ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE CIENCIAS

MODELOS ESTADÍSTICOS PARA LA DETECCIÓN DE PATRONES EN MEDIO AMBIENTE Y FINANZAS

REDES NEURONALES RECURRENTES PARA LA DETECCIÓN DE FRAUDES EN TRANSACCIONES DE TARJETA DE CRÉDITO

TRABAJO DE INTEGRACIÓN CURRICULAR PRESENTADO COMO REQUISITO PARA LA OBTENCIÓN DEL TÍTULO DE INGENIERÍA MATEMÁTICA

ERICK SEBASTIÁN SIMBAÑA GUARNIZO
erick.simbana01@epn.edu.ec

Director: PH. D. MIGUEL ALFONSO FLORES SÁNCHEZ
miguel.flores@epn.edu.ec

QUITO D.M. , FEBRERO 2022

CERTIFICACIONES

Yo ERICK SEBASTIÁN SIMBAÑA GUARNIZO, declaro que el trabajo de integración curricular aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

Erick Sebastián Simbaña Guarnizo

Certifico que el presente trabajo de integración curricular fue desarrollado por ERICK SEBASTIÁN SIMBAÑA GUARNIZO, bajo mi supervisión.

Ph. D. Miguel Alfonso Flores Sánchez
Director del Proyecto

DECLARACIÓN DE AUTORÍA

A través de la presente declaración, afirmamos que el trabajo de integración curricular aquí descrito, así como el (los) producto(s) resultante(s) del mismo, son públicos y estarán a disposición de la comunidad a través del repositorio institucional de la Escuela Politécnica Nacional; sin embargo, la titularidad de los derechos patrimoniales nos corresponde a los autores que hemos contribuido en el desarrollo del presente trabajo; observando para el efecto las disposiciones establecidas por el órgano competente en propiedad intelectual, la normativa interna y demás normas.

ERICK SEBASTIÁN SIMBAÑA GUARNIZO

PH. D. MIGUEL ALFONSO FLORES SÁNCHEZ

AGRADECIMIENTOS

Agradezco a mis padres, hermanos, familiares, profesores en particular a mi tutor Miguel Flores, amigos y novia que contribuyeron a que llegue a la culminación de mis estudios universitarios . Sin ellos jamás hubiese cumplido mis sueños y no existen suficientes palabras para agradecer todo lo que han hecho por mi.

DEDICATORIA

Esta trabajo de integración curricular está dedicada principalmente a mis abuelitos, quienes han sido pilares importantes en mis destrezas y habilidades. Además, han sabido forjar una familia que tienen como carta de presentación la responsabilidad y compromiso ante cualquier dificultad.

Índice general

Resumen 1

Abstract 2

1. Descripción del componente desarrollado 3

1.1. Objetivo . . . 3

1.1.1. Objetivos Específicos . . . 3

1.2. Alcance . . . 4

1.3. Marco Teórico . . . 4

1.3.1. Redes Neuronales . . . 5

1.3.2. Redes Neuronales Recurrentes . . . 8

2. Metodología 16

2.1. Descripción de la Base de datos . . . 16

2.2. Preprocesamiento de los datos . . . 18

2.3. Datos Desbalanceados . . . 22

2.4. Modelos y capas de Redes Neuronales en Keras . . . 24

2.4.1. Modelos . . . 24

2.4.2. Capas . . . 26

2.4.3. Capas de Redes Neuronales Recurrentes . . . 30

2.4.4. Capas de Agrupación para topologías no lineales . . . 33

2.5. Creación del Modelo . . . 34

2.5.1. Compilación del Modelo . . . 36

2.5.2. Ajuste del Modelo 42

3. Resultados, conclusiones y recomendaciones **46**

3.1. Resultados 46

3.2. Conclusiones 50

3.3. Recomendaciones 51

4. Anexos **52**

4.1. Apéndice A 52

Bibliografía **60**

Resumen

En el presente trabajo se implementa y estudia algunos aspectos de las redes neuronales recurrentes en el ámbito de la detección de transacciones fraudulentas con tarjetas de crédito. Para ello, se comienza con un repaso de redes neuronales. Luego, se desarrollan los conceptos básicos de las redes neuronales recurrentes y para ilustrarlo se presentan una introducción al problema de la detección de fraudes en transacciones con tarjetas de crédito. Finalmente, se muestra un modelo de redes neuronales recurrentes con Keras, en la cual se exhibe los procedimientos de tratamiento de datos, como el uso de librería Keras para la creación de un modelo y los resultados del problema planteado.

Palabras claves: Keras, redes neurales recurrentes, tarjetas de crédito, detección de fraudes.

Abstract

In this work, some aspects of recurrent neural networks are implemented and studied in the field of detecting fraudulent transactions with credit cards. To do this, it begins with a review of neural networks. Then, the basic concepts of recurrent neural networks are developed and to illustrate it, an introduction to the problem of detecting fraud in credit card transactions is presented. Finally, a model of recurrent neural networks in Python is shown, in which the data treatment procedures are presented; such as the use of Keras' library for the creation of a model and the results of the problem posed.

Keywords: Keras, recurrent neural networks, credit cards, fraud detection.

Capítulo 1

Descripción del componente desarrollado

Se propone estudiar e implementar un modelo de Redes Neuronales Recurrentes con Keras; una librería de Python, para la detección de fraudes en transacciones de tarjetas de crédito, adecuando: la arquitectura, funciones de activación y función de pérdida del modelo; con el propósito de resolver este problema de clasificación binaria. Para ello, se utilizará una base de datos no balanceada con transacciones por tarjeta de crédito, que tenga en cuenta el orden de las transacciones realizadas por cada uno de los usuarios, es decir, el modelo tomará en cuenta tanto una sucesión ordenada de los montos realizados por cada uno de los usuarios, como la sucesión ordenada de las personas a las que va dirigida dichos montos.

1.1. Objetivo

- Implementar una Red Neuronal Recurrente para detectar fraudes en transacciones de tarjetas de crédito a través de la librería Keras de Python, utilizando datos de las transacciones consecutivas de los titulares de las tarjetas de crédito, para ajustar los diferentes parámetros de modelo.

1.1.1. Objetivos Específicos

1. Estudiar los conceptos básicos de las Redes Neuronales, además de las arquitecturas de las Red Neuronal Recurrente y LSTM.

2. Estudiar el problema de las transacciones fraudulentas con tarjetas de crédito.
3. Realizar tareas de minería para la base de datos, y efectuar una exploración de las funciones, clases y objetos de la librería Keras en Python, para implementar una RNN adecuada para el problema en cuestión.

1.2. Alcance

Este proyecto cubre de manera completa la implementación de modelos de redes neuronales con Keras, como también los conceptos elementales de las Redes Neuronales, con especial énfasis en arquitecturas recurrentes. Asímismo, describe apropiadamente los procesos de minería de datos para conjuntos de datos no balanceados. No obstante, la implementación y estudio de algoritmos como el descenso del gradiente estocástico, propagación hacia atrás y estructuras de datos eficientes para estos modelos están fuera del alcance de este proyecto, aunque se realiza una explicación del funcionamiento de los mismos.

1.3. Marco Teórico

El rápido crecimiento de la tecnología ha hecho cada vez más común el pago de bienes y/o servicios a través de crédito; es decir, un préstamo para el cliente realizado por una institución financiera mediante una tarjeta. A partir de la década de los 80, ha existido un aumento impresionante en el uso de tarjetas de crédito, débito y prepago a nivel internacional. En el Informe Nilson de Octubre de 2020, en el año 2019 más de 51 billones de dólares fueron generados por este método de pago.[33]

Debido a los nuevos sistemas de transferencia en línea y la expansión del comercio electrónico en todo el mundo, los pagos por crédito están llegando de manera masiva a nuevas poblaciones de consumidores, lo cual ha llamado la atención de los ciberdelincuentes. Según el Informe Nilson, las pérdidas mundiales por fraude con tarjetas se elevaron a 61000 millones de dólares en 2019 frente a los 8000 millones de dólares en 2010. Estos fraudes representan grandes costos de oportunidad y costos contables a las entidades financieras[33].

El fraude se puede analizar al ver datos secuenciales de transacciones que realizaron los clientes con anterioridad. Normalmente, los bancos u otras autoridades

de este servicio advierten a sus clientes si notan alguna desviación de los patrones disponibles, es decir, las entidades reconocen estos patrones como posibles transacciones fraudulentas.

El diseño y la implementación de un sistema eficiente para la detección de fraudes es esencial para reducir tales pérdidas. En los últimos años, se han utilizado varias técnicas del aprendizaje automático para resolver el problema; entre ellas están: las máquinas de soporte de vectores (MSV), los árboles de decisión (RF), los modelos ocultos de Markov (HMM), los modelos de espacio estado (SSM), entre otras.[3]

La mayoría de estos modelos tratan cada transacción como un objeto único y descuidan las relaciones entre ellas. Frente a este inconveniente, las redes neuronales recurrentes (RNN) han empezado a ser utilizadas en el campo de detección de fraudes, ya que tienen un enfoque de aprendizaje automático capaz de analizar los comportamientos dinámicos de varias cuentas bancarias, modelando la dependencia secuencial entre las transacciones consecutivas de los titulares de las tarjetas de crédito[3].

1.3.1. Redes Neuronales

Una **red neuronal** es un modelo matématico supervisado utilizado tanto para la predicción como para la clasificación de datos; consiste de tres componentes principales: la arquitectura, funciones de activación y la función de pérdida.

