin de brindar una respuesta de autorización o rechazo de la transacción hacia el comercio.

Las pasarelas de pago disponen de diversos medios de pagos con el objetivo de brindar diferentes opciones para que los clientes realicen compras en los comercios; en ese sentido, la pasarela de pago se encarga de procesar los diferentes medios de pago, utilizando para ello otros procesadores de pago especializados y canales del banco adquirente, evitando así que el comercio tenga que realizar extensas configuraciones para ofrecer de diferentes formas de pago a los clientes.

Dentro de las opciones de pagos que ofrece una pasarela de pago se encuentran: las tarjetas de crédito y débito, transferencias bancarias, billeteras e incluso algunos no digitales como son depósito en puntos físicos y depósito de cheques. Algunos de estos métodos de pago, por su naturaleza son más susceptibles de ser utilizados por defraudadores, siendo las tarjetas de crédito el método más afectado.

## 1.2. Negocios Fachada

Los negocios fachada, son comercios que en la realidad no existen y que se crean dentro de una pasarela de pago con el objetivo de captar transacciones fraudulentas, especialmente mediante tarjetas de crédito hurtadas. Esto se hace con el fin acreditar en una cuenta (la cuenta del negocio fachada), dinero líquido y disponible a partir de tarjetas de crédito.

Los comportamientos de estos negocios presentan particularidades en varios aspectos que vamos a entrar a medir, estos aspectos estarán denominados como: V1, V2, V3 ... las cuales serán variables enmascaradas que darán cuenta de comportamientos importantes a medir de un comercio fachada, pero que por motivos de seguridad no pueden ser expresados aquí.

# Objetivos

# 2

El objetivo de este trabajo de fin de máster es realizar un estudio exhaustivo y complementario de dos paradigmas fundamentales en el campo de la inteligencia artificial: el aprendizaje automático y el razonamiento aproximado. Estos dos enfoques representan dos vertientes poderosas y versátiles de la IA, encarnadas respectivamente en las redes neuronales y en el control borroso, específicamente en el marco del modelo Mamdani.

La motivación detrás de esta investigación es comprender en profundidad cómo estos dos paradigmas pueden aplicarse de manera efectiva y complementaria en diversas aplicaciones del mundo real. En particular, se busca explorar cómo las redes neuronales, con su capacidad para el aprendizaje a partir de datos y la resolución de problemas complejos, pueden combinarse de manera sinérgica con el control borroso, que se destaca en situaciones en las que la incertidumbre y la imprecisión son características inherentes.

Además, este trabajo de investigación tiene como objetivo proporcionar pautas y recomendaciones para la selección y aplicación adecuada de redes neuronales y control borroso en diferentes contextos. Ya que en este se enumeran las ventajas y desventajas de cada sistema y como su combinación puede potenciar los resultados en el marco del *soft computing*.

- 1. Objetivo parcial 1.** Creación de un modelo de inteligencia artificial con la capacidad de clasificar si un negocio es ilegal, basado en sus comportamientos y minimizando los falsos positivos de los modelos anteriores.
- 2. Objetivo parcial 2.** Realizar la comparación entre técnicas de aprendizaje automático y lógica difusa, con el fin para medir sus resultados y tiempos de entrenamiento.
- 3. Objetivo parcial 3.** Explora la creación de un modelo que combine lógica difusa y redes neuronales artificiales.



# Redes Neuronales Artificiales

3

## 3.1. Historia

Una Red Neuronal Artificial(RNA), es un concepto inspirado tanto en la estructura como en el funcionamiento del cerebro humano, su desarrollo teórico comenzó en los años 40 cuando el sicólogo Donald Hebb creó una hipótesis de aprendizaje al que hoy se conoce como aprendizaje Hebb, el cual, en su forma más básica, se resume en la frase "células que se activan juntas, se conectan entre sí". En otras palabras, cuando dos neuronas se activan simultáneamente, la conexión entre ellas se refuerza, lo que facilita la comunicación entre esas neuronas en el futuro. Si una neurona A está conectada a una neurona B y ambas se activan al mismo tiempo, la conexión entre A y B se fortalece. Este tipo de aprendizaje puede ser considerado como a lo que hoy llamamos aprendizaje no supervisado porque no hay un agente externo que indique qué respuestas son correctas o incorrectas. En lugar de eso, el sistema simplemente "aprende" patrones y relaciones en los datos sin que se le proporcionen etiquetas de destino. (Acevedo et al., 2017).

El concepto de redes neuronales artificiales se origina con los trabajos de Warren McCulloch y Walter Pitts, quienes publicaron un artículo en 1943 que presentaba una representación simplificada de una neurona biológica y cómo podría simularse computacionalmente. Esto sentó las bases teóricas para las redes neuronales (Matich, 2001).

Década de 1960 y 1970: Se desarrollaron los primeros modelos computacionales de aprendizaje automático y redes neuronales, como el perceptrón, propuesto por Frank Rosenblatt en 1957. Sin embargo, las limitaciones de estos modelos y la falta de capacidad computacional y algunas limitaciones al no poder resolver problemas no lineales, limitaron su avance en ese momento (Matich, 2001). En 1969 Minsky y Papera, del Instituto Tecnológico de Massachusetts (MIT), publicaron un libro Perceptrons. En donde se demuestra matemáticamente que el Perceptrón no era capaz de encontrar la solución a problemas sencillos, como el aprendizaje de una función no-lineal. Este hecho frena la investigación de redes neuronales hasta la década de 1980 (Newell, 1969).

En 1986 se redescubre el algoritmo de *backpropagation* y a partir de este momento el panorama para la investigación y desarrollo de redes neuronales fue más alentador haciendo que en la década de 1990, las redes neuronales resurgen gracias a investigaciones que demuestran que los modelos más complejos y las técnicas de entrenamiento adecuadas pueden

mejorar significativamente su desempeño. Además, se desarrolla el algoritmo de retropropagación (backpropagation), que permite ajustar los pesos de las conexiones en función del error en la salida, lo que facilita el entrenamiento de redes profundas ([Acevedo et al., 2017](#)).

Década de 2000: Se producen avances significativos en el campo de las redes neuronales, como las redes neuronales convolucionales (CNN) para el procesamiento de imágenes y las redes neuronales recurrentes (RNN) para el procesamiento de secuencias. Estos avances permiten aplicaciones exitosas en reconocimiento de patrones y procesamiento del lenguaje natural ([Cetinic et al., 2020](#)).

Década de 2010 en adelante: La década actual ha sido testigo de un renacimiento de las redes neuronales, conocido como el auge del aprendizaje profundo. Las arquitecturas de redes neuronales profundas, como las redes neuronales convolucionales profundas y las redes neuronales recurrentes, han demostrado un rendimiento sorprendente en una amplia gama de tareas, incluyendo la visión por computadora, el procesamiento del lenguaje natural, la traducción automática y un nuevo hito que son las redes neuronales generativas, las cuales han permitido pasar de problemas de clasificación o regresión a generar nuevo contenido. En la sección siguiente se describirán algunos de los principales tipos de redes neuronales.

### 3.2. Perceptrón Simple

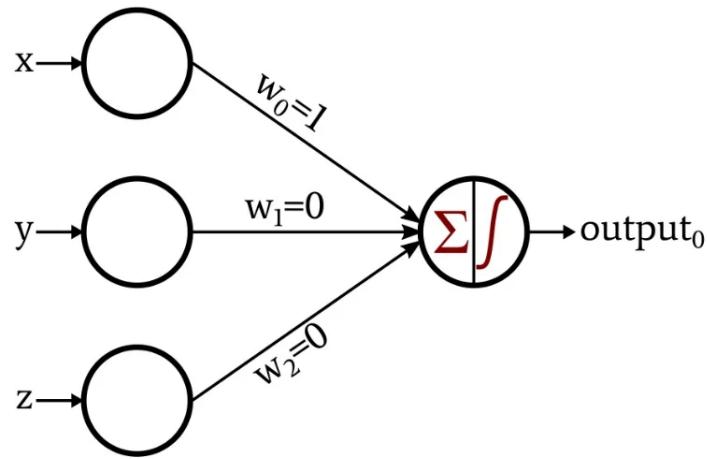
El perceptrón es la unidad básica de una red neuronal. Consiste en una sola capa de neuronas que realiza una combinación lineal de las entradas y aplica una función de activación. Si bien el perceptrón originalmente tenía limitaciones en términos de su capacidad para resolver problemas no lineales, su desarrollo sentó las bases para el estudio de redes neuronales más complejas y poderosas, como las redes neuronales multicapa. Estas redes, que constan de múltiples capas de perceptrones interconectados, pueden abordar problemas más sofisticados y aprender representaciones jerárquicas de datos [Rumelhart \(1986\)](#).

El perceptrón, a pesar de su simplicidad, sigue siendo un componente fundamental en el campo de la inteligencia artificial y el aprendizaje automático, y su comprensión es esencial para el estudio de redes neuronales más avanzadas y poderosas.

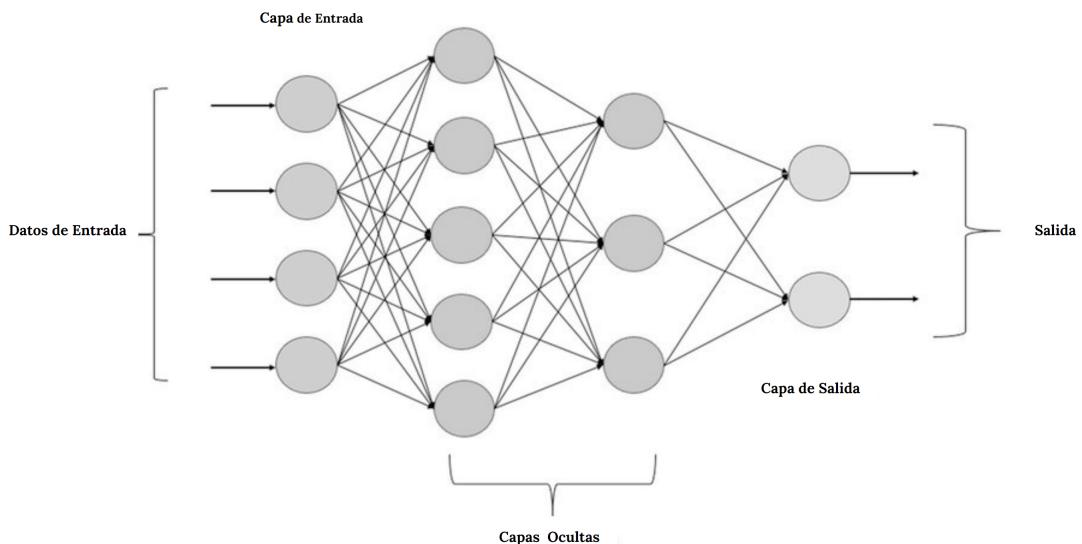
### 3.3. Perceptrón Multicapa (MLP)

Un perceptrón multicapa, también conocido como red neuronal multicapa (MLP, por sus siglas en inglés), representa una forma avanzada de red neuronal artificial, la cual consiste en múltiples capas de neuronas interconectadas, lo que incluye una capa de entrada, una o más capas ocultas y una capa de salida. Esta arquitectura, desarrollada a lo largo de los años a partir de los conceptos iniciales del perceptrón de una sola capa, ha demostrado ser una herramienta esencial en el campo de la inteligencia artificial y el aprendizaje automático [Rumelhart \(1986\)](#).

La característica distintiva de un MLP es su capacidad para modelar relaciones y patrones complejos en los datos. Cada neurona en una capa se conecta a todas las neuronas en la

**Figura 3.1: Perceptrón simple.**

capa siguiente, lo que permite la transferencia de información a través de múltiples niveles de procesamiento. Esta estructura jerárquica facilita la captura de representaciones de alto nivel de los datos de entrada [LeCun et al. \(2015\)](#).

**Figura 3.2: Perceptrón multicapa.**

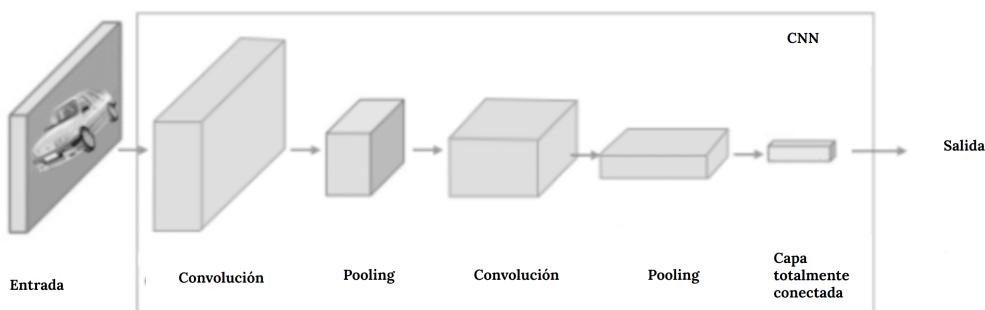
### 3.4. Redes Neuronales Convolucionales (CNN)

Las redes neuronales convolucionales (CNN) son una clase especializada de redes neuronales profundas diseñadas específicamente para procesar datos con una estructura de cuadrícula, como imágenes, secuencias de video y datos espaciales en general [LeCun et al. \(2015\)](#). A diferencia de las redes neuronales tradicionales, las CNN incorporan dos tipos clave de capas: capas convolucionales y capas de agrupación, que les permiten realizar una extracción

de características altamente eficiente y mantener la estructura espacial de los datos de entrada [Krizhevsky et al. \(2012\)](#).

Las capas convolucionales de una CNN están diseñadas para detectar características locales en los datos de entrada, como bordes, texturas o patrones específicos. Estas capas utilizan filtros (kernels) que se deslizan a lo largo de la cuadrícula de entrada y calculan productos escalares locales, lo que permite identificar características relevantes en diferentes regiones de la imagen [LeCun et al. \(1998\)](#).