Sean V, A conjuntos no vaciós, la **arquitectura** es un grafo dirigido $D = (V, A)$ con tres conjuntos de nodos $X, H, Y \subset V$ disjuntos por pares que llamaremos capas y una función $W : A \longrightarrow R : (i, j) \longrightarrow c((i, j)) = w_{i,j}$ que denominaremos pesos de las aristas. La primera capa es la capa de entrada, donde la cardinalidad de los nodos es igual al número de variables independientes, es decir $X = (x_1, x_2, ..., x_n) \in \mathbb{R}^n$, donde para cada $i \in \{1, \ldots, n\}$, x_i representa un nodo. Además entre estos nodos no existen aristas.

Después de este proceso de entrada de datos, los nodos de la capa de entrada realizan un producto interno con los pesos de las aristas que se conectan a los nodos j de la siguiente capa H que denominaremos capa oculta:

$$\langle X, W_{X,j} \rangle = \sum_{i=1}^{n} x_i w_{i,j} \tag{1.1}$$

Esta capa esta compuesta por subcapas de nodos, es decir, $H = (H_1, H_2, ..., H_q)$ tal que cada $H_j = (h_1^{(j)}, ..., h_{p_j-1}^{(j)}, h_{p_j}^{(j)}) \in \mathbb{R}^{p_j}$ donde $h_i^{(j)}$ representa el nodo i de la subcapa j. Hay que señalar que los nodos de las subcapas son secuenciales, es decir los nodos de la subcapa H_i se conectan mediante un producto interno con los pesos de las aristas de la subcapa H_j si y solo si $i < j$:

$$\langle H_i, W_{H_i, H_j^{(k)}} \rangle = \sum_{i=1}^{p_j} h_i w_{i,k}^{(j)}. \tag{1.2}$$

Posteriormente los nodos de la última subcapa de H se multiplican con los pesos de las aristas que conectan a cada uno de los nodos de la última capa $Y = (y_1, y_2, ..., y_m) \in \mathbb{R}^m$, llamada capa de salida[4].

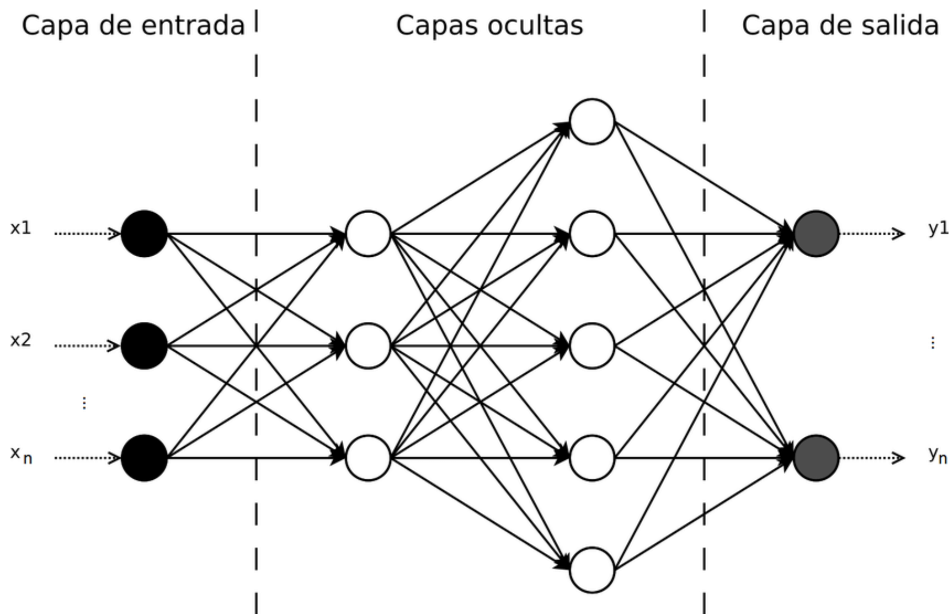

Figura 1.1: Ejemplo de un Red Neuronal Multicapa
Elaboración:Charu[1]

Dentro de los nodos de la capa oculta y capa de salida, despues de procesar los productos internos que ingresan a estos nodos, se activan funciones ϕ a las cuales llamaremos **funciones de activación**. Por ejemplo, dentro de las funciones de los nodos de la capa salida tenemos las siguientes funciones de clasificación o predicción:

- Para clasificación binaria, la última subcapa se une a un único nodo de salida que tiene como función denominada sigmoidea:

$$\phi(x) = \frac{1}{1 + e^{-x}} \tag{1.3}$$

- Para una clasificación múltiple de n etiquetas, la última capa tiene n nodos. Sea i un nodo de la última capa, se tiene que:

$$\phi(x_1, x_2, ..., x_j) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}. \tag{1.4}$$

- Finalmente, para la predicción se puede utilizar cualquier función continua que se adapte a los datos de salida, desde una función lineal, hasta una función trigonométrica.

Por el contrario, las funciones de las capas ocultas indican procesos que se realizan respecto a los datos de entrada, por ejemplo:

- Cuando se ingresa una imagen a una red neuronal, cada nodo de entrada es un pixel, en el cual, la entrada a la red neuronal es un matriz en la que se ubica números de 0 a 225 que representan una escala de grises. En estos casos se aplican funciones de activación denominadas **filtros**, que son operaciones que se realizan a ciertas filas, columnas o grupos de las mismas, como por ejemplo:

  Sea $P \in \mathbb{R}^{n \times n}$. Entonces $\forall i \in \{1, 2, 3..., n-2\}$:

$$\phi(p_{i,i}, p_{i+1,i}, p_{i,i+1}, p_{i+1,i+1}) = \frac{p_{i,i} + p_{i+1,i} + p_{i,i+1} + p_{i+1,i+1}}{4}. \tag{1.5}$$

Observacion.- Hay que notar que las redes neuronales tienen la posibilidad de ingresar datos como imágenes, sonido, frases, números y sucesiones; o incluso combinaciones de estos, a los cuales se los puede enlazar mediante funciones de activación en capas ocultas llamadas **capa de fusionamiento**.

Por último, al ser un modelo supervisado, como la regresión lineal, tenemos un problema de minimización de error respecto a los pesos de las aristas que son los parámetros del modelo, a dicha función de error la denominamos **función de pérdida** L, cuyas entradas son los valores reales y predichos por el modelo de red neuronal:

$$\min_{w_{i,j} \in A} L(\hat{y}, y). \tag{1.6}$$

Para la clasificación podemos utilizar como función de perdida la entropía cruzada, mientras que para la predicción cualquier norma de $\mathbb{R}^m$:

$$\textbf{Entropía cruzada} = L = -\sum_{i=1}^{m} y_i \log(\hat{y}_i) \tag{1.7}$$

Al ser un problema de optimización continuo buscamos las derivadas respecto a los pesos recurrentemente, desde las conecciones de la última subcapa oculta hacia la

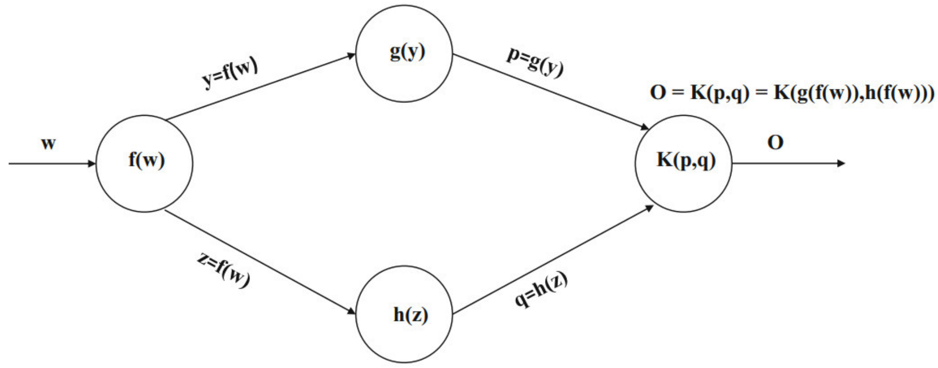

Figura 1.2: 2 capas ocultas de tamaño 1 y 2 con una nodo de entrada y salida **Elaboración**:Charu[1]

capa de salidas hasta la conección entre la capa de entrada con la primera subcapa oculta. La programación de este tipo de algoritmos se llama **propagación hacia atrás**[40].

A continuación, mostramos un ejemplo del mismo:

$$\begin{aligned}\frac{\partial o}{\partial w} =& \frac{\partial o}{\partial p} \cdot \frac{\partial p}{\partial w} + \frac{\partial o}{\partial q} \cdot \frac{\partial q}{\partial w} \\ &= \frac{\partial o}{\partial p} \cdot \frac{\partial p}{\partial y} \cdot \frac{\partial y}{\partial w} + \frac{\partial o}{\partial q} \cdot \frac{\partial q}{\partial z} \cdot \frac{\partial z}{\partial w} \\ &= \frac{\partial K(p,q)}{\partial p} \cdot g'(y) \cdot f'(w) + \frac{\partial K(p,q)}{\partial q} \cdot h'(z) \cdot f'(w).\end{aligned} \tag{1.8}$$

Agregando la función de pérdida obtenemos:

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial o} \cdot \left[\frac{\partial K(p,q)}{\partial p} \cdot g'(y) + \frac{\partial K(p,q)}{\partial q} \cdot h'(z) \right] \cdot f'(w). \tag{1.9}$$

1.3.2. Redes Neuronales Recurrentes

Las arquitecturas neuronales descritas anteriormente están diseñadas de forma inherente para datos multidimensionales en la que los individuos son en gran medida independientes entre sí. Sin embargo, ciertos tipos de datos, como series de tiempo, textos y datos biológicos, contienen dependencias secuenciales entre los individuos. Los valores individuales de una secuencia pueden ser números reales o simbólicos.

Observación.-Los valores simbólicos son representaciones ingeniosas de carácteres o palabras mediante codificacion tales como: one-hot o un número de 1 hasta el tamaño del diccionaro usado.

Los dos principales deseos para el procesamiento de secuencias incluyen:

1. La capacidad de recibir y procesar las entradas en el mismo orden en que están presentes en la secuencia.
2. El tratamiento de las entradas en cada instante de tiempo debe ser similar en relación con el historial previo de entradas.

Es decir, el desafío clave es construir una red neuronal con un número fijo de parámetros, pero con la capacidad de procesar un número variable de entradas.

Estos deseos se satisfacen con el uso de redes neuronales recurrentes. En una red neuronal recurrente existe una correspondencia uno a uno entre las capas de la red y las posiciones específicas de la secuencia. La posición en la secuencia también se conoce como su instante de tiempo. Con ello, en lugar de un número variable de entradas en una sola capa, la red contiene un número variable de capas y cada capa tiene una única entrada correspondiente a ese instante de tiempo. Por lo tanto, se permite que las entradas interactúen directamente con capas ocultas anteriores dependiendo de sus posiciones en la secuencia. Cada capa utiliza el mismo conjunto de parámetros para garantizar un modelado similar en cada instante de tiempo y, por ende, el número de parámetros también es fijo.

En otras palabras, la misma arquitectura de capas se repite en el tiempo y, por lo tanto, la red se denomina recurrente. Las redes neuronales recurrentes también son redes de retroalimentación con una estructura específica basada en la noción de capas de tiempo, de modo que pueden tomar una secuencia de entradas y producir una secuencia de salidas. Cada capa temporal puede tomar un punto de datos de entrada (ya sea un solo atributo o múltiples atributos) y, opcionalmente, producir una salida multidimensional. Dichos modelos son útiles para aplicaciones como la traducción automática o para predecir el siguiente elemento de una secuencia.

A continuación, se muestra la arquitectura de una red neuronal recurrente:

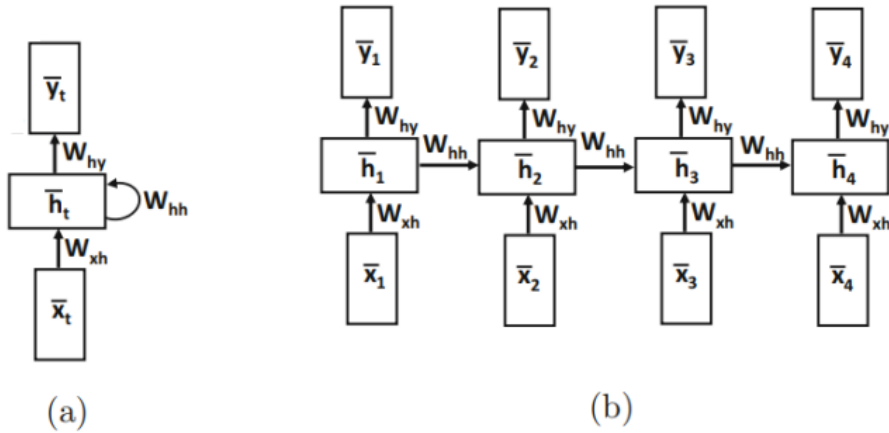

Figura 1.3: (a) Red Neuronal Recurrente y (b) Representación de (a) en capas de tiempo
Elaboración:Charu[1]

Los pesos en diferentes capas temporales se comparten para garantizar que se utilice la misma función en cada instante de tiempo. Las anotaciones $(W_{xh})_{p\times d}$, $(W_{hh})_{p\times p}$ y $\left(W_{hy}\right)_{p\times d}$ hacen evidente el intercambio.

Es importante destacar que la Figura [1.3] muestra un caso en el que cada capa de tiempo tiene una entrada, una salida y una unidad oculta. En la práctica, es posible que falten las unidades de entrada o de salida en cualquier capa de tiempo. La elección de las entradas y salidas que faltan dependerá de la aplicación específica en cuestión.

Por ejemplo, en el pronóstico de series de tiempo, es posible que necesitemos salidas en cada capa de tiempo para predecir el siguiente valor en la serie. Por otro lado, en una clasificación de secuencia, es posible que solo necesitemos un único nodo de salida al final de la secuencia correspondiente.

En general, los nodos de entrada, oculta y salida en el tiempo t son $(x_t)_d$,$(h_t)_p$ y $(y_t)_d$, respectivamente. Entonces, el nodo oculto en el tiempo t viene dado por una función del nodo de entrada en el tiempo t y el nodo oculto en el tiempo $(t-1)$:

$$\begin{aligned} h_t &= f(W_{xh}x_t + W_{hh}h_{t-1}), \\ y_t &= g(W_{hy}h_t). \end{aligned} \tag{1.10}$$

Esta función se define con el uso de matrices de pesos de aristas y funciones de activación, y se usan los mismos pesos en cada instante de tiempo. Por lo tanto, aunque el estado oculto evoluciona con el tiempo, los pesos y la función subyacente $f(\cdot,\cdot)$

permanecen fijos en cada instante de tiempo. Se usa una función separada $y_t = g(h_t)$ para aprender las probabilidades de salida de los estados ocultos. En el primer instante de tiempo, se asume que h_{t-1} es un vector constante predeterminado [5].

Tenga en cuenta que h_t es una función de $x_1, ..., x_t$. Dado que la salida y_t es una función de h_t, estas propiedades también son heredadas por y_t, es decir:

$$y_t = F_t(x_1, x_2, ..., x_t) \tag{1.11}$$

Redes Neuronales bidireccionales y multicapa

En la **red recurrente bidireccional**[6], tenemos capas ocultos separados $h_t^{(f)}$ y $h_t^{(b)}$ para las direcciones hacia adelante y hacia atrás. Las capas ocultas hacia adelante interactúan solo entre sí y lo mismo para los estados ocultos hacia atrás. Sin embargo, tanto $h_t^{(f)}$ como $h_t^{(b)}$ reciben entrada del mismo vector x_t e interactúan con el mismo vector de salida y_t. En la Figura 1.4 se muestra un ejemplo:

Las condiciones de recurrencia se pueden escribir de la siguiente manera:

$$\begin{aligned} h_t^{(f)} &= f(W_{xh}^{(f)} x_t + W_{hh}^{(f)} h_{t-1}^{(f)}) \\ h_t^{(b)} &= f(W_{xh}^{(b)} x_t + W_{hh}^{(b)} h_{t+1}^{(b)}) \\ y_t &= W_{hy}^{(f)} h_t^{(f)} + W_{hy}^{(b)} h_t^{(b)} \end{aligned} \tag{1.12}$$

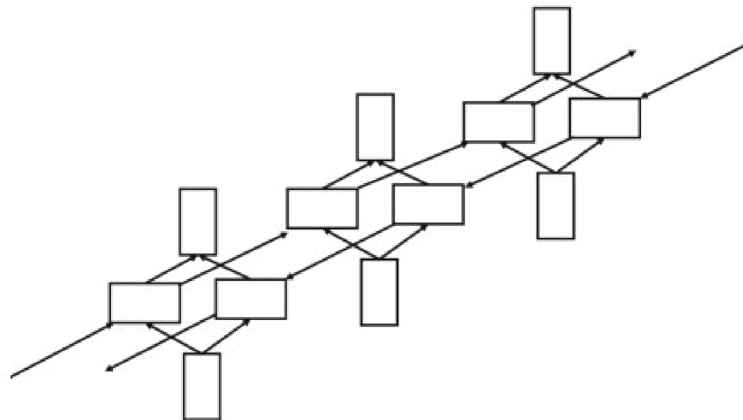

Figura 1.4: Red Neuronal Recurrente Bidireccional
Elaboración:Charu[1]

Una observación inmediata sobre las capas ocultas en la dirección hacia adelante y hacia atrás es que no interactúan entre sí en absoluto. Por lo tanto, primero se podría ejecutar la secuencia en la dirección de avance para calcular las capas ocultos

en la dirección de avance, y luego ejecutar la secuencia en la dirección de retroceso para calcular las capas ocultas en la dirección de retroceso. En este punto, la capa de salida se calculan a partir de los estados ocultos en las dos direcciones.

En general, las redes neuronales recurrentes bidireccionales funcionan bien en aplicaciones donde las predicciones se basan en un contexto bidireccional. Ejemplos de tales aplicaciones incluyen el reconocimiento de escritura a mano y el reconocimiento de voz, en los que las propiedades de los elementos individuales de la secuencia dependen de los que se encuentran a cada lado de la misma. Por ejemplo, si una escritura a mano se expresa en términos de trazos, los trazos a cada lado de una posición en particular son útiles para reconocer el carácter en cuestión.

Una red neuronal bidireccional logra casi la misma calidad de resultados que el uso de un conjunto de dos redes recurrentes separadas, una en la que la entrada se presenta en forma original y la otra en la que la entrada se invierte. La principal diferencia es que los parámetros de los estados hacia adelante y hacia atrás se entrenan conjuntamente en este caso. Sin embargo, esta integración es bastante débil porque los dos tipos de estados no interactúan directamente entre sí.

Hasta el momento se ha utilizado una arquitectura de red neuronal recurrente de una sola capa. Sin embargo, en aplicaciones prácticas, se utiliza una arquitectura multicapa para construir modelos de mayor complejidad. Además, estas arquitecturas se puede utilizar en combinación con variaciones avanzadas de las redes neuronales recurrentes, como la arquitectura LSTM. En la Figura 1.4 se muestra un ejemplo de una red profunda que contiene tres capas ocultas. Tenga en cuenta que los nodos de las capas de nivel superior reciben información de los de las capas de nivel inferior.