Además, las capas de agrupación, también conocidas como capas de pooling, se utilizan para reducir la dimensionalidad de los datos y conservar la información más importante. Esto hace que las CNN sean especialmente adecuadas para el procesamiento de imágenes de alta resolución, ya que reducen la cantidad de información a procesar sin perder información crítica [Boureau et al. \(2010\)](#).



**Figura 3.3: Redes neuronales convolucionales(CNN).**

### 3.5. Redes Neuronales Recurrentes (RNN)

Las redes neuronales recurrentes (RNN) son un tipo especializado de redes neuronales diseñadas específicamente para procesar datos secuenciales o de series temporales. A diferencia de las redes neuronales feedforward tradicionales, las RNN incorporan conexiones retroalimentadas, lo que les permite mantener información en estados anteriores y utilizarla para tomar decisiones en estados futuros. Esta capacidad de mantener una 'memoria' interna las hace excepcionalmente adecuadas para tareas que implican secuencias de datos, donde la relación entre los elementos de la secuencia es fundamental [Hochreiter y Schmidhuber \(1997\)](#).

Un aspecto distintivo de las RNN es su capacidad para capturar dependencias a lo largo del tiempo en los datos de entrada. Cada neurona en una RNN está conectada a sí misma en el tiempo anterior, lo que significa que pueden aprender y recordar patrones y relaciones a medida que se desarrolla una secuencia. Esto hace que las RNN sean especialmente útiles en aplicaciones como el procesamiento del lenguaje natural (NLP), donde se deben comprender y generar texto coherente, así como en la predicción de series temporales, donde se busca prever valores futuros en función de datos pasados.

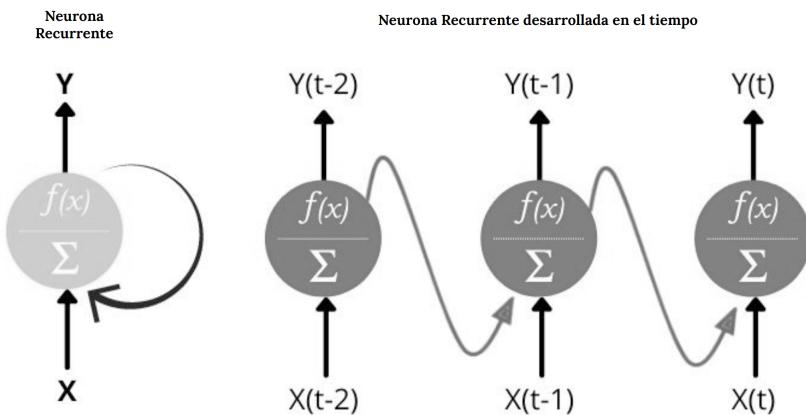


Figura 3.4: *Redes neuronales recurrentes (RNN).*

### 3.6. Redes Neuronales de Memoria a Corto y Largo Plazo (LSTM)

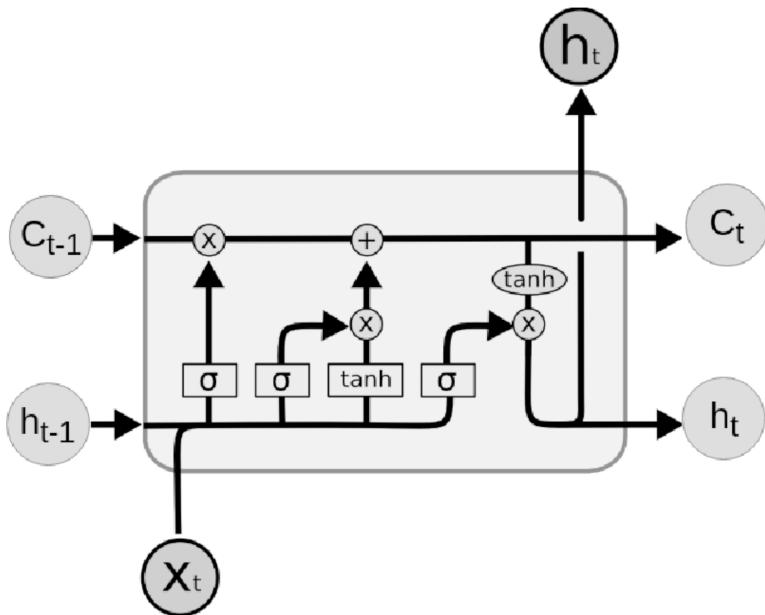
Las Redes Neuronales de Memoria a Corto y Largo Plazo (LSTM) representan una variante avanzada y altamente efectiva de las Redes Neuronales Recurrentes (RNN) diseñada para abordar uno de los desafíos fundamentales en el procesamiento de secuencias: el problema de desvanecimiento del gradiente [Hochreiter y Schmidhuber \(1997\)](#). A medida que las RNN procesan secuencias largas, las actualizaciones de los pesos pueden volverse extremadamente pequeñas o grandes, lo que dificulta el aprendizaje efectivo en secuencias largas. Las LSTM fueron propuestas como una solución a este problema crucial.

La característica distintiva de las LSTM es su arquitectura interna que incluye celdas de memoria y puertas de regulación del flujo de información. Estas celdas de memoria pueden mantener y actualizar información durante largos períodos de tiempo, lo que les permite capturar dependencias a largo plazo en secuencias de datos. Las puertas en las LSTM, controlan cuándo y qué información se almacena o se descarta en la memoria de la celda, lo que las hace altamente eficientes para modelar patrones en secuencias largas.

Las LSTM han demostrado ser particularmente útiles en una amplia variedad de aplicaciones, desde el procesamiento de lenguaje natural (NLP) hasta el reconocimiento de voz y la traducción automática. Su capacidad para capturar relaciones a largo plazo las hace ideales para tareas que involucran secuencias complejas de datos, como la generación de texto coherente o la predicción precisa de valores en series temporales.

### 3.7. Redes Neuronales Generativas Adversarias (GAN)

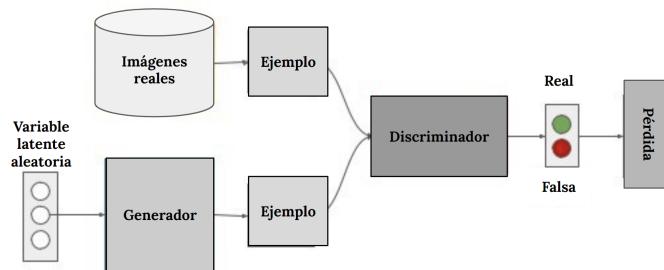
Las Redes Neuronales Generativas Adversarias (GAN) representan una innovadora arquitectura de redes neuronales diseñada para la generación de datos realistas y de alta calidad [Goodfellow et al. \(2014\)](#). Consiste dos redes, un generador y un discriminador, que compiten



**Figura 3.5: Redes neuronales de memoria a corto y largo plazo (LSTM).**

entre sí. El generador crea datos falsos y el discriminador intenta distinguir entre datos reales y falsos. Esta competencia mejora la calidad de los datos generados.

El proceso de entrenamiento de una GAN se asemeja a un juego de suma cero, donde el generador y el discriminador compiten. A medida que el generador mejora su capacidad para engañar al discriminador, el discriminador se vuelve más experto en distinguir datos reales de los falsos. Esta competencia dinámica conduce a la generación de datos sintéticos de alta calidad que pueden ser indistinguibles de los datos reales [Salimans et al. (2016)].



**Figura 3.6: Redes neuronales generativas adversarias (GAN).**

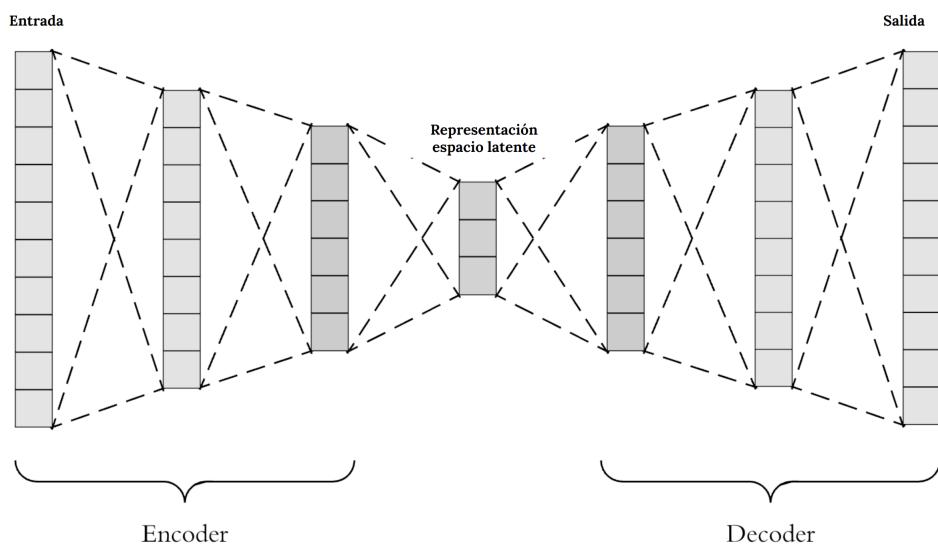
## 3.8. Autoencoders

Los autoencoders son una clase de redes neuronales artificiales diseñadas para aprender representaciones eficientes de datos de entrada Hinton y Salakhutdinov (2006). Su arquitectura se compone de tres capas principales: una capa de entrada, una capa oculta (también llamada capa latente o espacio latente) y una capa de salida. La capa de entrada recibe datos sin

procesar, mientras que la capa de salida intenta reconstruir los datos de entrada. La capa oculta en el medio actúa como una versión comprimida o codificación de los datos de entrada.

La idea fundamental detrás de los autoencoders es aprender una representación compacta y significativa de los datos. Durante el entrenamiento, la red busca minimizar la diferencia entre los datos de entrada y los datos de salida generados. Esto obliga a la capa oculta a aprender características relevantes y descartar información redundante o ruido en los datos de entrada. Como resultado, los autoencoders son particularmente efectivos en la reducción de la dimensionalidad de los datos.

Los autoencoders se utilizan en una variedad de aplicaciones, incluida la reducción de ruido de imágenes, la compresión de datos, la generación de datos sintéticos y la extracción de características relevantes para tareas de clasificación y reconocimiento de patrones. .



**Figura 3.7: Autoencoder.**



# Lógica Borrosa

# 4

## 4.1. Conceptos Básicos y Definiciones

La Lógica Difusa fue introducida en 1965 por el ingeniero azerbaiyano, Lotfi Asker Zadeh, guiado por el principio de que las matemáticas pueden ser usadas para encadenar el lenguaje con la inteligencia humana. La teoría de conjuntos difusos es un intento de desarrollar una serie de conceptos para tratar de un modo sistemático el tipo de imprecisión que aparece cuando los límites de las clases de objetos no están claramente definidos. ([Astocondor Villar, 2011](#))

La lógica difusa es una extensión de la lógica clásica (binaria, verdadero/falso) que permite manejar la incertidumbre y la imprecisión en la toma de decisiones. A diferencia de la lógica clásica, donde una proposición es verdadera o falsa de manera absoluta, en la lógica difusa los valores de verdad pueden variar entre completamente verdadero y completamente falso, pasando por diferentes grados de verdad en el medio. La lógica difusa es especialmente útil para modelar y razonar sobre conceptos y situaciones que no son claramente definibles y pueden tener diferentes grados de pertenencia. Un ejemplo común es el término <sup>alto.</sup><sub>en</sub> "bajo." en relación con una temperatura. En lugar de definir arbitrariamente un umbral rígido para lo que se considera <sup>alto.</sup><sub>en</sub> "bajo", la lógica difusa permite representar esto de manera más flexible y contextual.

## 4.2. Conjuntos Difusos

Los conjuntos difusos, o conjuntos borrosos, son una extensión de los conjuntos tradicionales en matemáticas que permiten la representación de la incertidumbre y la imprecisión en la clasificación y descripción de objetos o elementos [Goguen \(1973\)](#). A diferencia de los conjuntos clásicos, donde un elemento pertenece o no pertenece completamente al conjunto, en los conjuntos difusos, los elementos pueden tener grados variables de pertenencia al conjunto, expresados mediante una función de membresía que asigna a cada elemento un valor entre 0 y 1. Lo anterior lo podemos representar de la siguiente forma.

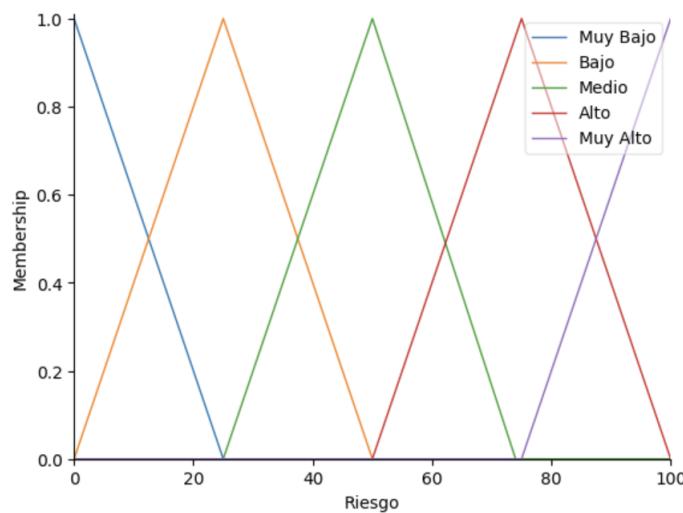
$$\mu_A = X \rightarrow [0, 1]$$

Lo anterior nos está indicando que un número puede tener un grado de pertenencia a un

conjunto borroso y no necesariamente estar completamente contenido en un conjunto, como ocurre con los conjuntos clásicos. En nuestros ejemplos es posible que un comercio tenga un grado de pertenencia de 0.7 al el conjunto de comercios de alto riesgo y un grado de pertenencia de 0.5 al conjunto de comercios de riesgo medio. Es de notar que la suma de grados de pertenencia no es necesariamente 1, lo que nos indica que este valor no corresponde a una probabilidad.

Es importante notar la diferencia entre probabilidad y grado de pertenencia, mientras que la **probabilidad** se utiliza en situaciones de incertidumbre para cuantificar la posibilidad de que un evento ocurra o no, el **grado de pertenencia** se emplea en la teoría de conjuntos difusos para representar cuán bien un elemento se ajusta a una categorización gradual o difusa en un conjunto. Ambos conceptos están relacionados con la incertidumbre, pero se aplican en contextos y marcos conceptuales diferentes.

El siguiente conjunto de términos pretende reflejar el nivel de riesgo de un comercio.



**Figura 4.1: Conjunto borroso comercios riesgo.**

Para un puntaje obtenido por un comercio entre 0 y 100 se define un grado pertenencia a cada uno de los términos: Muy bajo, Bajo, Medio, Alto y Muy alto.

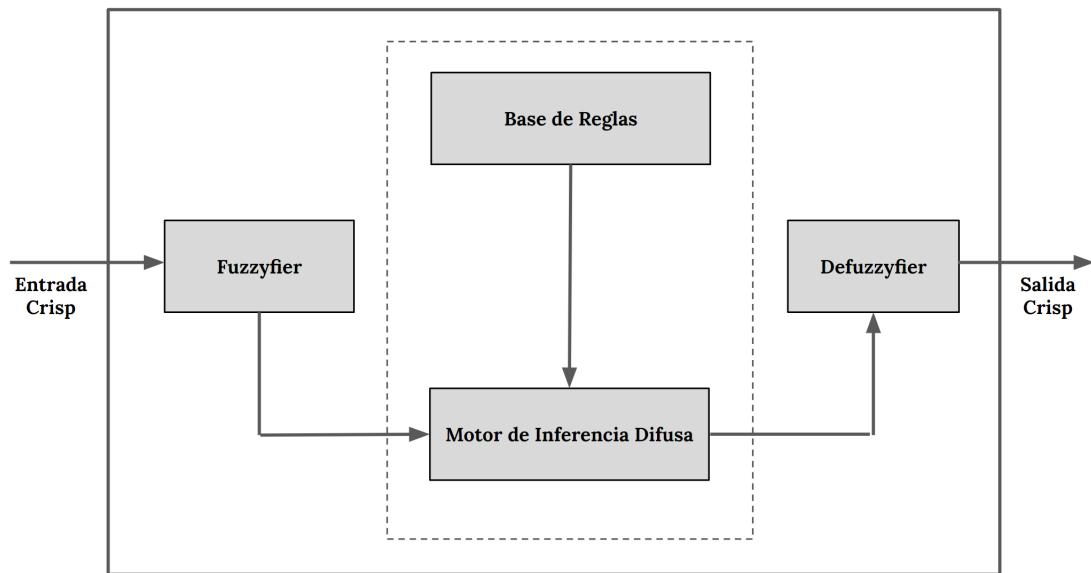
### 4.3. Control Difuso

Actualmente, una de las áreas de aplicación con mayor relevancia dentro de la teoría de conjuntos difusos la componen los Sistemas Basados en Reglas Difusas (SBRDs). En lugar de utilizar reglas de control rígidas y valores numéricos precisos, el control difuso utiliza conjuntos difusos y reglas lingüísticas para representar y gestionar la incertidumbre y la imprecisión en los sistemas de control.

En el control difuso se utilizan reglas tipo "SI-ENTONCES"(IF-THEN) en donde se cuentan con antecedentes y consecuentes, los cuales forman proposiciones difusas. Para modelar

estas proposiciones, se requiere de una **base de conocimiento** la cual representa todo el conocimiento que tenemos del sistema. Esta habitualmente está construida a partir del conocimiento de un experto, el cual se encarga de establecer las relaciones entre los antecedentes y los consecuentes. La base de conocimiento está compuesta por:

- **Base de Reglas**, el cual contiene las reglas difusas del sistema
- **Base de Datos**, la cual persiste los términos lingüísticos y las funciones de pertenencia.



**Figura 4.2: Estructura controlador difuso.**

#### 4.4. Modelo Mamdani

El Control Borroso de Mamdani es un enfoque de control basado en la teoría de conjuntos difusos, desarrollado por Ebrahim Mamdani en la década de 1970. Su objetivo es diseñar sistemas de control capaces de manejar la incertidumbre y la imprecisión en la toma de decisiones y el control de sistemas. En este enfoque un experto plasma su conocimiento acerca del sistema en reglas lingüísticas, en donde las etiquetas lingüísticas describen los estados de la variables. A cada valor de entrada del sistema se le asigna una etiqueta lingüística, las cuales en su conjugación activan ciertas reglas, las cuales definen la salida del sistema también en términos de etiquetas.

Las reglas de un sistema Mamdani son de la forma:

- **Regla 1** Si la temperatura es alta y la humedad es baja, entonces reduce la potencia del calentador.

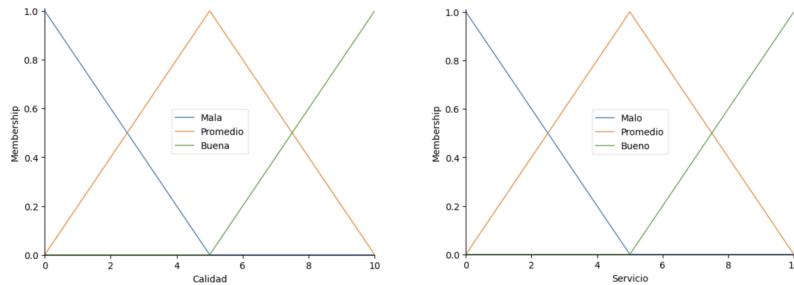
- **Regla 2** Si la temperatura es baja y la humedad es alta, entonces aumenta la potencia del calentador

El proceso de control implica la evaluación de estas reglas, en donde se examina el grado de cumplimiento de cada regla de acuerdo a la variable medida, con el fin de determinar la acción de control a tomar. Esto se hace mediante un proceso de inferencia difusa que combina las reglas y produce una salida difusa. Luego, la salida difusa se convierte en una salida real mediante una operación llamada defusificación.

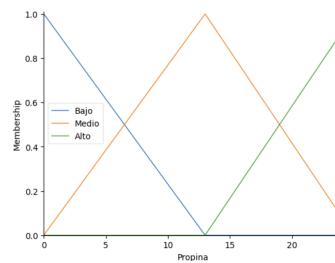
#### Ejemplo

Vamos a calcular la propina sugerida en un restaurante, teniendo en cuenta la calidad de los alimentos y el servicio prestado y para esto tenemos las siguientes reglas:

- **Regla 1**, Si el servicio es bueno y la calidad es buena, entonces la propina es alta
- **Regla 2**, Si el servicio es bueno y la calidad promedio, entonces la propina es media.
- **Regla 3**, Si el servicio es promedio y la calidad promedio, entonces la propina es media.
- **Regla 4**, Si el servicio es malo o la calidad mala, entonces la propina es baja.



**Figura 4.3: Funciones de pertenencia para calidad y servicio.**



**Figura 4.4: Funciones de pertenencia para la propina.**

Nuestra salida espera un valor de propina entre 0 y 25, lo cual corresponde al porcentaje de la cuenta que dejaremos como beneficio en el restaurante

Si suponemos que los datos medidos son calidad en la comida de 7 y un servicio de 9, obtenemos los siguientes grados de pertenencia a los conjuntos borrosos.

Calidad	Servicio
$\mu_{Mala}(7) = 0$	$\mu_{Malo}(9) = 0$
$\mu_{Promedio}(7) = 0,6$	$\mu_{Promedio}(9) = 0,2$
$\mu_{Buena}(7) = 0,4$	$\mu_{Bueno}(9) = 0,8$

Tabla 4.1: Grado de pertenencia a conjuntos borrosos.

Dado estos valores encontramos que se activan las reglas 1 y 2. Ahora vamos a evaluar las reglas:

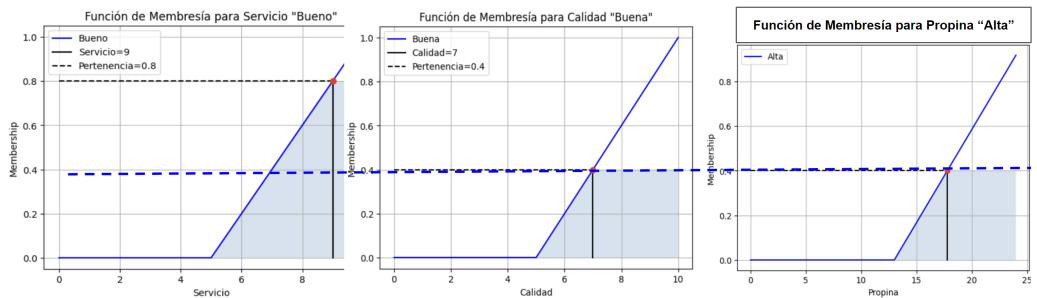


Figura 4.5: Evaluación Regla 1.

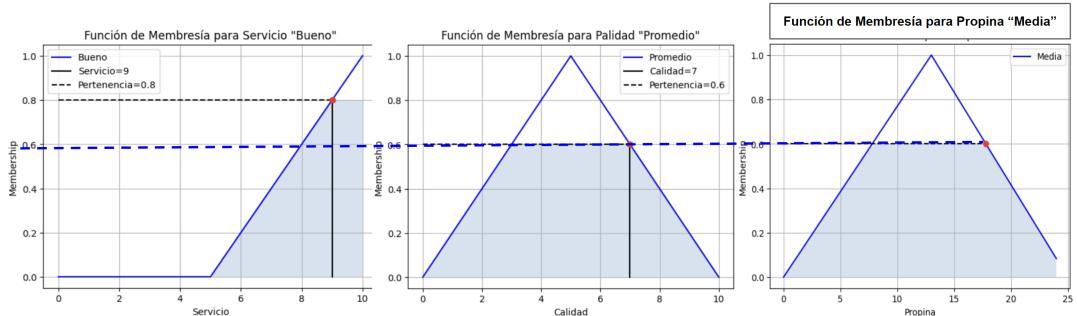


Figura 4.6: Evaluación Regla 2.

Tras la evaluación de cada regla, se combinan los dos conjuntos borrosos obtenidos de la salida de las reglas mediante la operación máximo, sería la unión de ambos conjuntos de salida.

Por lo general un sistema no es capaz de entender un conjunto borroso de salida, por lo cual es necesario realizar una **defuzzificación o desborrosificación**, para la cual se pueden utilizar varias estrategias.

**Media del máximo:** Usar el valor medio del máximo conjunto de salida, por lo cual tendríamos,  $(7.5+18.1)/2 = 12.8$ .

**Proyección del centro de gravedad:** Para esto se calcula la proyección del centro de gravedad del conjunto borroso en el eje x, para este caso el valor de salida sería 13.53.

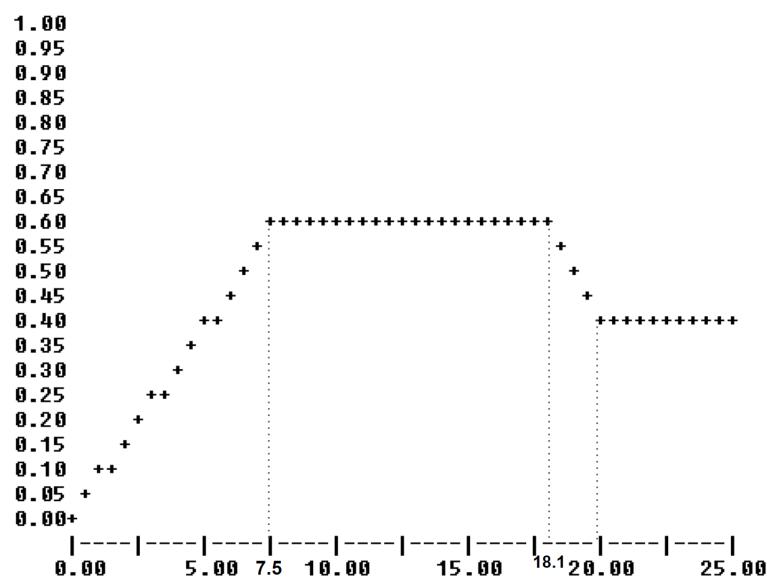


Figura 4.7: *Conjunto de salida en fuzzy clips.*

## 4.5. Modelo Takagi-Sugeno

En 1985, Takagi y Sugeno introdujeron un nuevo enfoque en la teoría del control difuso conocido como el Método Takagi-Sugeno (TS). En este se mantien la misma especificación de las particiones borrosas de los dominios de las entradas que en el modelo Mamdani, pero no se requiere una partición borrosa del dominio de salida ([Olivas, 2001](#)). Este enfoque se ha demostrado especialmente útil en la modelización de sistemas complejos que requieren mayor precisión, generalmente sistema de control físicos.