Por lo tanto, centrandonos en todas las capas ocultas k para $k \geq 2$. La condición de recurrencia para las capas con $k \geq 2$ también está en una forma muy similar a la ecuación 1.10:

$$h_t^{(k)} = f(W_{hh}^{(k-1)} h_t^{(k-1)} + W_{hh}^{(k)} h_{t-1}^{(k)}). \quad (1.13)$$

Memoria a corto plazo (LSTM)

Las redes neuronales recurrentes tienen problemas asociados a la función de pérdida. Este es un problema común donde la multiplicación sucesiva por la matriz $W^{(k)}$ es inherentemente inestable; dando como resultado que el gradiente del problema desaparezca durante la propagación hacia atrás, o que estalle a valores

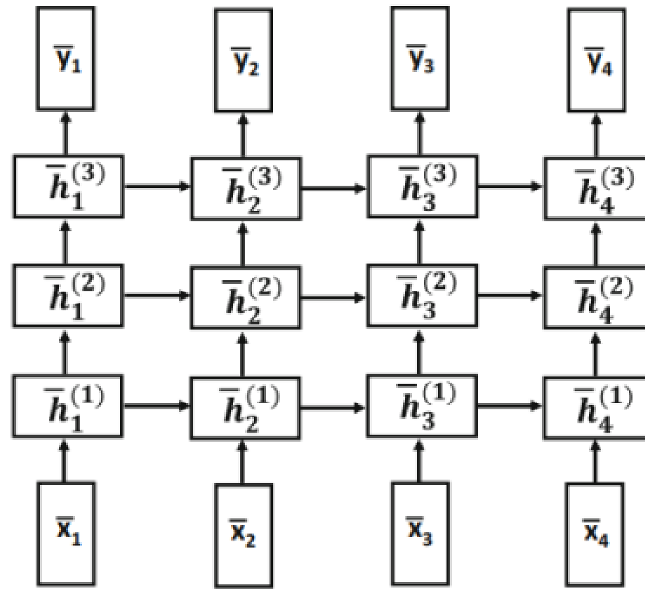

Figura 1.5: Red Neuronal Recurrente Multicapa
Elaboración:Charu[1]

grandes de manera inestable.

El LSTM es una mejora de la arquitectura de red neuronal recurrente. Para lograr este objetivo, tenemos un vector oculto p dimensional dentro de las subcapas ocultas, que se denota por $c_t^{(k)}$ y se conoce como el **estado de celda**. Se puede ver al estado de la celda como una especie de memoria a largo plazo que retiene al menos una parte de la información de estados anteriores mediante el uso de una combinación de operaciones de olvido e incremento parciales. Una forma de entender esto intuitivamente es que si los estados en diferentes capas temporales comparten un mayor nivel de similitud (memoria a largo plazo), es más difícil que los gradientes con respecto a los pesos entrantes sean drásticamente diferentes.[2]

Al igual que con la red recurrente multicapa, la matriz de actualización para las capas ocultas se denota por $\left(\bar{W}^{(k)}\right)_{4p\times 2p} = [W^{k-1}, W^k]$ y se utiliza para multiplicar previamente el vector $\left([h_t^{(k-1)}, h_{t-1}^{(k)}]^T\right)_{2p\times 1}$.En este caso, las actualizaciones utilizan cuatro variables vectoriales intermedias p-dimensionales i, f, o y c que corresponden al vector dimensional $4p$ resultante de la multiplicación. La determinación del vector de capa oculto $h_t^{(k)}$ y del vector de estado de celda $c_t^{(k)}$ utiliza un proceso de varios pasos para calcular primero estas variables intermedias, y luego calcular las

variables ocultas a partir de estas variables intermedias. Las actualizaciones son:

$$\begin{matrix}\text{Puerta de entrada:}\\ \text{Puerta de olvido:}\\ \text{Puerta de salida:}\\ \text{Nuevo C Estado:}\end{matrix} \begin{bmatrix} i \\ f \\ o \\ c \end{bmatrix} = \begin{pmatrix} sigm \\ sigm \\ sigm \\ tanh \end{pmatrix} \bar{W}^{(k)} \begin{bmatrix} h_t^{(k-1)} \\ h_{t-1}^{(k)} \end{bmatrix} \left[\text{Capas ocultas,}\right] \tag{1.14}$$

$$c_t^{(k)} = f \odot c_{t-1}^{(k)} + i \odot c \left[\text{Olvido y adicionamiento a la LTM,}\right] \tag{1.15}$$

$$h_t^{(k)} = o \odot tanh\left(c_t^{(k)}\right) \left[\text{Filtro selectivo de LTM a una capa,}\right] \tag{1.16}$$

Con

$$LTM := \text{Memoria a largo plazo.}$$

Aquí, el producto de vectores elemento a elemento se denota por $\odot$ y la notación $sigm$,$tanh$ denota una operación sigmoidea y tangente hiperbólica a cada una de las componentes de los vectores p dimensionales. En implementaciones prácticas, los sesgos también se utilizan en las actualizaciones anteriores, aunque se omiten aquí por simplicidad[35].

Observación.- El sesgo se define como un nodo con función de activación constante, que se estima en el proceso de minimización de la función de pérdida y una arista que sale del nodo con peso 1, dichos sesgos se pueden incluir en las subcapas ocultas o capa de salida.

Los vectores i, f y o se denominan **puertas de entrada, olvido y salida**. Estos vectores se utilizan conceptualmente como puertas booleanas para decidir si agregar u olvidar a un estado de celda, y si se permite la fuga de la información a una capa oculto desde un estado de celda . En la práctica, estas variables contienen un valor continuo en (0,1) por la diferenciabilidad de la función de pérdida.El vector c contiene los contenidos recién propuestos del estado de celda, aunque las puertas de entrada y olvido regulan cuánto se permite cambiar el estado de la celda anterior (para retener la memoria a largo plazo).

La Ecuación 1.15 tiene dos partes. La primera parte usa los p valores de olvido en f para decidir cuál de los p estados de la celda del instante de tiempo anterior restablecer a 0, y usa los p valores de entrada en i para decidir si agregar los componentes correspondientes de c a cada uno de los estados de la celda. Se puede ver al vector de estado de celda como una memoria a largo plazo continuamente actuali-

zada, donde los valores de olvido y entrada deciden respectivamente si restablecer los estados de la celda del instante de tiempo anterior y olvidar el pasado, y si incrementar los estados de celda del instante de tiempo anterior para incorporar nueva información en la memoria a largo plazo de la entrada actual.Además, el vector c contiene las p cantidades con las que incrementar los estados de celda.

Por último, la Ecuación 1.16 copia una forma funcional de cada uno de los p estados de celda en cada una de las p nodos ocultos, dependiendo de si la puerta de salida o es 0 o 1. Por supuesto, en la configuración continua de redes neuronales, se produce una activación parcial y sólo una fracción de la información se copia de cada estado de celda a la capa oculta correspondiente.

Capítulo 2

Metodología

Una vez conocido el marco teórico sobre las Redes Neuronales Recurrentes y el problema de las transacciones fraudulentas con tarjeta de crédito, el presente capítulo se encarga del modelamiento de los datos secuenciales de las transacciones de los títulares de las tarjetas de crédito desde el preprocesamiento de los datos hasta el modelo en sí. Asimismo, se presentarán las funciones a utilizar de la librería Keras de Python y en el Apéndice A, se encuentra el código realizado para este Capítulo.

2.1. Descripción de la Base de datos

En la página de Github[39] se dispone de datos de transferencias bancarias por tarjeta de crédito, dichos datos son diarias desde el 1 de abril hasta el 30 de septiembre del 2018, la información del primer mes se la utilizará para el modelo, ya que cada día tiene alrededor de 10000 observaciones.

La información proporcionada se presenta en un dataframe con 9 columnas y 287918 filas; recordemos que un dataframe es un objeto optimizado para el almacenamiento de datos de la librería Pandas de Python. Además, dicha estructura de datos no es conveniente para una Red Neuronal Recurrente, pues necesitamos obtener alguna estructura que guarde datos secuenciales.

A continuación, podemos observar en la Figura 2.1 los datos proporcionados por Github y una breve descripción de cada una de las variables:

	TRANSACTION_ID	TX_DATETIME	CUSTOMER_ID	TERMINAL_ID	TX_AMOUNT	TX_FRAUD
0	0	2018-04-01 00:00:31	596	3156	57.16	0
1	1	2018-04-01 00:02:10	4961	3412	81.51	0
2	2	2018-04-01 00:07:56	2	1365	146.00	0
3	3	2018-04-01 00:09:29	4128	8737	64.49	0
4	4	2018-04-01 00:10:34	927	9906	50.99	0
...	...	...	...	...	...	...
278297	278297	2018-04-29 23:52:15	691	392	11.24	0
278298	278298	2018-04-29 23:52:29	4974	7931	125.06	0
278299	278299	2018-04-29 23:54:19	1469	712	226.61	1
278300	278300	2018-04-29 23:55:31	510	9154	49.76	0
278301	278301	2018-04-29 23:56:06	4990	5421	112.88	0

287918 rows × 6 columns

Figura 2.1: dataframe con la información correspondiente
Elaboración: Autor

1. **TRANSACTION-ID**.- Representa el ID de la transacción por tarjeta de crédito.
2. **TX-DATETIME**.- Representa la fecha y hora en la que se realizó la transacción por tarjeta de crédito.
3. **CUSTOMER-ID**.- Representa ID del titular que realiza la transacción con tarjeta de crédito.
4. **TERMINAL-ID**.- Representa el ID de la persona que recibe la remuneración de la transacción por tarjeta de crédito.
5. **TX-AMOUNT**.- Representa el monto de la transacción por tarjeta de crédito.
6. **TX-FRAUD**.- Representa con 1 si la transacción es fraudulenta y 0 si no.

Las Redes Neuronales, en general, aceptan cualquier tipo de dato como: fotos, audio, video, secuencias, textos, etc. Sin embargo, estos datos son simbólicos, es decir, mediante alguna manera ingeniosa lo podemos codificar a tensores de números reales o números naturales.

Un tensor es un generalización de matrices, en la cual, el número de dimensiones puede ser cualquier número natural, intuitavamente se puede pensar a una matriz

o 2-tensor como un rectángulo de información y a un 3-tensor como un "paralelepipedo"de información. Por ejemplo:

- **0-tensor**.- $c \in \mathbb{R}$.
- **1-tensor**.- $v \in \mathbb{R}^n$ con dimension $n \in \mathbb{N}$.
- **2-tensor**.- $A \in \mathbb{R}^{n \times m}$ con dimensiones $n \times m \in \mathbb{N} \times \mathbb{N}$.
- **3-tensor**.- $A \in \mathbb{R}^{n \times m \times k}$ con dimensiones $n \times m \times k \in \mathbb{N} \times \mathbb{N} \times \mathbb{N}$.
- **p-tensor**.- $Q \in \mathbb{R}^{n_1 \times n_2 \times ... \times n_p}$ con dimensiones $n_1 \times n_2 \times ... \times n_p \in \mathbb{N}^p$.

En Python dichos objetos se los puede almacenar como tensores de la librería Tensorflow o arreglos de la librería Numpy ; que en efecto son las estructuras de datos para la entrada de datos en los modelos de Redes Neuronales de Keras.

De este modo, hay que tener en cuenta que los operaciones en estas estructuras de datos se las realiza elemento a elemento. En particular, si $\square$ representa cualquier operación númerica, se tiene que dicha operacion se aplica si:

$$a_{i,j,k} \square b_{l,m,n} \iff i = l \wedge j = m \wedge k = n \tag{2.1}$$

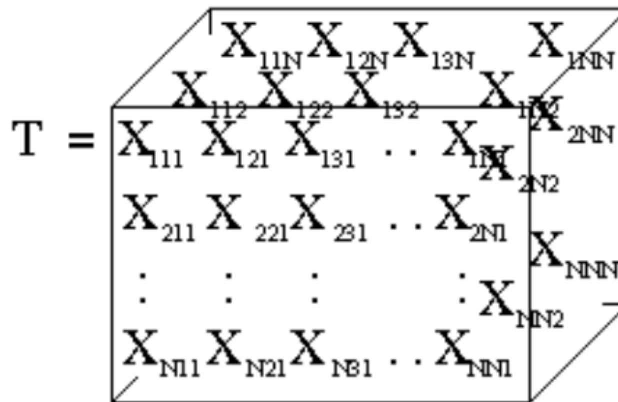

Figura 2.2: 3-tensor: de dimensiones (N, N, N)
Elaboración:Programador clic[32]

2.2. Preprocesamiento de los datos

Para nuestros fines comenzaremos normalizando mediante el método min-max[7] tanto para la variable TERMINAL-ID como para TX-AMOUNT. Esto debido a que

podemos encontrar problemas de convergencia al momento de ajustar nuestro modelo. El método min-max consiste en que $\forall i \in \{1, ..., q\}$:

$$\hat{X}_i = \frac{X_i - X_{min}}{X_{max} - X_{min}}, \tag{2.2}$$

donde X_{min} representa el mínimo valor de los datos y X_{max} el máximo valor de los mismos.

	CUSTOMER_ID	TERMINAL_ID	TX_AMOUNT	TX_FRAUD
0	596	0.315632	0.060334	0
1	4961	0.341234	0.086035	0
2	2	0.136514	0.154106	0
3	4128	0.873787	0.068071	0
4	927	0.990699	0.053821	0
...	...	...	...	...
278297	691	0.039204	0.011864	0
278298	4974	0.793179	0.132003	0
278299	1469	0.071207	0.239191	1
278300	510	0.915492	0.052523	0
278301	4990	0.542154	0.119147	0

287918 rows × 4 columns

Figura 2.3: Datos Normalizados
Elaboración: Autor

Luego de obtener la base de datos, se debe organizar la misma en una estructura secuencial; de manera que cada vector represente un secuencia temporal, es decir, los elementos de la posición n tengan afinadad con sus elementos del pasado o del futuro, dicho de otra forma, con los elementos de la posición $n-1$ o $n+1$ con $n \in \mathbb{N}$.

Entonces, con la información proporcionada del mes de abril de 2018 analizaremos los patrones de consumo de los titulares de las tarjetas de crédito; estos patrones los podemos identificar siempre que conozcamos a quien va dirigida la transacción como a que precio adquirió el servicio o el producto el titular de tarjeta de crédito.

Por ende, para cada ID de consumidor que simboliza al titular de la tarjeta de crédito que va a realizar la transacción, creamos tres arreglos de numpy, donde cada uno de estos, contenga la codificación de patrones de consumo, de tal forma que

tengamos secuencias de datos para cada uno de los titulares de las tarjetas.

Hay que señalar que los dos arreglos antes mencionados, TX-TERMINAL y TX-AMOUNT, representan a las variables independientes del modelo. Por consiguiente, creamos el tercer arreglo para la variable dependiente, que representa el fraude para cada titular de la tarjeta, utilizando la variable TX-FRAUD.

```
[9]  [3665 5106 5045 3220 2091 9816 431 978 2192 3744 6887 3345 5310 6303 9048
      117 6641 7538 6765 5167 6303 6641 3220 1050 5310 4045 6999 9586 115 3665
      6512 5416 5106 1738 9113 3878 9866 8451 937 5167 1274 891 1345 9113 5106
      4606 4606 1050 3695]
     [2.08 10.1 14.26 2.19 6.08 1.58 0.58 4.62 8.32 6.01 8.69 2.77 5.13 5.82
      6.73 4.32 3.52 15.17 6.02 5.75 8.6 6.82 10.37 4.21 9.16 7.18 10.92 8.47
      0.01 5.22 10.2 8.07 11.34 5.24 4.13 0.76 9.15 10.0 8.22 1.5 0.85 8.32
      9.64 8.2 9.97 10.73 10.73 6.69 5.95]
     [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
      0 1 0 0 0 0 0 0 0 0 0 0]

[16] [2171 1878 2988 9080 8113 3587 2171 7207 9861 3371 6591 3515 2346 6328
      2598 2171 9278 3181 3358 291 6556 3758 1314 1583 6591 5121 2171 7624 2988
      7808 4247 9487 3720 6900 2171 3358 88 4756 9861 4074 1453 7808 8073 4674
      2056]
     [84.23 62.15 86.78 35.35 100.91 71.35 99.63 59.33 74.64 84.49 69.36 121.5
      101.3 114.3 53.04 122.81 155.51 84.22 136.68 101.01 86.66 91.31 100.4
      62.27 79.97 25.75 53.42 59.78 79.73 57.8 50.94 114.31 76.71 88.48 55.33
      65.47 117.95 97.8 77.45 46.28 60.79 90.57 90.98 42.32 71.21]
     [0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0
      0 0 0 0 0 0 0 0]
```

Figura 2.4: Secuencias generadas del titular 9 y 16
Elaboración: Autor

Después de esta reestructuración de datos obtenemos 3194 observaciones para cada uno de los tres arreglos, donde cada uno de las observaciones representa una secuencia con una longitud que puede varian entre 40 y 49. En otras palabras, tenemos un arreglo de numpy de longitud 3194 y cada elemento del arreglo tiene un dimensión entre 40 y 49. En la figura 2.4 podemos notar que la observaciones sin normalización min-max [9] tiene una longitud de 49 y la observación sin normalización min-max [16] una longitud de 45. Asimismo, podemos representar la información como gráficos series de tiempo; a manera de ejemplo podemos ver esta representacione en la Figura 2.5.

Naturalmente surge la pregunta: Si existe un número fijo de capas para la red neuronal recurrente, ¿ Cómo trabajo con secuencias de diferente tamaño?

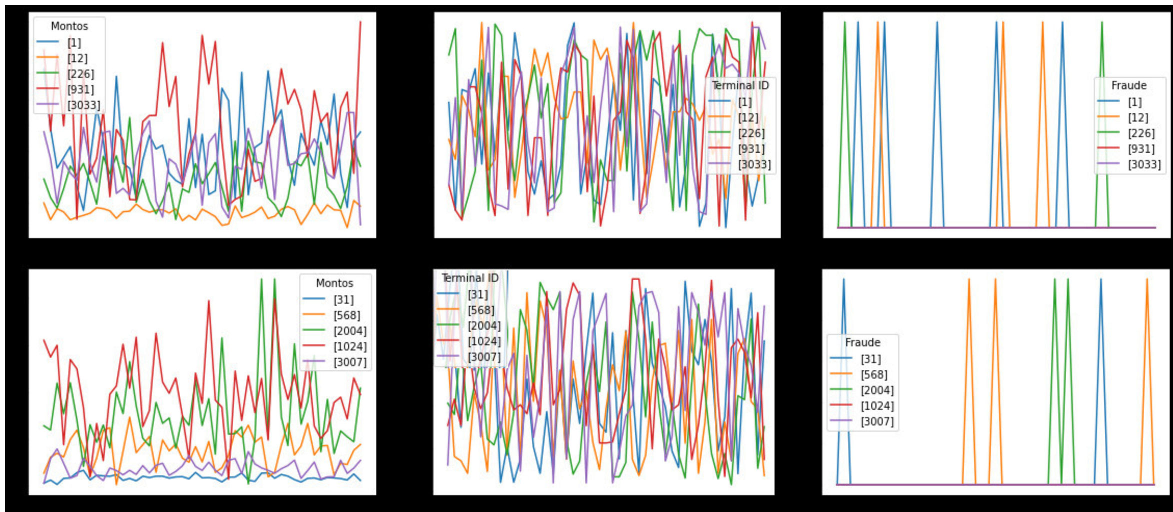

Figura 2.5: Representación de las secuencias como gráficos de series de tiempo
Elaboración: Autor

Keras nos da una solución, llamada el proceso de relleno o truncamiento que se encuentra en el atributo pad_sequences de la clase preprocessing[36]:

```
tf.Keras.preprocessing.sequence.pad_sequences(
    sequences, maxlen=None, dtype="int32", padding="pre", truncating="
   pre", value=0.0
)
```

Listing 2.1: Atributo preprocessing.sequence.`pad_sequences`

Dicho atributo por medio de *maxlen*, nos da la posibilidad de:

1. **Rellenar**

   Por ejemplo si tenemos un secuencia $x =$ | 1 | 2 | 3 | 4 | 5 | 6 | y la longitud máxima de la totalidad de secuencias es 10, mediante padding podemos agregar un valor determinado, ya sea al principio o final de la secuencia.