Sean  $A_i$  y  $B_i$ , con  $i = 1, 2, \dots, n$ , conjuntos difusos de nuestro sistema. Las reglas tendrían la siguiente forma:

$$R_1 : Si\ x\ es\ A_1 \wedge y\ es\ B_1\ entonces\ z = f_1(x, y)$$

$$R_2 : Si\ x\ es\ A_2 \wedge y\ es\ B_2\ entonces\ z = f_2(x, y)$$

...

$$R_n : Si\ x\ es\ A_n \wedge y\ es\ B_n\ entonces\ z = f_n(x, y)$$

La principal distinción del método TS con respecto al de Mamdani radica en la ausencia de un proceso de defuzzificación. Esto se debe a que, en lugar de obtener conjuntos difusos, generamos un conjunto de funciones lineales. Por lo tanto, en el método TS, podemos calcular directamente la salida del sistema mediante una expresión del tipo:

$$Z_0 = \frac{\sum_{i=1}^n \omega_i f_i(x_i, y_i)}{\sum_{i=1}^n \omega_i}$$

donde  $\omega_i$  se obtiene calculando el mínimo de los valores de entrada en cada regla  $R_i$ .

El sistema TS se utiliza principalmente para modelar relaciones numéricas entre variables de entrada y salida, lo que lo hace adecuado para aplicaciones donde se requiere un cálculo preciso, como el control de sistemas físicos. El sistema de Mamdani, en cambio, se utiliza para modelar relaciones difusas donde los resultados pueden ser categorías lingüísticas. Es por eso que para este ejercicio se utilizó en enfoque de Mamdani.



# Metodología

# 5

## 5.1. Diseño de la Investigación

A continuación nos sumergiremos en el proceso detallado de recopilación de datos para después pasar al análisis de atributos, donde se examinará cómo cada atributo contribuye a la capacidad del modelo para identificar comercios fachada. Luego nos centraremos en los resultados obtenidos a través de los algoritmos mencionados, evaluando su rendimiento y precisión en la identificación de comercios fachada. Este enfoque metódico nos permitirá perfeccionar aún más nuestros modelos y avanzar hacia soluciones más efectivas en la identificación de comercios fachada en una pasarela de pago.

## 5.2. Recopilación de Datos

Para llevar a cabo nuestra investigación, utilizamos un conjunto de datos reales que consta de 12.229 registros de comercios, que luego mediante un mecanismo de generación de balanceo mediante muestras sintéticas, pasaron a ser 23.020 registros. El conjunto inicial de datos contaba con un total de 15 atributos, pero en los experimentos iniciales de aprendizaje supervisado, no se obtuvieron buenos resultados, por tal motivo fue necesario hacer una re-ingeniería de atributos y llegar a 38 atributo, de los cuales finalmente quedaron 33; cada registro en el conjunto de datos está etiquetado con el riesgo medido del comercio: alto, medio y bajo. Finalmente la clasificación llevada a cabo, fue una clasificación binaria, en la cual las categorías medio y bajo quedaron catalogadas como 0 y alto como 1, por sugerencia del equipo de seguridad.

La clasificación de los comercios en estas categorías fue realizada por un equipo de seguridad, el cual lleva a cabo una revisión exhaustiva de los comercios registrados en la plataforma, un proceso que se realiza de manera manual y minuciosa, sobre el cual es importante destacar que esta revisión manual es un proceso laborioso y que consume mucho tiempo.

## 5.3. Preprocesamiento de Datos

En el proceso de preparación de los datos, se llevaron a cabo varias etapas críticas para garantizar la calidad y la idoneidad de los datos utilizados en los modelos de análisis. A conti-

nuación, se describen detalladamente los procedimientos de selección de atributos, imputación de datos faltantes y normalización de los datos, destacando su importancia en la creación de un conjunto de datos coherente y confiable para su posterior análisis.

### 5.3.1. Selección de atributos

Para la selección de atributos, se llevó a cabo un proceso exhaustivo de evaluación de la capacidad discriminativa de cada atributo con respecto a su clase correspondiente. Este proceso se realizó en varias etapas, utilizando tanto análisis gráficos como pruebas estadísticas rigurosas. A continuación, se describen las pruebas utilizadas en este proceso junto con sus definiciones y referencias relevantes:

**1. Análisis Gráfico de Atributos:** En una primera fase, se llevó a cabo un análisis visual de los atributos mediante representaciones gráficas, como histogramas y gráficos de caja. Estos gráficos proporcionaron una visión inicial de la distribución de los datos y posibles diferencias entre las clases. Sin embargo, para una evaluación más precisa, se realizaron pruebas estadísticas.

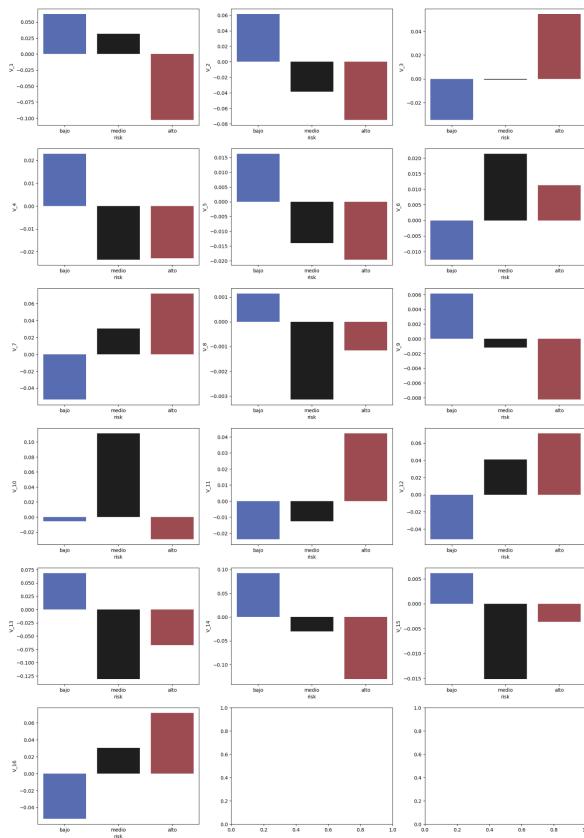


Figura 5.1: Capacidad discriminativa de los atributos.

**2. Prueba de Kolmogorov-Smirnov:** El test de Kolmogorov-Smirnov es una prueba no paramétrica utilizada para determinar si una muestra sigue una distribución normal. En el contexto de la selección de atributos, esta prueba se aplicó para evaluar si la distribución de cada

atributo se ajusta a una distribución normal. La hipótesis nula es que los datos siguen una distribución normal. Si el valor p resultante es significativo (menor que un umbral predefinido), se rechaza la hipótesis nula, lo que indica que el atributo no sigue una distribución normal. (Kolmogorov, A., Smirnov, N. (1933))

**3. Prueba de Mann-Whitney U:** Cuando se determinó que un atributo no sigue una distribución normal (según el resultado del test de Kolmogorov-Smirnov), se aplicó la prueba de Mann-Whitney U. Esta prueba es una prueba no paramétrica utilizada para comparar dos muestras independientes y evaluar si sus distribuciones difieren significativamente. En el contexto de selección de atributos, se utilizó para comparar las distribuciones de los valores de un atributo entre las diferentes clases. (Mann, H. B., Whitney, D. R. (1947)).

**4. T-Test de Medias:** Cuando se determinó que un atributo sigue una distribución normal (según el resultado del test de Kolmogorov-Smirnov), se aplicó un T-test de medias. Esta prueba estadística se utiliza para comparar las medias de dos grupos y determinar si existen diferencias significativas entre ellos. En el contexto de selección de atributos, se empleó para evaluar si las medias de los valores de un atributo difieren significativamente entre las diferentes clases.

Estas pruebas estadísticas desempeñaron un papel fundamental en la selección de atributos, permitiendo identificar aquellos atributos que demostraron una mayor capacidad discriminativa con respecto a las clases objetivo. El enfoque combinado de análisis gráfico y pruebas estadísticas proporcionó una evaluación sólida y confiable de la relevancia de cada atributo en el contexto de la tarea de clasificación. Todo esto permitió pasar de 38 a 33 atributos finales.

### 5.3.2. Imputación de valores atípicos

En el proceso de preparación de datos, se llevó a cabo una fase para la identificación y posterior tratamiento de los valores atípicos, también conocidos como outliers. Dado que los outliers son observaciones que se alejan significativamente de la tendencia general de un conjunto de datos y pueden distorsionar los resultados de análisis y modelado, es importante abordarlos en los algoritmos de aprendizaje automático y también en los modelos de inferencia de lógica borrosa, como veremos más adelante.

**Cálculo del Z-Score:** En primera instancia, se utilizó el estadístico del z-score para evaluar la distancia entre cada valor en las columnas seleccionadas y la media de la muestra correspondiente. El z-score es una medida que cuantifica cuántas desviaciones estándar un valor particular se encuentra alejado de la media. Esto proporciona una medida estandarizada de la distancia relativa de cada punto de datos con respecto a la tendencia central del conjunto de datos. Si un valor tiene un z-score alto, significa que está considerablemente alejado de la media y podría considerarse un outlier potencial.

**Establecimiento de un Umbral:** Para determinar qué valores se consideran outliers, se estableció un umbral predefinido, que en este caso se fijó en un valor de umbral de 2. Esto significa que cualquier valor con un z-score superior a 2 o inferior a -2 se considera un candidato a outlier.

**Identificación de Outliers:** Luego de calcular los z-scores, se identificaron los valores que superaban este umbral en cada columna seleccionada. Estos valores se consideraron como candidatos a outliers y requerían un tratamiento especial.

**Imputación de Outliers:** Para tratar los outliers identificados, se aplicó un proceso de imputación. En lugar de eliminar estos valores, se optó por imputarlos, lo que significa reemplazarlos por valores aceptables y dentro de límites razonables. Los valores atípicos superiores al umbral se imputaron con el valor máximo aceptado dentro del umbral, mientras que los valores atípicos inferiores al umbral se imputaron con el valor mínimo aceptado dentro del umbral. Esto aseguró que los valores atípicos fueran corregidos sin eliminar información valiosa del conjunto de datos A.

**Imputación de Outliers:** Para tratar los outliers identificados, se aplicó un proceso de imputación. En lugar de eliminar estos valores, se optó por imputarlos, lo que significa reemplazarlos por valores aceptables y dentro de límites razonables. Los valores atípicos superiores al umbral se imputaron con el valor máximo aceptado dentro del umbral, mientras que los valores atípicos inferiores al umbral se imputaron con el valor mínimo aceptado dentro del umbral. Esto aseguró que los valores atípicos fueran corregidos sin eliminar información valiosa del conjunto de datos.

**Beneficios del Enfoque:** Este enfoque de identificación y tratamiento de outliers basado en el z-score y la imputación permitió mantener la integridad de los datos y al mismo tiempo garantizar que los valores extremos no distorsionaran los resultados del análisis posterior.

### 5.3.3. Normalización de los atributos

La normalización es un proceso en el cual se transforman las características o variables de un conjunto de datos para que tengan una escala común o estandarizada. El objetivo principal de la estandarización es garantizar que todas las características tengan la misma influencia en los modelos de aprendizaje automático y que ninguna de ellas domine debido a sus unidades o escalas originales.

$$\mathbf{X} = (\mathbf{X} - \mu) / \sigma$$

### 5.3.4. Generación de muestras sintéticas y balanceo de clases

Para abordar el desbalanceo de clases en el conjunto de datos inicial, se implementó una estrategia de balanceo mediante la generación de muestras sintéticas. Esta técnica es llevada a cabo utilizando la biblioteca SMOTE de Python. Como resultado de este proceso, el conjunto de datos original, que inicialmente contenía 12,229 muestras, se expandió a un total de 23,020 muestras. Esta estrategia de aumento de datos contribuyó significativamente a mejorar el rendimiento de los entrenamientos y a equilibrar las clases en el conjunto de datos.

Tipo	Etiqueta	Cantidad de Comercios
Negocio Fachada	1	719
Negocio Legal	0	11.510
<b>Total</b>		<b>12.229</b>

**Tabla 5.1: Distribución inicial de muestras.**

Tipo	Etiqueta	Cantidad de Comercios
Negocio Fachada	1	11.510
Negocio Legal	0	11.510
<b>Total</b>		<b>23.020</b>

**Tabla 5.2: Distribución final de muestras.**

## 5.4. Construcción de Modelos

### 5.4.1. Modelos de redes neuronales

Para llevar a cabo los experimentos con redes neuronales, se diseñó y construyó un modelo generador de experimentos personalizado para perceptrón multicapa y otro para autoencoders. Estos modelos se basan en una clase que acepta una serie de parámetros, los cuales son necesarios para definir y configurar cada experimento. Entre los parámetros clave se incluyen la cantidad de neuronas en cada red neuronal, así como la opción de especificar regularizadores de cada capa en caso de ser necesario. Este enfoque modular permite una gran flexibilidad y adaptabilidad a diferentes escenarios de experimentación.

Además de permitir la configuración detallada de cada experimento, el modelo generador también desempeña un papel importante en el almacenamiento de información de cada experimento. Para cada uno se registran todas las características y especificaciones del modelo, como la arquitectura de las capas, los hiperparámetros utilizados y los resultados obtenidos durante las fases de entrenamiento y validación. Esto proporciona un registro completo y organizado de cada prueba, lo que resulta fundamental para un análisis detallado y una comparación efectiva de los resultados.

#### Perceptrón Multicapa

El primer enfoque de redes neuronales, se utilizaron una arquitectura de perceptrón multicapa, mediante la clase "PercetronMulticapa", la cual se encarga de generar los experimentos. El método "build-A" se encarga de construir la arquitectura de la red neuronal.

Para el entrenamiento, se agregaron algunos callbacks los cuales sirven para personalizar y controlar el flujo de entrenamiento del modelo, para este caso se utilizaron:

- **Early Stopping:** parada temprana, el objetivo de este es evitar el sobreajuste (overfitting) y mejorar la eficiencia del proceso de entrenamiento. Su funcionamiento se basa en la monitorización de una métrica específica durante el entrenamiento y detiene el proceso

de entrenamiento cuando esta métrica deja de mejorar o empeora después de un cierto número de épocas consecutivas, para este caso la métrica a monitorear fue la perdida en validación.

- **Decay Schedule:** plan de decaimiento de la tasa de aprendizaje, es una estrategia que controla cómo disminuye la tasa de aprendizaje, a lo largo del tiempo durante el proceso de entrenamiento.
- **CSV Logger:** bitácora del entrenamiento, se utiliza para registrar automáticamente las métricas y el progreso del entrenamiento en un archivo CSV (valores separados por comas). Su función principal es ayudar en la monitorización y el análisis del rendimiento del modelo a lo largo del tiempo durante el entrenamiento, lo cual ayuda a no caer en mínimos locales en etapas tempranas del entrenamiento y a posibilitar la convergencia en las etapas finales del entrenamiento.
- **Model Checkpoint:** guardado de puntos de control, su función principal es guardar automáticamente los puntos de control del modelo durante el proceso de entrenamiento, esto permite evitar perdidas de progreso y reanudar el entrenamiento desde un punto determinado.

Ver código [A](#).

Layer (type)	Output Shape	Param #
<hr/>		
input (InputLayer)	[(None, 35)]	0
dense_125 (Dense)	(None, 1024)	36864
batch_normalization_80 (BatchNormalization)	(None, 1024)	4096
dense_126 (Dense)	(None, 512)	524800
batch_normalization_81 (BatchNormalization)	(None, 512)	2048
dense_127 (Dense)	(None, 128)	65664
batch_normalization_82 (BatchNormalization)	(None, 128)	512
dense_128 (Dense)	(None, 1)	129

**Figura 5.2: Arquitectura MLP.**

A continuación un ejemplo de un entrenamiento, utilizando la clase "PerceptronMulticapa", en donde primero se definen los hyper-parámetros necesarios, luego se construye la clase, se crea la arquitectura, se compila y finalmente se realiza el entrenamiento.

## CAPÍTULO 5. METODOLOGÍA

---

### Algoritmo 1: Llamado *entrenamiento red neurona MLP*

---

```
1 # Dimensiones de entrada
2 input_dim = X_train.shape[1]
3 # Capas y numero de neuronas
4 units = [1024,512,128,1]
5 # Funcion de activacion de cada una de las capas
6 activation = ['sigmoid','relu','relu','sigmoid']
7
8 # Tasa de aprendizaje
9 lr = 0.001
10 # Tamano del lote de aprendizaje
11 batch_size = 32
12 # Epochas de entrenamiento
13 epochs = 100
14 # Funcion de perdidas
15 loss='binary_crossentropy'
16
17 # Metricas
18 metrics=['AUC']
19
20 # Regularizadores
21 # Normalizacion de lotes
22 use_batch_norm=True
23 # Dropout
24 use_dropout=False
25
26
27 #llamado entrenamiento
28 if is_training:
29     # Construcion de la clase PerceptronMulticapa
30     my_MLP = PerceptronMulticapa(input_dim ,units ,activation)
31     # Construcion de la arquitectura
32     my_MLP.build(use_batch_norm=use_batch_norm ,use_dropout=use_dropout)
33     # Compilacion
34     my_MLP.compile(learning_rate=lr , loss=loss , metrics=metrics)
35     # Entrenamiento
36     H = my_MLP.train(X_train , y_train , epochs , batch_size , 0 , run_folders)
```

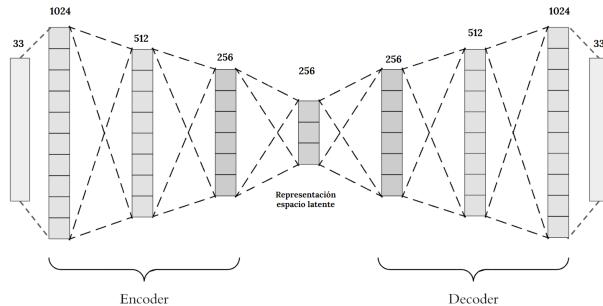
---

Para nuestro entrenamiento solo fueron necesarias 100 épocas ya que el entrenamiento convergía relativamente rápido, a través de los múltiples experimentos también se evidenció que el tamaño de lote de 32 y la tasa de aprendizaje de 0.001 fueran las que arrojaron mejores métricas de área bajo la curva (AUC).

### Autoencoders

En el segundo enfoque de las redes neuronales, implementamos una arquitectura de autoencoders, el cual es una técnica de aprendizaje no supervisado y su objetivo principal es aprender a representar datos de entrada de manera eficiente, extrayendo características significativas y reduciendo la dimensionalidad de los mismos. En este la red neuronal se entrena utilizando solo ejemplos de la clase negativa, es decir, datos que representan comportamientos de negocios legales o no fraudulentos. El proceso comienza con una fase de codificación (encoder), en la que los datos de un comercio se transforman en una representación compacta en un espacio conocido como el espacio latente. Esta representación captura las características más importantes de los datos originales.

Luego, esta representación codificada pasa por una fase de decodificación (decoder). El objetivo del decoder es reconstruir una versión lo más cercana posible a los datos originales a partir de la representación codificada. Durante el entrenamiento, la red se ajusta para minimizar la pérdida, que es la diferencia entre los atributos originales y la salida del decoder. La idea es que, al aprender a reconstruir los datos originales de manera precisa, el autoencoder aprende automáticamente a extraer y representar las características más relevantes de los datos de entrada.



**Figura 5.3: Arquitectura autoencoders comercios.**

Durante la fase de pruebas, alimentamos el modelo con muestras de ambas clases: negocios legales (clase negativa) y negocios fachada (clase positiva). El objetivo de esta fase es evaluar cómo se comporta el autoencoder en la detección de negocios fachada. Una característica clave es que, dado que el autoencoder se ha entrenado principalmente con ejemplos de la clase negativa, se espera que esta clase presente una menor pérdida durante la fase de prueba. Esto se debe a que el modelo ha aprendido a representar y reconstruir eficientemente los datos de negocios legales.

Por otro lado, cuando se presentan muestras de negocios fachada (clase positiva), se observa una mayor pérdida en comparación con las muestras de negocios legales. Esta diferencia en las pérdidas es crucial para la identificación de negocios fachada. El autoencoder ha aprendido a reconstruir de manera deficiente los datos de negocios fachada, ya que estos datos son atípicos en comparación con los ejemplos en los que se entrenó principalmente.

Para llevar a cabo el entrenamiento, utilizamos la clase "PerceptronMulticapa", la cual se encarga de generar y gestionar los experimentos. Su funcionamiento es análogo al explicado anteriormente en el contexto del perceptrón multicapa. La distinción principal radica en el método "build-A", el cual se encarga de construir la arquitectura del autoencoder y este recibe las dimensiones del decoder, el espacio latente y el decoder.

Model: "model_51"		
Layer (type)	Output Shape	Param #
encoder_input (InputLayer)	[None, 35]	0
encoder_0 (Dense)	(None, 1024)	36864
encoder_1 (Dense)	(None, 512)	524800
encoder_2 (Dense)	(None, 256)	131328
encoder_output_z (Dense)	(None, 256)	65792
decoder_t0 (Dense)	(None, 256)	65792
decoder_t1 (Dense)	(None, 512)	131584
decoder_t2 (Dense)	(None, 1024)	525312
decoder_t3 (Dense)	(None, 35)	35875
<hr/>		
Total params:	1517347 (5.79 MB)	
Trainable params:	1517347 (5.79 MB)	
Non-trainable params:	0 (0.00 Byte)	

**Figura 5.4: Arquitectura de nuestro autoencoder.**

A continuación un ejemplo de un entrenamiento, utilizando la clase "AutoEncoder", en donde primero se definen los hyper-parámetros necesarios, luego se construye la clase, se crea la arquitectura, se compila y finalmente se realiza el entrenamiento.

**Algoritmo 2:** Llamado *entrenamiento red neuronal AE*

```

1  # Dimensiones de entrada
2  input_dim = X_train.shape[1]
3  # Capas y neuronas del encoder
4  encoder_units = [1024,512,256]
5  # Capas y neuronas del decoder
6  decoder_units = [256,512,1024,input_dim]
7  # Dimensione del espacio latente
8  z_dim = 256
9
10 # Tasa de aprendizaje
11 lr = 0.001
12 # Tamano del lote de aprendizaje
13 batch_size = 32
14 # Epochas de entrenamiento
15 epochs = 100
16 # Funcion de perdidas
17 loss='mse'
18 # Regularizadores
19 use_batch_norm=False
20 use_dropout=False
21
22 is_training = True
23
24 #llamado entrenamiento
25 if is_training:
26     # Construccion de la clase Autoencoder
27     my_AE = Autoencoder(input_dim,encoder_units,decoder_units,z_dim)
28     # Construccion de la arquitectura
29     my_AE.build(use_batch_norm=use_batch_norm,use_dropout=use_dropout)
30     # Compilacion
31     my_AE.compile(learning_rate=lr, loss=loss)
32     # Entrenamiento
33     H = my_AE.train(X_train,X_val,epochs,batch_size,run_folders)
34

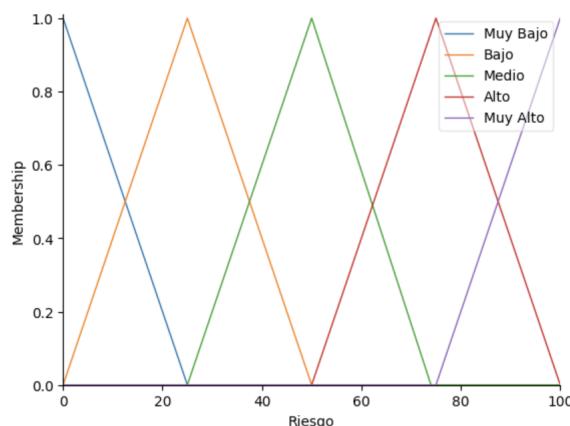
```

### 5.4.2. Modelo de Lógica Difusa

Nuestro modelo de lógica difusa, está provisto de un motor de inferencia, el cual solo recibe seis atributos del comercios, denominados V1,V2,V3,V4,V5 y V6 y como salida obtenemos un puntaje obtenido por un comercio entre 0 y 100 y también define un grado pertenencia a cada una de la etiquetas lingüísticas:

- Muy bajo: comercios que no representan ningún riesgo y en donde sus clientes tienen un comportamiento ejemplar.
- Bajo: comercios que no representan ningún riesgo y en donde sus clientes tienen un comportamiento casi ejemplar.
- Medio: comercios en los que algunos de sus comportamiento se pueden clasificar como no tan adecuados, los comercios pueden seguir realizando transacciones, pero se deben de prestar atención a algunas de sus propiedades.
- Alto: comercios que debido a la mayoría de sus comportamientos se clasifican como comercios con riesgo alto y el sistema puede suspender los desembolsos automáticamente, para luego ser evaluados y suspendido definitivamente.
- Muy Alto: comercios que debido a todos comportamientos se clasifican como comercios con riesgo muy alto y el sistema los puede prescribir automáticamente, para ser bloqueados y no permitir más transacciones de estos.

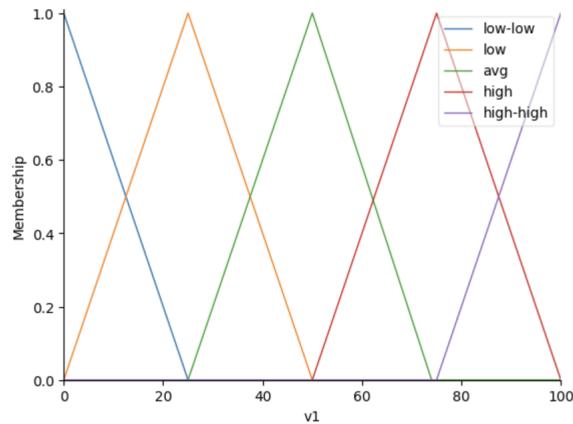
Para nuestro modelo una calificación alta en la calidad de un comercio, significa un mayor grado de pertenencia a las clases de riesgo. Es decir que una calificación alta, es un aspecto negativo del comercio, ya que se acerca más a ser un comercio fachada. Esto con el fin de tener un comportamiento similar los modelos de aprendizaje supervisado y que este resultado actué, como la clase positiva en dichos modelos y así poder realizar comparaciones y agregaciones entre ambos modelos.



**Figura 5.5: Conjunto borroso comercios riesgo.**

Los atributos de entrada corresponden a tasas medidas entre dos variables, ejemplo:

$$X' = X1/Xtotal$$



**Figura 5.6: Función de membresía atributo V1.**

A continuación veremos como se definen las reglas de nuestro modelo: [5.3](#)

<b>Regla</b>	<b>Atributo</b>	<b>Valor</b>	<b>Calificación Riesgo Comercio</b>
R 1	V1	muy-alto	muy-alto
R 2	V2	muy-alto	muy-alto
R 3	V3	muy-alto	muy-alto
R 4	V4	muy-alto	muy-alto
R 5	V5	muy-bajo	muy-alto
R 6	V6	muy-bajo	muy-alto
R 7	V1	alto	muy-alto
R 8	V2	alto	muy-alto
R 9	V3	alto	muy-alto
R 10	V4	alto	alto
R 11	V5	bajo	alto
R 12	V6	bajo	alto
R 13	V1	avg	alto
R 14	V2	avg	alto
R 15	V3	avg	alto
R 16	V4	avg	medio
R 17	V5	avg	medio
R 18	V6	avg	medio

Tabla 5.3: *Reglas motor de inferencia primera parte.*

<b>Regla</b>	<b>Atributo</b>	<b>Valor</b>	<b>Calificación Riesgo Comercio</b>
R 19	V1	bajo	medio
R 20	V2	bajo	medio
R 21	V3	bajo	medio
R 22	V4	bajo	bajo
R 23	V5	alto	bajo
R 24	V6	alto	bajo
R 25	V1	muy-bajo	muy-bajo
R 26	V2	muy-bajo	muy-bajo
R 27	V3	muy-bajo	muy-bajo
R 28	V4	muy-bajo	muy-bajo
R 29	V5	muy-alto	muy-bajo
R 30	V6	muy-alto	muy-bajo

**Tabla 5.4:** Reglas motor de inferencia segunda parte.

### Construcción en Python

Para la construcción del motor de inferencia se utilizó la librería de Python ScipyFuzzy. Para esto se realizaron cuatro procedimientos, el primero es el import de las librerías necesarias [7](#), segundo la definición de los conjuntos difusos [8](#), tercero la definición de las reglas y creación del motor de inferencia [9](#) y por último el llamado a la simulación [10](#).