   (x,maxlen="10",padding="pre", value=-1) ≡ | -1 | -1 | -1 | -1 | 1 | 2 | 3 | 4 | 5 | 6 |

   (x,maxlen="10",padding="post", value=-1) ≡ | 1 | 2 | 3 | 4 | 5 | 6 | -1 | -1 | -1 | -1 |

   - O por el contrario:

2. **Trucar**

   Es decir, si tenemos un secuencia $x =$ | 1 | 2 | 3 | 4 | 5 | 6 | y la longitud máxima de la totalidad de secuencias es 4, mediante truncating podemos quitar valores, ya sea al principio o final de la secuencia.

(x,maxlen="4",truncating="pre") ≡

3	4	5	6

(x,maxlen="4", truncating="post") ≡

1	2	3	4

Con este precedente, cualquiera de las dos nociones pueden ser convenientes para nuestros propositos, dicho de otro modo, el proceso de truncamiento con máxima longitud 40 o el proceso de relleno con máxima longitud 49 son aceptables. Sin embargo, los resultados pueden ser diferentes.

Ahora bien, otro de los problemas a resolver es el desequilibrio entre las clases de la variable dependiente a estudiar, que se presentan en este tipo de problemas. Por tanto, antes de aplicar el proceso de relleno o truncamiento necesitamos tener datos balanceados.

2.3. Datos Desbalanceados

Los datos desbalanceados se dan en problemas de clasificación; específicamente cuando una de las clases de la variable dependiente tiene un cantidad de observaciones mucho menor a las otras.

Por ejemplo, supongamos que tenemos 10000 observaciones de la variable dependiente **Es_moroso**, que tiene dos clases: 1 si es moroso y 0 si no, igualmente supongamos que la clase 1 tiene 1000 observaciones y 0 tiene 9000.

Podemos notar que el 90 % de nuestros datos son 0 y solo el 10 % son 1. En estos casos los modelos de aprendizaje profundo tienden a clasificar a todos los datos en la clase con mayor observaciones; en este caso 0. Por tanto, para resolver estos problemas tenemos el siguiente conjunto de métodos para problemas de aprendizaje profundo:

Muestreo sobre la clase mayoritaria

Este método consiste en tomar una muestra de la clase mayoritaria, cuyo tamaño sea igual a la clase minoritaria.

Por ejemplo, si quisieramos aplicar el método de muestreo sobre la clase mayoritaria a tres clases $1, 2, 3$ con 9000,8000 y 1000 observaciones respectivamente, tendremos que realizar un muestreo simple de tamaño 1000 tanto a la clase 1 como la clase 2.

Muestreo sobre la clase minoritaria

Este método consiste en tomar una muestra de la clase minoritaria, hasta poder llegar a un tamaño de observaciones igual a la clase mayoritaria.

Por ejemplo, si quisieramos aplicar el método de muestreo sobre la clase minoritaria a tres clases $1, 2, 3$ con 9000,8000 y 1000 observaciones respectivamente, tendremos que realizar 8 y 9 muestreos simples de tamaño 1000 tanto a la clase 1 como la clase 2.

Existen otros métodos llamados técnicas de sobremuestreo de minorías sintéticas o **SMOTE** [34] . Consiste en seleccionar ejemplos cercanos en el espacio de las variables dependientes para la clase minoritaria. Por tanto, SMOTE utiliza técnicas del aprendizaje automático como k_medias y bosques aleatorios. Sin embargo, estas técnicas se utilizan para variables independientes entre sí y no pueden utilizarse para datos secuenciales.

El estudio de técnicas para balancear datos secuenciales es un territorio fértil para la investigación, por ende, utilizaremos el método de muestreo sobre la minoritaria y además utilizaremos un atributo llamado class_weight, que se estudiará al momento de ajustar el modelo.

Por tanto, de las 3194 secuencias de la variable dependiente TX-FRAUD, la variable que indica si existe o no fraude, 2842 son sucesiones que son solo ceros mientras que 352 de las secuencias tienen al menos una transacción fraudulenta. En otras palabras, el 11.02 % de los datos de la variable dependiente presentan al menos una transacción fraudenta y el 88.98 % no.

En resumen, de los 3194 datos tanto de la variables dependiente TX-FRAUD como de las variables independientes TERMINAL-ID y TX-FRAUD; pasamos a tener 5658 secuencias para cada una de las variables, en donde la variable dependiente ahora tiene 2842 secuencias, en las cuales, no se presenta ninguna transaccion fraudulenta y 2816 secuencias que tienen al menos una transacción fraudulenta.

Con todo este estudio de preprocesamiento de datos procedemos a estudiar la creación de las redes neuronales dentro de Keras, empezando con el estudio de los **modelos** implementados en esta librería.

2.4. Modelos y capas de Redes Neuronales en Keras

2.4.1. Modelos

Keras nos permite utilizar 2 tipos de modelos, en los cuales se pueden implementar redes neuronales multicapa, convolucionales, recurrentes, entre otras.

El primero de ellos es Sequential[29]:

```
Keras.Sequential([
        capa de entrada,
        capas ocultas,
        capa de salida,
    ])
```

Listing 2.2: Sequential

Este modelo es eficiente cuando tenemos topologías lineas. Una topología lineal hace referencia a que la red solo va a tener un tipo de entrada, un tipo de salida y no puede compartir capas; con **un tipo** nos referimos a que las entradas pueden ser únicamente una secuencia, un vector de $\mathbb{R}^n$, o una matriz. No obstante, cuando trabajamos con vectores cada elemento del mismo puede representar variables diferentes. Por ejemplo, un vector en $\mathbb{R}^3$ la primera componente puede ser el precio de electrodomésticos, la segunda el IVA de video juegos y la tercera el precio de descuento de cosas de hogar; mientras que en matrices y secuencias no es así.

En matrices y secuencias cambia está noción, pues por ejemplo, en el análisis de patrones en dos tipos de secuencia, como en nuestro caso el monto y el ID no se las puede incluir como una misma entrada. Debido a que se necesita una LSTM o RNN para analizar el patrón de cada una de ellas, lo cual es imposible en Sequential, ya que permite una única entrada.

Por último, cuando hablamos de que no se pueden compartir capas no referimos a que dos o más capas distintas no se pueden unirse a una capa, o viceversa. Es decir, existe una unica conexión entre dos capas distintas.

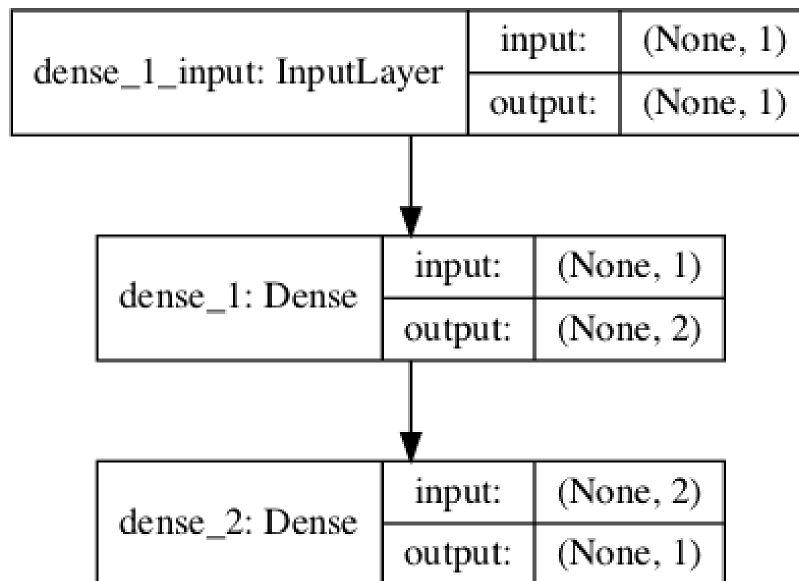

Figura 2.6: Ejemplo de un modelo Sequential
Elaboración: Keras[29]

El siguiente modelo resuelve los limitaciones de las topologías lineales que presenta Sequential.

```
Keras.Model([
        capa de entrada,
        capas ocultas,
        capa de salida
    ])
```

Listing 2.3: Sequential

Model[23] trabaja con topologías no lineales, lo que se acopla de manera perfecta para el tipo de datos que tenemos. Cabe señalar que Keras tiene la opción de crear sus propios modelos para tareas específicas.

Además, es esencial considerar que los modelos implementados en Keras, contienen un atributo llamado summary que nos permite obtener el número de parámetros que tiene nuestro modelo en cada capa.