**Antecedentes:** El modelo tiene como antecedentes de seis *termsets* denominados así:

$$V1 = V1'/V_{total}$$

$$V2 = V2'/V_{total}$$

$$V3 = V3'/V_{total}$$

$$V4 = V4'/V_{total}$$

$$V5 = V5'/V_{total}$$

$$V6 = V6'/V_{total}$$

Como es de esperarse las tasas tienen un dominio entre 0 y 1 y luego son escalados a los valores esperados por las funciones de pertenencia de los conjuntos borrosos, por lo cual se multiplican por 100. Toda las funciones de membresía de los los antecedentes tienen la misma forma, esto se debe a que son tasas.

Los atributos V1,V2,V3,V4 tienen un comportamiento positivo en la calificación, es decir si la medición aumenta, aumenta el grado de pertenencia a alto riesgo. Por el contrario las variables V5 y V6 tienen una incidencia negativa en la clasificación, es decir que cuando aumenta el valor, los comercios presentan menos riesgos.

**Consecuente:** como consecuente de nuestro modelo tenemos la calificación del comercio descrita anteriormente.

**Clase FuzzyFacade:** en el marco del arquetipo se definió la clase ‘FuzzyFacadeRank’, la cual contiene toda la lógica de construcción del modelo difuso: definición de conjuntos borroso, definición de reglas, creación del motor de inferencia y finalmente la simulación del motor.

- **init :** el método init es el constructor de la clase, y en este se define todos los conjuntos borrosos de antecedentes y precedentes.
- **getCombineSim:** este método define todas las reglas del sistema y finalmente construye el motor de inferencia de este, basado en los conjuntos borrosos y reglas definidas anteriormente. El motor de inferencia se almacena en la variable: self.combineSimulation.
- **compCombineSimulation:** este método se encarga de realizar una simulación, para esto puede recibir tres parámetros: **row**, el cual trae un registro desde un dataframe, para poder construir el diccionario y simular el sistema, este se utiliza por lo general para procesos masivos. **dicComb**, es el diccionario ya construido, por lo cual no es necesario construirlo, por lo general se utiliza para consultas únicas. Adicionalmente se tiene el parámetro **view**, el cual indica si se requiere visualizar gráficamente el resultado.

Ver código fuente de construcción del motor de inferencia [B](#).

Un ejemplo de llamado al motor de inferencia sería:

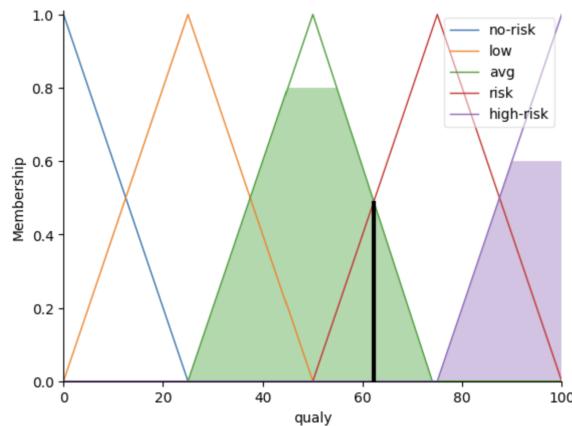
```
# Creación motor de inferencia
fmodel.get_combine_sim()

input_dict = {'v1':     87,
              'v2' :    90,
              'v3' :    45,
              'v4' :    55,
              'v5' :    10,
              'v6' :   38.0}
y_agg = fmodel.comp_combine_simulation(dic_comb=input_dict, view=True)

print(f'Calificación comercio:{y_agg}' )
```

**Figura 5.7: Llamado a motor de inferencia.**

Calificación comercio:62.32800313541049  
Etiqueta: risk



**Figura 5.8: Resultado llamado a motor de inferencia.**

## 5.5. Validación de Modelos

Para la validación de los modelos de redes neuronales, se utilizaron diversas métricas de aprendizaje supervisado, que incluyen la matriz de confusión, la curva ROC, la curva lift y la curva de calibración. En el caso de los ejercicios relacionados con lógica difusa, se llevó a cabo una validación mediante lotes de resultados en colaboración con expertos. Durante este proceso, se analizaron las etiquetas lingüísticas utilizando evaluaciones manuales realizadas por analistas de seguridad. Además, como punto de referencia, se examinaron la matriz de confusión y la curva ROC en los ejercicios relacionados con lógica difusa.

**Curva ROC** (Receiver Operating Characteristic), es una representación gráfica utilizada especialmente en problemas de clasificación binaria. Esta curva muestra la relación entre la tasa de verdaderos positivos (Sensibilidad o Recall) en el eje Y y la tasa de falsos positivos (1-Especificidad) en el eje X, en diferentes umbrales de decisión para un modelo de clasificación.

**Matriz de confusión**, muestra la cantidad de predicciones correctas e incorrectas hechas por el modelo en relación con las clases reales de un conjunto de datos. Está organizada en filas y columnas, donde cada fila representa la clase real y cada columna representa la clase predicha.

**Curva lift**, muestra cómo mejora la capacidad predictiva de un modelo en comparación con un enfoque aleatorio o sin modelo. Representa la relación entre la tasa de verdaderos positivos acumulados y la tasa de falsos positivos acumulados a medida que cambiamos el umbral de decisión del modelo, está también sirvió para definir el umbral de decisión óptimo del modelo.

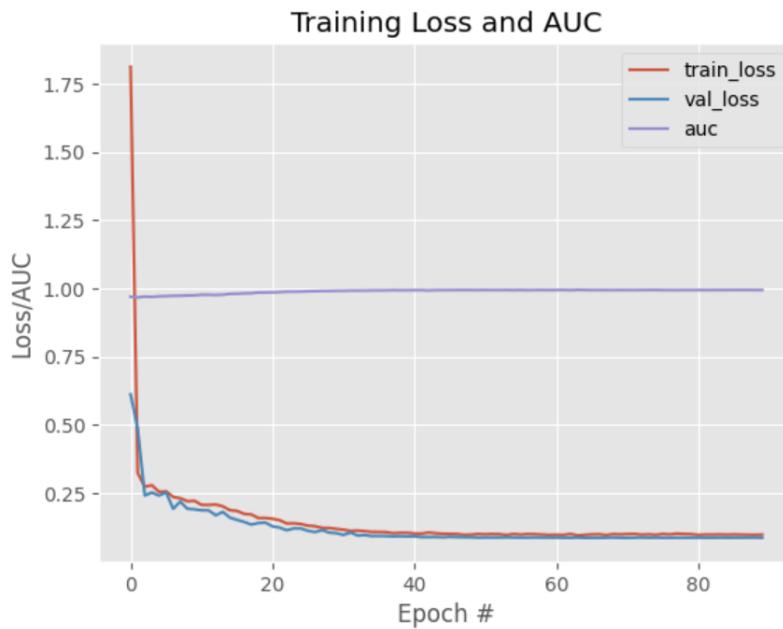
**Curva de calibración**, esta métrica se utiliza para medir la precisión de las probabilidades pronosticadas por un modelo en relación con las frecuencias observadas de eventos positivos, esta es útil para evaluar si un modelo está generando probabilidades bien calibradas, lo que significa que las probabilidades reflejan con precisión la probabilidad real de que ocurra un evento positivo. Una calibración perfecta se vería como una línea diagonal de 45 grados en la gráfica, donde las probabilidades pronosticadas son iguales a las tasas observadas de eventos positivos.

### 5.5.1. Validación del Modelo de Redes Neuronales

Se llevaron a cabo un total de 10 experimentos, documentando modelos de redes neuronales de perceptrón multicapa, así como 9 experimentos con Autoencoders. Destacando el experimento 010 de perceptrón multicapa como el que ofreció los mejores resultados, por lo tanto este fue seleccionado como nuestro modelo definitivo en el enfoque de redes neuronales. Además, nos propusimos incluir un ejemplo de Autoencoders debido a su enfoque de entrenamiento no supervisado; sin embargo, lamentablemente, no obtuvo buenos resultados. Los valores de f1 estuvieron por debajo del 70 %, y su rendimiento fue significativamente inferior al modelo de perceptrón multicapa.

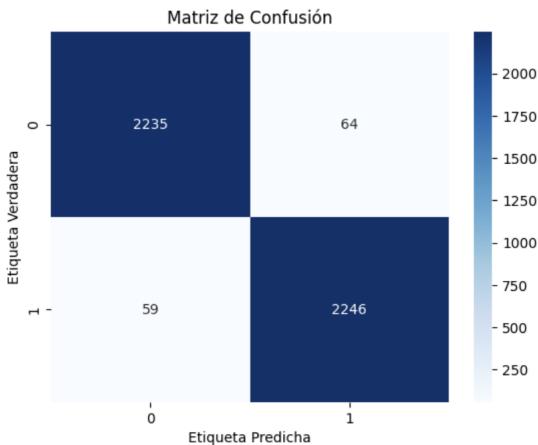
Una característica del entrenamiento del MLP fue su rápida convergencia, alcanzando estabilidad en aproximadamente 100 épocas, en las épocas finales la perdida en validación y el

valor del área bajo la curva (AUC), no presentaron mejoras significativas, por eso se determinó que fue un buen punto para detener el entrenamiento.



**Figura 5.9: Historial de entrenamiento experimento 010.**

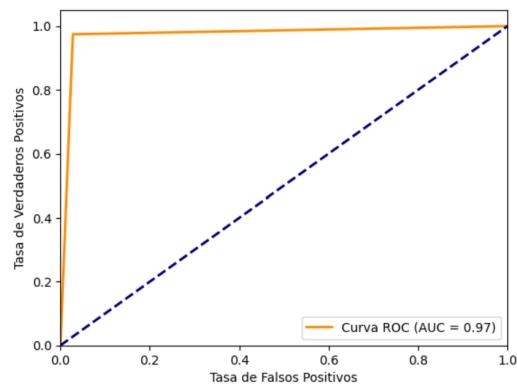
La matriz de confusión del experimento nos arrojó muy buenos resultados, a continuación podemos ver la matriz de confusión generada en pruebas del modelo.



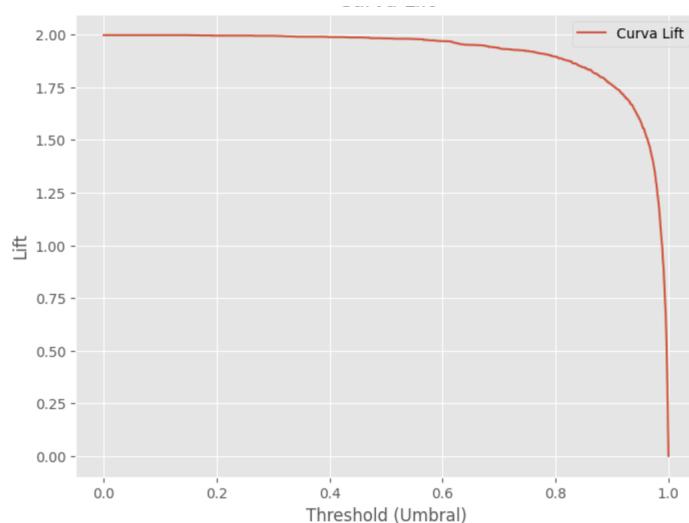
**Figura 5.10: Matriz de confusión experimento 010.**

Con un valor de 0.97 de área bajo la curva, la curva ROC arrojó un resultado, sobresaliente del experimento 010, lo que hizo que este fuera nuestro modelo escogido de redes neuronales.

La curva lift nos hablan de la buena capacidad predictiva del modelo , como el modelo tiene un Lift mayor que 1 nos indica que es mejor que una elección aleatoria.

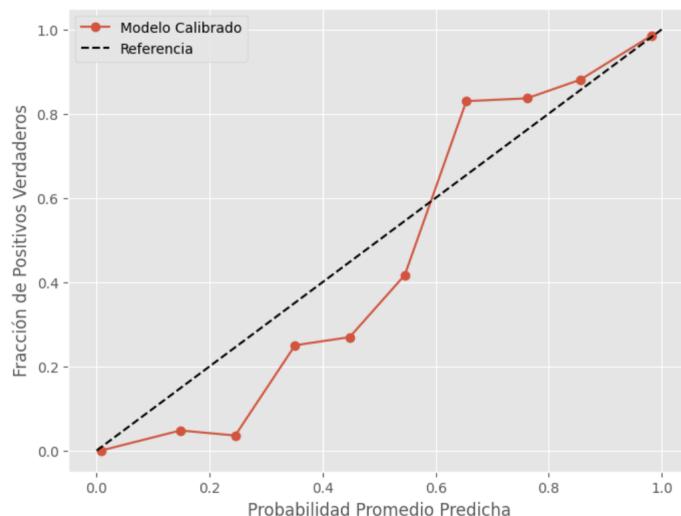


**Figura 5.11: Curva ROC experimento 010.**



**Figura 5.12: Curva lift experimento 010.**

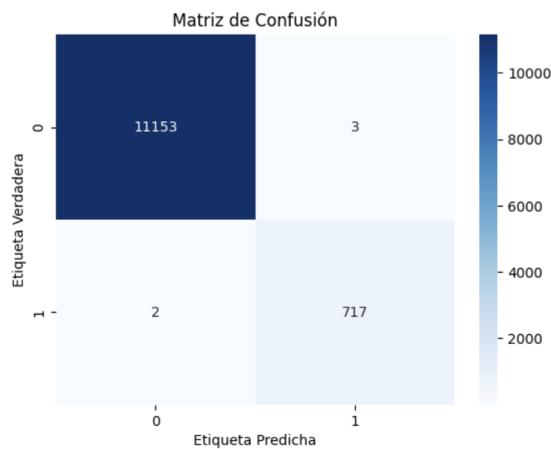
La gráfica de calibración nos muestra un comportamiento cercano a la diagonal, lo cual nos habla de la precisión de modelo.