Con las nociones explicadas anteriormente podemos describir las capas que tiene incorporada Keras, con especial enfásis en aquellas que utilizaremos para el modelamiento del problema de las transacciones fraudentas con tarjetas de crédito.

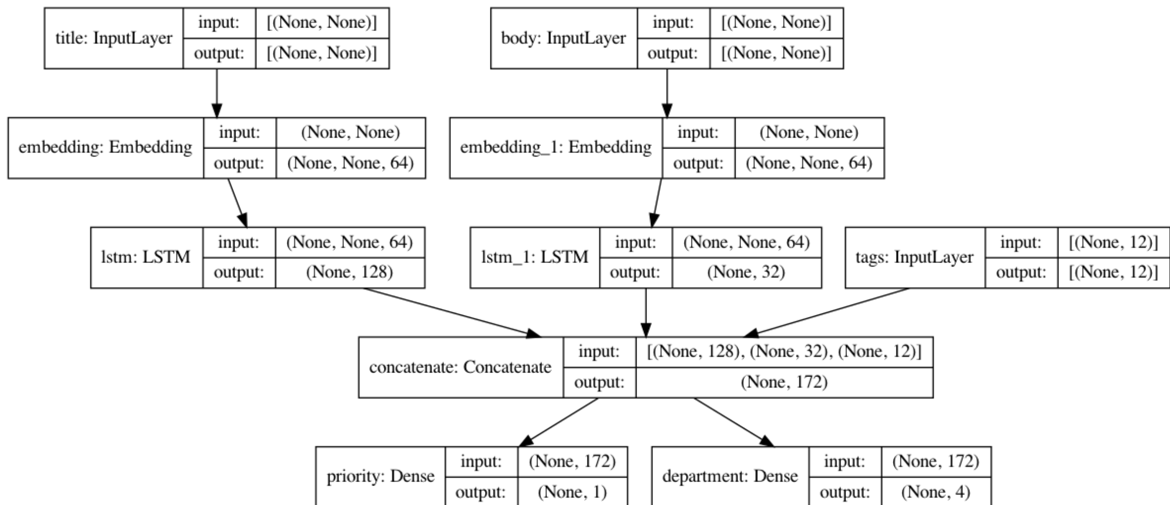

Figura 2.7: Ejemplo de un modelo Model
Elaboración: Keras [23]

2.4.2. Capas

Las capas[18] son los componentes básicos de las redes neuronales en Keras. Una **capa** es un objeto invocable que consta de una función de activación, tensor de entrada, tensor de salida y los pesos de la capa. Entre los principales atributos que tienen las capas tenemos:

```
tf.Keras.layers.Capa_a_usar(
    units= int32,
    activation='relu',
    kernel_initializer='random_normal',
    bias_initializer='zeros',
    kernel_constraint=max_norm(2.)
)
```

Listing 2.4: Clase Capa

Observación.- Un objeto invocable es cualquier objeto que puede llamarse como una función de una clase predeterminada.

Además, consideremos que en Keras existe la posibilidad de crear nuevas capas dependiendo del modelo que se quiera realizar; no obstante, Keras ofrece capas en las cuales vienen parámetros específicos para las diferentes tareas que se quiera realizar, como también, los parámetros que vienen por defecto en todas las capas. Estos últimos se encuentran en el listing 2,4.

1. **units** .- Indica el número de neuronas que se encuentran en la capa.

2. **activation**.- Indica que función de activación van a tener los nodos de la capa, entre ellas se encuentran: relu, sigmoid, softmax, softplus, softsign, tanh, selu, elu,exponential. Este proceso es opcional, por defecto, si no se incluye este parámetro la función a utilizar es la identidad.

3. **kernel_initializer** .- Indica el conjunto de pesos a utilizar antes del entranamiento de la red, entre ellas se encuentran: random_normal,random_uniform, truncated_normal, zeros, ones, glorot_normal, glorot_uniform,he_normal,he_uniform, identity,orthogonal, constant, variance_scaling. Este proceso es opcional, por defecto, empiezan siendo 0 los pesos de las aristas

4. **bias_initializer** .- Indica el conjunto de sesgos a utilizar antes del entranamiento de la red,entre ellas se encuentran: random_normal,random_uniform, truncated_normal, zeros, ones, glorot_normal, glorot_uniform,he_normal,he_uniform, identity,orthogonal, constant, variance_scaling. Este proceso es opcional, por defecto, empiezan siendo 0 los pesos de las aristas

5. **kernel_constraint** .- Permiten establecer restricciones (por ejemplo, no negatividad) en los parámetros del modelo durante el entrenamiento, entre ellas se encuentran: max_norm, minmax_norm,non_neg, unit_norm, radial_constraint.

Naturalmente estas funciones no permiten el aprendizaje profundo en las redes neuronales, sino las configuraciones y operaciones que se realizan en las capas de nuestra red. Por tanto, realizamos una breve descripción de las capas que Keras nos ofrece para realizar los modelos supervisados. Se asumirá que dentro de estas capas estan implicitas las funciónes anteriormente descritas.

Capa de Entrada

La capa de entrada[17] esta conformada por los nodos que ingresan al modelo de red neuronal:

```
tf.Keras.Input(
    shape=None,
    name=None
)
```

Listing 2.5: Capa de Entrada

En esta capa, shape representa el tensor de entrada, si se especifica None, representa que las dimensiones se desconocen. Por ejemplo, shape $= (3, 2, 5,)$

$$\begin{bmatrix}\begin{bmatrix}\begin{bmatrix}0 & 1 & 2 & 3 & 4\end{bmatrix}\\ \begin{bmatrix}5 & 6 & 7 & 8 & 9\end{bmatrix}\end{bmatrix}\\ \begin{bmatrix}\begin{bmatrix}0 & 1 & 2 & 3 & 4\end{bmatrix}\\ \begin{bmatrix}5 & 6 & 7 & 8 & 9\end{bmatrix}\end{bmatrix}\\ \begin{bmatrix}\begin{bmatrix}0 & 1 & 2 & 3 & 4\end{bmatrix}\\ \begin{bmatrix}5 & 6 & 7 & 8 & 9\end{bmatrix}\end{bmatrix}\end{bmatrix}$$

Finalmente, en el ejemplo después de 5 se coloca la coma, ya que, se establece que los lotes para el entrenamiento se los colocará al momento de ajustar el modelo.

Observación.- Los lotes se utliza cuando tenemos una gran cantidad de datos, en otras palabras si tenemos 2 lotes, al momento del ajuste del modelo se lo realizará primero con la mitad de los datos y en segunda instancia con la otra mitad de los datos.

Capa Densa

La capa densa[14] implementa la siguiente operación:

$$\text{salida} = \text{activacion}(\text{producto}(\text{entrada}, \text{pesos}) + \text{sesgo}),$$

donde activacion es la función de activación, pesos es la matriz de pesos de las aristas creada por la capa y sesgo un vector de sesgo creado por la capa (solo se aplica si lo use_biases True). Todos estos son atributos de Dense.

```
tf.Keras.layers.Dense(
    use_bias=True
    )
```

Listing 2.6: Capa Densa

Además, hay que considerar que el número de parámetros es igual a la multiplicación de los nodos de la capa anterior con esta capa más el número de nodos de esta capa densa. En particular, supongamos que tenemos una entrada de 32 dimensiones y una capa densa de 16 unidades, entonces el número de parámetros será 528.

Capa de incrustación

Esta capa de incrustación[16] solo se puede utilizar como la primera capa oculta en un modelo, las entradas de esta función son input_dim que representa el número valores únicos en un arreglo, mientras que output_dim representa el tamaño del arreglo de salida capa e input_length representa el tamaño del arreglo de ingreso a la capa. También, hay que considerar que la entrada a esta capa es un tensor de dos dimensiones, que es el tamaño del lote que no se específica y input_dim, y como salida se obtiene un tensor de tres dimensiones, que son el tamaño del lote, la input_dim y output_dim.

Por último, mask_zero es un valor booleano, si el valor es True, quiere decir que existe un valor de relleno"(0) especial que no debe tomarse en cuenta. Esto es útil cuando se utilizan capas recurrentes que pueden requerir una entrada de longitud variable.

```
tf.Keras.layers.Embedding(
    input_dim,
    output_dim,
    mask_zero=False,
    input_length=None
)
```

Listing 2.7: Capa de Incrustación

Adicionalmente hay que considerar que el número de parámetros es igual a la multiplicación de input_dim por output_ dim.

Capa de enmascaramiento

La capa de enmascaramiento[21] no toma en cuenta un valor predeterminado en las secuencias que se usan en las Redes Neuronales Recurrentes. Dicho valor que no se toma en cuenta se lo establece con mask_values y en esta capa no existen parámetros.

```
tf.Keras.layers.Masking(mask_value=-1.0)
```

Listing 2.8: Capa de Enmascaramiento

Capas de Convolución y Agrupación

Estas capas se las utiliza particularmente cuando las entradas a la red son imágenes o videos, las de convolución para realizar operaciones y las de agrupación para reducir el número de elementos en los tensores en Keras. Existen las siguientes capas:

Capas de Convolución

Conv1D layer, Conv2D layer, Conv3D layer, SeparableConv1D layer, SeparableConv2D layer, DepthwiseConv2D layer, Conv2DTranspose layer, Conv3DTranspose layer.

Capas de Agrupación

MaxPooling1D layer, MaxPooling2D layer, MaxPooling3D layer, AveragePooling1D layer, AveragePooling2D layer, AveragePooling3D layer, GlobalMaxPooling1D layer, GlobalMaxPooling2D layer, GlobalMaxPooling3D layer, GlobalAveragePooling1D layer, GlobalAveragePooling2D layer, GlobalAveragePooling3D layer,

Para más información consulte en [13] y [26], respectivamente.