**Figura 5.13: Curva de calibración experimento 010.**

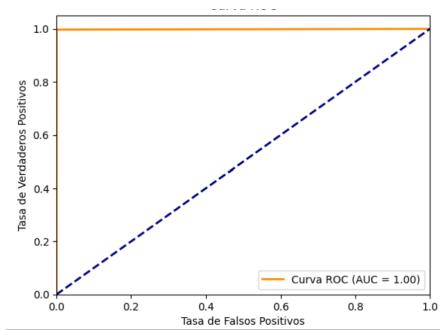
### 5.5.2. Validación del Modelo de Lógica Difusa

El modelo de lógica difusa fue sometido a múltiples validaciones por parte de expertos, y los resultados obtenidos fueron altamente positivos. Durante las pruebas, se evaluaron varios escenarios, incluyendo el análisis de nuevos comercios, y se alcanzó una tasa de éxito del 100 %. Solo se detectó una única incidencia que no correspondía a un comercio fachada, pero se determinó que se trataba de un comercio en proceso de prueba, lo que explicaba su comportamiento atípico. Además de las validaciones expertas, también se realizó una validación utilizando la matriz de confusión, que arrojó un alto índice de éxito.



**Figura 5.14: Matriz de confusión experimento modelo lógica difusa.**

En la curva ROC de nuestro modelo de lógica difusa, observamos un valor de área bajo la curva (AUC) igual a 1. Este valor representa un rendimiento perfecto del modelo, lo cual, aunque inusual, puede tener varias interpretaciones. Puede sugerir una calidad óptima del modelo, pero también podría levantar sospechas de sobreajuste. Abordaremos estas consideraciones en nuestro enfoque final, que combinará aspectos de modelos neuro-difusos y destacará las posibles mejoras que podríamos obtener al hacerlo.



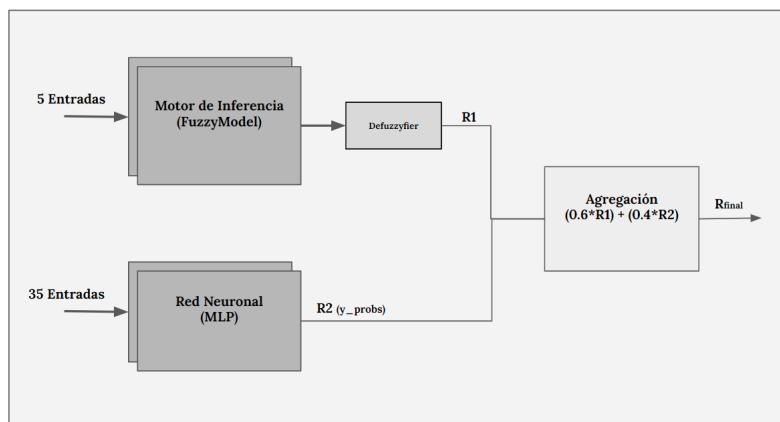
**Figura 5.15: Curva ROC modelo de lógica difusa.**

## 5.6. Sistema Neuro-Difuso

Nuestro sistema neuro-difuso fusiona los modelos de lógica difusa (fuzzy) con nuestro modelo de redes neuronales (MLP). Su funcionamiento consiste en evaluar un comercio utilizando los 5 atributos que componen los antecedentes del modelo difuso y los 35 atributos de entrada del modelo MLP. Luego, los resultados de ambos modelos se combinan mediante una función de agregación, donde el 60 % del peso se asigna al modelo fuzzy y el 40 % al modelo MLP, utilizando un promedio ponderado.

$$R_{final} = (R_{fuzzy} * 0,6) + (R_{MLP} * 0,4)$$

La elección de la función de agregación promedio ponderado, se hizo basado en diversas pruebas de combinación de ambos modelos en los que se exploraron las funciones min, max y diversas OWA's.



**Figura 5.16: Modelo Neuro-difuso.**

El interés en combinar ambos enfoques radica en aprovechar lo mejor de cada uno de ellos, permitiendo que nuestro modelo de lógica difusa capture y modele el conocimiento y la experiencia de expertos en el campo, inclusive se tiene la capacidad de modelar políticas empresariales mediante reglas, lo que a su vez nos brinda la ventaja de anticiparnos y ser capaces de inferir ante situaciones que aún no se han presentado, pero que mediante las reglas se hayan podido anticipar.

Por otro lado, el modelo de redes neuronales juega un papel crucial al aprender de los datos y descubrir nuevos patrones de comportamiento en los comercios fraudulentos. Esto es esencial porque las organizaciones fraudulentas a menudo cambian y prueban nuevos patrones de comportamiento en un intento de eludir los sistemas de detección. La capacidad de adaptación y aprendizaje de la red neuronal nos permite mantenernos actualizados y efectivos en la detección de estas tácticas en constante evolución.

## 5.7. Herramientas Utilizadas

Las herramientas utilizadas dentro de este trabajo son:

- FuzzyClips V6.10d
- Python V3.11
- Visual Studio Code V1.82.2
- DBeaver V23.2



# Resultados y Discusión

# 6

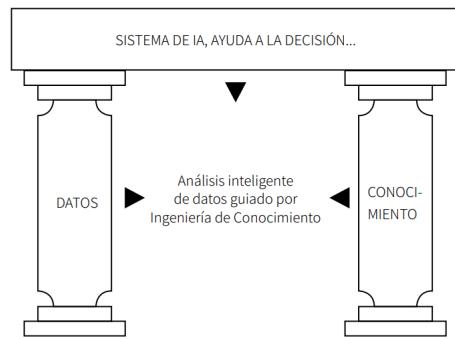
En el desarrollo de nuestro sistema neuro-boroso se construyeron dos modelos que a primera vista pueden competir, pero que alineados con nuestro propósito, pueden ser complementarios. Como métrica comparativa de ambos modelos utilizaremos el AUC, el cual obtuvo un valor e 0.97 para modelo MLP y 1 para nuestro modelo difuso. A primera vista, con puntuaciones tan altas, podría parecer razonable optar por uno de los modelos y avanzar. Sin embargo, hay razones fundamentales que respaldan nuestra decisión de combinar ambos modelos. Nuestro propósito radica en aprovechar lo mejor de ambos mundos: la riqueza de los datos y la potencia de la ingeniería del conocimiento.

Nuestro modelo de redes neuronales, nos brinda un aprovechamiento de los datos, los cuales representan el comportamiento de los comercios y a partir de los cuales se intentan aprender patrones. En el ámbito de la ciberseguridad esto es de suma importancia ya que nos da la opción de detectar nuevos patrones que representen las adaptaciones que hacen los ciberdelincuentes a las acciones que buscan impedir el fraude.

Como segunda medida, nuestro modelo basado en lógica difusa se aprovecha de la ingeniería del conocimiento a través del uso de reglas. Esto se torna crucial al anticipar patrones que, aunque aún no se han manifestado, la experiencia nos sugiere que podrían materializarse. De esta manera, enriquecemos nuestro modelo con la sabiduría de expertos en detección de fraude y aplicamos fácilmente políticas de seguridad sugeridas y concretadas a través de reglas. La configuración de estas reglas es simple, pero su poder radica en la capacidad de controlar todas las posibles combinaciones, gracias al uso de lógica difusa. Esta aproximación permite manejar atributos continuos sin necesidad de generar árboles de decisión complejos que cubran todo el espectro de combinaciones de atributos.

Una representación de lo anterior es la siguiente figura que muestra en el sentido de que ambas columnas pueden interactuar para generar y/o comprobar/refutar hipótesis, proponer modelos, patrones, etc. en ambos sentidos [Olivas et al. \(2023\)](#).

Uno de los modelos de redes neuronales que se exploraron en este trabajo fue el de auto-encoders, sin embargo, lamentablemente, no se obtuvieron resultados satisfactorios. Una de las posibles razones detrás de este resultado fue la relativa poca cantidad de datos disponibles para el entrenamiento. Los autoencoders son una de las técnicas más vanguardistas en la actualidad para la detección de fraudes. Estos modelos se centran en generar una reconstrucción



**Figura 6.1: Sistemas guiados por ingeniería del conocimiento.**

de los datos originales a partir de su representación en el espacio latente, pero, desafortunadamente, en este experimento no se logró esta reconstrucción con éxito. Esto resultaba especialmente interesante, dado que los autoencoders se basan en un enfoque de aprendizaje no supervisado, pero en cual es posible realizar validación extrínseca y de esta forma explorar un aspecto diferente del aprendizaje automático aplicado a un mismo problema.

# Conclusiones

7

1. El uso intensivo de datos y la aplicación de conocimiento, dos vertientes dentro del campo de la inteligencia artificial, son dos enfoques que, en lugar de competir, demuestran ser altamente complementarios. Los datos brindan la materia prima para que los algoritmos aprendan y descubran patrones, mientras que el conocimiento experto proporciona las directrices y las reglas para la toma de decisiones. Esta sinergia entre el aprendizaje automático basado en datos y la lógica difusa, que aprovecha la experiencia humana, nos ha permitido abordar el desafío de la detección de fraudes de manera más eficiente y efectiva.
2. En un mundo donde los patrones de fraude evolucionan constantemente, contar con la capacidad de aprender de los datos en tiempo real y, al mismo tiempo, tomar decisiones basadas en el conocimiento experto, se vuelve esencial. La combinación de estos enfoques nos permite no solo identificar los patrones de fraude conocidos, sino también anticiparnos a las nuevas estrategias de los cibercriminales.
3. La lógica difusa, aunque conocida principalmente por su aplicación en sistemas de control, ha demostrado ser una herramienta versátil y eficaz para la toma de decisiones inteligentes en sistemas de información.
4. Este trabajo de investigación resalta la importancia de no limitarse a una sola técnica, sino de adoptar un enfoque integral al aprovechar el poder de los datos junto con la experiencia humana. Los modelos neurodifusos de inteligencia artificial se han mostrado como una herramienta valiosa en la detección de fraudes, y su potencial se extiende a otros campos donde la toma de decisiones preventiva, precisa y adaptativa es fundamental.



# Limitaciones y Perspectivas de Futuro

8

El trabajo que presentas sobre el sistema neuro-boroso para la detección de comercios fachadas es sin duda un paso importante en la aplicación de la inteligencia artificial a la seguridad y la detección de fraudes. Sin embargo, como en cualquier investigación, existen limitaciones y perspectivas de futuro que vale la pena considerar.

## Limitaciones

- **Tamaño del conjunto de datos:** una de las limitaciones evidentes en tu trabajo es el tamaño del conjunto de datos. Los resultados de nuestro modelo de autoencoders no fueron satisfactorios posiblemente debido a la cantidad limitada de datos, este es un desafío común en la mayoría de los proyectos de inteligencia artificial. En el futuro, la obtención de datos más extensos y representativos podría mejorar los resultados y adaptarse a las nuevas condiciones.
- **Adaptabilidad:** Un desafío crítico en los sistemas de detección de fraudes radica en su capacidad para mantenerse al día con las tácticas en constante evolución empleadas por los ciberdelincuentes. Las reglas difusas que se han establecido con el propósito de inferir patrones a partir del estado actual deben mantenerse en constante adaptación para seguir siendo efectivas. Este proceso de adaptación se apoya en gran medida en nuestro modelo de redes neuronales, que puede detectar y aprender nuevos patrones de comportamiento a medida que surgen. La continuidad de esta adaptabilidad es esencial para mantener la eficacia de los sistemas de detección de fraudes en un entorno en constante cambio.

## Perspectivas de futuro

- **Aumento de datos:** una de las perspectivas más evidentes es el aumento de datos. Obtener y etiquetar un conjunto de datos más grande y diverso permitiría una capacitación más efectiva de los modelos y posiblemente mejorar los resultados.
- **Función de agregación:** Es importante evaluar y analizar nuevas medidas de agregación de ambos modelos, hay nuevas métricas que son importantes considerar como es la distancia en la medición de ambos modelos para un mismo comercio y entender el significado de estas discrepancias. Además, se plantea la necesidad de explorar la posibilidad de ajustar la función de agregación utilizada para tener en cuenta esta distancia entre

---

ambas mediciones. Esta evaluación y análisis enriquecerán significativamente nuestro entendimiento de cómo los modelos de lógica difusa y redes neuronales trabajan en conjunto y cómo pueden mejorarse aún más

- **Investigación continua:** la inteligencia artificial y la detección de fraudes son campos en constante evolución. Continuar investigando y manteniéndote al tanto de las últimas tendencias y avances es esencial para seguir siendo relevante y efectivo en la detección de fraudes.
- **Automatización total:** una perspectiva a mediano plazo podría ser la automatización completa del proceso de detección y prevención de fraudes, donde el sistema no solo detecte fraudes sino que también tome medidas para mitigarlos de manera autónoma. Si el sistema está o no preparado para hacer esto, nos lo dirá la calidad de su funcionamiento en un ambiente productivo.

# **Lista de Acrónimos**

**AE** *Autoencoders.*

**AI** Inteligencia Artificial o *Artificial Intelligence.*

**AUC** Área Bajo la Curva o *Area Under Curve.*

**ANN** Red Neuronal Artificial o *Artificial Neural Network.*

**CNN** Red Neuronal Convolucional o *Convolutional Neural Network.*

**GAN** Red Generativa Antagónica o *Generative Adversarial Network.*

**LSTM** Red Neuronal de Memoria Corto y Largo Plazo o *Long and Short Long Term Memory.*

**ML** Aprendizaje Automático o *Machine Learning.*

**MLP** Perceptrón Multicapa o *Multilayer Perceptron .*

**RNN** Red Neuronal Recurrente o *Recurrent Neural Network .*

# Código Redes Neuronales

A

---

### Algoritmo 3: Algoritmo *Imputación valores áтипicos*

---

```
1 impute_cols = ['V_1', 'V_2', 'V_3']
2 umbral = 2
3 for column in impute_cols:
4     #define z_scores
5     z_scores = stats.zscore(facade[column])
6     #define max and min threshold
7     column_max_ = facade[z_scores < umbral][column].max()
8     column_min_ = facade[z_scores > -umbral][column].min()
9
10    #inpute outlier to max/min acepeted value
11    facade[column] = facade[column].apply(lambda x: min(x, column_max_))
12    facade[column] = facade[column].apply(lambda x: max(x, column_min_))
```

---

### Algoritmo 4: Algoritmo *Construcción de la arquitectura MLP*

---

```
1 def build(self, use_batch_norm=False, use_dropout=False):
2     # Definición 'n capa de entrada
3     input_layer = Input(shape=self.input_dim, name = 'input')
4     x = input_layer
5     for i in range(len(self.units)-1):
6         layer = Dense(units = self.units[i]
7                         ,activation=self.activation[i]
8                         ,kernel_regularizer=regularizers.l2(0.01))
9         x = layer(x)
10        # Agregar Batch Normalization
11        if use_batch_norm:
12            x = BatchNormalization()(x)
13        # Agregar Dropout
14        if use_dropout:
15            x = Dropout(0.25)(x)
16
17        #Última Capa
18        layer = Dense(units = self.units[-1],activation=self.activation[-1])
19        x = layer(x)
20        self.model = Model(input_layer,x)
```

### Algoritmo 5: Algoritmo *Entrenamiento modelo MLP*

---

```
1  def train(self, X_train, y_train, epochs, batch_size, validation_split, run_folders, weight_class_1=1):
2
3      # Definición 'n' Callbacks
4      csv_logger = CSVLogger(run_folders["log_filename"])
5      checkpoint = ModelCheckpoint(os.path.join(run_folders["model_path"])
6                                    , run_folders["exp_name"]+ '/weights/MLP_weights.h5')
7                                    , save_weights_only=True
8                                    , verbose=1)
9
10     lr_sched = self.step_decay_schedule(initial_lr=self.learning_rate
11                                         , decay_factor=0.75
12                                         , step_size=2)
13     early_stop = EarlyStopping(monitor='val_loss', patience=25)
14     callbacks_list = [csv_logger, checkpoint, lr_sched, early_stop]
15
16     # Creación 'n' particiones internas
17     X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.2, random_state=42)
18
19     # Entrenamiento
20     print("[INFO]: Training")
21     history = self.model.fit(X_train, y_train,
22                               epochs=epochs,
23                               batch_size=batch_size,
24                               verbose=1,
25                               callbacks=callbacks_list,
26                               validation_data=(X_val, y_val),
27                               class_weight={0: 1.0, 1: weight_class_1})
28
29     try:
30         self.save_model(run_folders, history)
31     except:
32         print(':::ALERT::: Error when save the model.')
33
34     return history
```

---

### Algoritmo 6: Algoritmo Construcción de la arquitectura AE

---

```
1  def build(self, use_batch_norm=False, use_dropout=False, VAE=False):
2
3      # Definición 'n del encoder
4      encoder_input = Input(shape=self.input_dim, name = 'encoder_input')
5      x = encoder_input
6      for i in range(len(self.encoder_units)):
7          layer = Dense(units = self.encoder_units[i]
8                          ,activation='tanh'
9                          ,name='encoder_'+str(i))
10         x = layer(x)
11         if use_batch_norm:
12             x = BatchNormalization()(x)
13
14         if use_dropout:
15             x = Dropout(0.25)(x)
16
17     # Creación 'n del espacio latente
18     x = Dense(self.z_dim, name='encoder_output_z')(x)
19
20     # Condición 'n para autoencoder variacional
21     if VAE:
22         self.mu = Dense(self.z_dim, name='mu')(x)
23         self.log_var = Dense(self.z_dim, name='log_var')(x)
24         def sampling(args):
25             mu, log_var = args
26             epsilon = K.random_normal(shape=K.shape(mu), mean=0., stddev=1.)
27             return mu + K.exp(log_var/2)*epsilon
28         encoder_output = Lambda(sampling, name='encoder_output')([self.mu, self.log_var])
29
30     # Definición 'n del decoder
31
32     for i in range(len(self.decoder_units)):
33         if i<len(self.decoder_units)-1:
34             activation='tanh'
35         else:
36             activation='tanh' # en caso de requerir otra activación para la salida
37             t_layer=Dense(units = self.decoder_units[i]
38                           ,activation=activation
39                           ,name='decoder_t'+str(i))
40
41         x = t_layer(x)
42
43     #Modelo definitivo
44     autoencoder = Model(encoder_input, x)
45     self.model = autoencoder
```

# Código Lógica Difusa

B

---

## Algoritmo 7: Algoritmo Importar librerías SKfuzzy

---

```
1 import skfuzzy as fuzz
2 from skfuzzy import control as ctrl
```

---

## Algoritmo 8: Algoritmo Definición conjuntos difusos

---

```
1
2 def __init__(self):
3
4     # Antecedentes
5
6     # Conjunto difuso V1
7     v1 = ctrl.Antecedent(np.arange(0,101,1), 'v1')
8     v1['low-low'] = fuzz.trimf(self.v1.universe, [0,0,25])
9     v1['low'] = fuzz.trimf(self.v1.universe, [0,25,50])
10    v1['avg'] = fuzz.trimf(self.v1.universe, [25,50,74])
11    v1['high'] = fuzz.trimf(self.v1.universe, [50,75,100])
12    v1['high-high'] = fuzz.trimf(self.v1.universe, [75,100,100])
13
14    # Los atributos V2,V3,V4,V5,V6 se definen exactamente igual
15
16    # Consecuente
17    qualy = ctrl.Consequent(np.arange(0,101,1), 'qualy')
18    qualy['no-risk'] = fuzz.trimf(self.qualy.universe, [0,0,25])
19    qualy['low'] = fuzz.trimf(self.qualy.universe, [0,25,50])
20    qualy['avg'] = fuzz.trimf(self.qualy.universe, [25,50,74])
21    qualy['risk'] = fuzz.trimf(self.qualy.universe, [50,75,100])
22    qualy['high-risk'] = fuzz.trimf(self.qualy.universe, [75,100,100])
```

---

**Algoritmo 9:** Algoritmo *Reglas del motor de inferencia*

---

```
1 def get_combine_sim(self, flush_after_run=10):
2
3     #Define Rules
4     rules = []
5
6     # HighRisk rules
7     rules.append(crtl.Rule(self.v1['high-high'],self.qualy['high-risk']))
8     rules.append(crtl.Rule(self.v2['high-high'],self.qualy['high-risk']))
9     rules.append(crtl.Rule(self.v3['high-high'],self.qualy['high-risk']))
10    rules.append(crtl.Rule(self.v4['high-high'],self.qualy['high-risk']))
11    rules.append(crtl.Rule(self.v5['low-low'],self.qualy['high-risk']))
12    rules.append(crtl.Rule(self.v6['low-low'],self.qualy['high-risk']))
13
14    # risk
15    rules.append(crtl.Rule(self.v1['high'],self.qualy['high-risk']))
16    rules.append(crtl.Rule(self.v2['high'],self.qualy['high-risk']))
17    rules.append(crtl.Rule(self.v3['high'],self.qualy['high-risk']))
18    rules.append(crtl.Rule(self.v4['high'],self.qualy['risk']))
19    rules.append(crtl.Rule(self.v5['low'],self.qualy['risk']))
20    rules.append(crtl.Rule(self.v6['low'],self.qualy['risk']))
21
22    # Medium
23    rules.append(crtl.Rule(self.v1['avg'],self.qualy['risk']))
24    rules.append(crtl.Rule(self.v2['avg'],self.qualy['risk']))
25    rules.append(crtl.Rule(self.v3['avg'],self.qualy['risk']))
26    rules.append(crtl.Rule(self.v4['avg'],self.qualy['avg']))
27    rules.append(crtl.Rule(self.v5['avg'],self.qualy['avg']))
28    rules.append(crtl.Rule(self.v6['avg'],self.qualy['avg']))
29
30    # Low-risk
31    rules.append(crtl.Rule(self.v1['low'],self.qualy['avg']))
32    rules.append(crtl.Rule(self.v2['low'],self.qualy['avg']))
33    rules.append(crtl.Rule(self.v3['low'],self.qualy['avg']))
34    rules.append(crtl.Rule(self.v4['low'],self.qualy['low']))
35    rules.append(crtl.Rule(self.v5['high'],self.qualy['low']))
36    rules.append(crtl.Rule(self.v6['high'],self.qualy['low']))
37
38    # NoRisk rules
39    rules.append(crtl.Rule(self.v1['low-low'],self.qualy['no-risk']))
40    rules.append(crtl.Rule(self.v2['low-low'],self.qualy['no-risk']))
41    rules.append(crtl.Rule(self.v3['low-low'],self.qualy['no-risk']))
42    rules.append(crtl.Rule(self.v4['low-low'],self.qualy['no-risk']))
43    rules.append(crtl.Rule(self.v5['high-high'],self.qualy['low']))
44    rules.append(crtl.Rule(self.v6['high-high'],self.qualy['low']))
45
46    # Create the control system based on rules
47    combine_ctrl = ctrl.ControlSystem(rules)
48    # Create simulation
49    combine_simulation = ctrl.ControlSystemSimulation(combine_ctrl, flush_after_run=flush_after_run)
50    # Set Rules
51    self.comb_rules = rules
52    # Set simylation
53    self.combine_simulation = combine_simulation
```

**Algoritmo 10:** Algoritmo *Simulación del motor de inferencia*

---

```
1  def comp_combine_simulation(self, row=None, dic_comb=None, view=False):
2      # Validate combine model
3      if dic_comb is None:
4          dic_comb = {'facade_rate':int(round(row['v1'])),
5                      'chargebacks_rate':int(round(row['v2'])),
6                      'foreign_rate':int(round(row['v3'])),
7                      'tc_rate':int(round(row['v4'])),
8                      'unique_rate':int(round(row['v5'])),
9                      'approved_rate':int(round(row['v5']))}
10         self.combine_simulation.inputs(dic_comb)
11     # Compute model
12     self.combine_simulation.compute()
13     #view
14     if view: self.qualy.view(sim=self.combine_simulation)
15     # Save result
16     return(self.combine_simulation.output['qualy'])
```

---

# Bibliografía

- Acevedo, E., Serna, A., y Serna, E. (2017). Principios y características de las redes neuronales artificiales. *Desarrollo e innovación en ingeniería*, 173.
- Astocondor Villar, J. (2011). Diseño del sistema de control por lógica difusa aplicado al brazo robótico translacional de 2dof.
- Boureau, Y.-L., Ponce, J., y LeCun, Y. (2010). A theoretical analysis of feature pooling in visual recognition. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 111–118.
- Cetinic, E., Lipic, T., y Grgic, S. (2020). Learning the principles of art history with convolutional neural networks. *Pattern Recognition Letters*, 129:56–62.
- Goguen, J. A. (1973). La zadeh. fuzzy sets. information and control, vol. 8 (1965), pp. 338–353.-la zadeh. similarity relations and fuzzy orderings. information sciences, vol. 3 (1971), pp. 177–200. *The Journal of Symbolic Logic*, 38(4):656–657.
- Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., y Bengio, Y. (2014). Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680.
- Hinton, G. E. y Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507.
- Hochreiter, S. y Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.
- Krizhevsky, A., Sutskever, I., y Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1098.
- LeCun, Y., Bengio, Y., y Hinton, G. (2015). Deep learning. *nature*, 521(7553):436–444.
- LeCun, Y., Bottou, L., Bengio, Y., y Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.
- Matich, D. J. (2001). Redes neuronales: Conceptos básicos y aplicaciones. *Universidad Tecnológica Nacional, México*, 41:12–16.
- Newell, A. (1969). A step toward the understanding of information processes: Perceptrons. an introduction to computational geometry. marvin minsky and seymour papert. mit press, cambridge, mass., 1969. vi+ 258 pp., illus. cloth, 12;paper, 4.95. *Science*, 165(3895):780–782.
- Olivas, J. A. (2001). La lógica borrosa y sus aplicaciones. *Universidad de Castilla. La Mancha. Extraído el 18 de Febrero del 2010 desde http://arantxa. ii. uam.es/dcama/lógica/recursos/fuzzy-into-esp. pdf*, page 6.
- Olivas, J. A., Montoro, A., y Lorenzo, A. (2023). Radicalización en redes sociales y consecuencias económicas de la pandemia. page 18.
- Rumelhart, D. (1986). David e. rumelhart, geoffrey e. hinton, and ronald j. williams learning representations by back-propagating errors nature 323: 533-536. *nature*, 323:533–536.
- Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., y Chen, X. (2016). Improved techniques for training gans. *Advances in neural information processing systems*, 29:2234–2242.