2.4.3. Capas de Redes Neuronales Recurrentes

Capa Red Neuronal Recurrente Simple

Esta capa[28] muestra a las redes neuronales recurrentes vistas en el capítulo anterior a detalle. Por lo tanto, presentamos las entradas de la función:

```
tf.Keras.layers.SimpleRNN(
    units= None,
    use_bias=True,
    dropout=0.0,
    recurrent_dropout=0.0,
    return_sequences=False,
    return_state=False,
    go_backwards=False,
    stateful=False,
)
```

Listing 2.9: Capa Red Neuronal Recurrente Simple

En ella, podemos ver el uso de sesgo con use_bias, asimismo dropout y recurrent_dropout se utilizan para la eliminación aleatorio de nodos de la capa de entrada y oculta respectivamente, por tanto, en estas funciónes se establece un número entre 0 y 1 para establecer la fracción de nodos a olvidar.

De la misma manera, la funcion return_sequences y return_state si son verdaderas retornan los valores de los nodos de salida y nodo final de la capa oculta respectivamente, mientras que si go_backwards es verdadero hace que la secuencia de entrada se la tome a la inversa y si stateful es verdadero toma como h_0 el último estado oculto de la anterior secuencia de entranimiento en la red.

De este modo, procedemos a explicar como es la entrada y salida de los datos a esta capa. Para la entrada de datos tenemos un tensor de 3 dimensiones (k, n, m) donde k representa el tamaño del lote, n el tamaño de caracteres diferentes,por ejemplo para el modelado del lenguaje representa el número de palabras en un diccionario, y m el número de de nodos de la capa entrada. Y la salida de datos es igual a m.

Finalmente el número de parámetros se calcula como:

$$(m + 1 + units) * units.$$

LSTM

Esta capa[20] muestra a la LSTM vista en el capítulo anterior a detalle. Por lo tanto, presentamos las entradas de esta función:

```
tf.Keras.layers.LSTM(
    units,
    use_bias=True,
    dropout=0.0,
    recurrent_dropout=0.0,
    return_sequences=False,
    return_state=False,
    go_backwards=False,
    stateful=False
)
```

Listing 2.10: LSTM

Realiza las mis funciones que una red neuronal recurrente simple solo que return_state si es verdadero, retorna el ultimo valor de la capa de olvido c_t. Finalmente, el núme-

ro de parámetros se calcula como:

$$4(m + 1 + units) * units$$

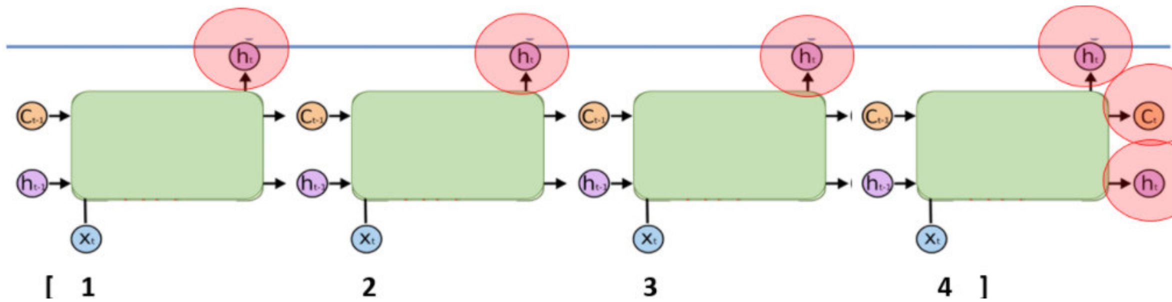

Figura 2.8: return_sequence= True y return_state = True en LSTM
Elaboración:Autor

Capa Distribuida en el tiempo

TimeDistributed[31] permite aplicar la misma capa en cada paso de tiempo, es decir, si tenemos un lote de 32 muestras, cada muestra tiene 10 palabras y cada palabra está representada por un vector de 32 dimensiones, por lo que la entrada es (32, 10, 16); podemos utilizar TimeDistributed para aplicar la capa Densa de forma independiente a cada uno de estos 10 pasos de tiempo.

Por ejemplo, en un video, es posible que desee aplicar la misma operación convolucional a cada cuadro.

```
tf.Keras.layers.TimeDistributed(capa)
)
```

Listing 2.11: Tiempo Distribuido

Capa Bidireccional

Recordemos que la capa bidireccional[9] permite realizar un análisis de atrás hacia adelante y de adelante hacia atrás de una secuencia.

```
tf.Keras.layers.Bidirectional(
    capa,
    merge_mode="concat"
```

```
4 )
```

Listing 2.12: Bidireccional

Por tanto, está capa como la capa distribuida en el tiempo se aplican a capas recurrentes como la LSTM.

Sin embargo, en la capa bidireccional existe la función merge_mode que permite combinar las salidas de las redes neuronales recurrentes hacia adelante y hacia atrás. Dichas combinaciones pueden ser: suma, multiplicación, concatenar, Promedio, Ninguno. Si es Ninguno, las salidas no se combinarán, se devolverán como una lista. El valor predeterminado es concatenar.

2.4.4. Capas de Agrupación para topologías no lineales

Capa de concatenar

Esta capa[12] concatena capas previas, donde la concatenación se la puede hacer en cuaquier dimensión de los tensores de entrada mediante axis.

```
1 tf.Keras.layers.Concatenate(axis=-1)([capa_1,..., capa_n])
```

Listing 2.13: LSTM

En otras palabras, si contamos de dos capas para concatenar la capa_1 con una dimensión de 32 y y la capa_2 con una dimensión de 15 la capa concatenada tendra una dimensión 47.

Capa de Operación

Las capas de operación[22] son un conjunto de capas de agrupación donde se produce una operación entre las funciones de activación de los nodos de las capas a juntar o agrupar, entre estas capas tenemos:

```
1 tf.Keras.layers.Multiply()
2 tf.Keras.layers.Average()
3 tf.Keras.layers.Maximum()
4 tf.Keras.layers.Minimum()
5 tf.Keras.layers.Subtract()
6 tf.Keras.layers.Add()
```

Listing 2.14: LSTM

2.5. Creación del Modelo

En las secciones anteriores, realizamos una breve descripción de los problemas que podemos encontrar en los datos, como también, una breve explicación de los modelos y capas de la librería Keras. Con este conocimiento preliminar, realizamos la creación de una arquitectura ajustada a los datos y apropiada para solucionar el problema de las transacciones fraudulentas.

Dado que en la sección de **Datos desbalanceados** obtuvimos una proporción similar, tanto de las sucesiones que tienen al menos una transacción fraudulenta como aquellas que no presentan ninguna; procedemos a realizar un relleno a posterior para obtener sucesiones de tamaño fijo igual a 49, con un valor de relleno -1 en las sucesiones que representar las variables independientes y con un valor de relleno 1 en la sucesión que representa la variable dependiente.

Presentamos a continuación el código:

```
padded_inputs1 = tf.Keras.preprocessing.sequence.pad_sequences(
    e, padding="post", maxlen=None, dtype='int32', value=-1.0)
padded_inputs2 = tf.Keras.preprocessing.sequence.pad_sequences(
    f, padding="post", maxlen=None, dtype='float32', value=-1.0)
padded_outputs = tf.Keras.preprocessing.sequence.pad_sequences(
    g, padding="post", maxlen=None, dtype='int32', value=1.0)
```

Listing 2.15: Padding

Ahora, puesto que la entrada para una red neural recurrente es un tensor de 3 dimensiones realizamos un redimensionamiento a nuestras entradas, pasamos de tener un tensor de dimensiones 5658×49 por un tensor de $5658 \times 1 \times 49$ mediante el siguiente código:

```
x_train1 = padded_inputs1.reshape(-1,1,49)
x_train2  = padded_inputs2.reshape(-1,1,49)
y_train  = padded_outputs.reshape(-1,1,49)
```

Listing 2.16: Reshape

Notemos que la segunda dimensión es 1, ya que "no tenemos diccionario". Un diccionario significa poder representar a cada elemento de las sucesiones del problema mediante codificaciones ingeniosas; uno de ellos es la codificación one-hot, la que consiste en representar cada elemento de la sucesión como una posición en un vector cuya dimensión es el tamaño del diccionario. Por ejemplo, supongamos que

tenemos un problema con 2 instancias que son:

El gato tiene grandes ojos

Los perros son grandes

Por ende, en este caso el tamaño del diccionario es 8, donde:

$\{1 = "El", 2 = "gato", 3 = "tiene", 4 = "grandes", 5 = "ojos", 6 = "Los", 7 = "perros", 8 = "son"\}$

Con la cual obtenemos la siguente representación one-hot:

$$\left[\left[\left[\begin{matrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{matrix}\right]\right]\right]$$

Sin embargo, dichas codificaciones no son obligatorias, en casos como las series de tiempo y datos con números reales no son necesarios. Por tal motivo, nuestro diccionario no tiene elementos. Además, en el Listing 2.16 el -1 en el redimensionamiento se lo utiliza cuando no sabe o no quiere decir explícitamente la dimensión de ese eje. Por ejemplo, si tiene una matriz de forma (2,4) y luego la remodelará con (-1, 1), entonces la matriz se remodelará de tal manera que la matriz resultante tiene solo 1 columna y esto solo es posible si se tiene 8 filas, por lo tanto, (8,1).

Luego del redimensionamiento creamos el modelo:

```
monto_input = tf.Keras.Input(shape=(1,49,), name="monto")

receptor_input = tf.Keras.Input(shape=(1,49,), name="receptor")

monto_capa = tf.Keras.layers.Masking(mask_value=-1.0)(monto_input)

receptor_capa = tf.Keras.layers.Masking(mask_value=-1.0)(
    receptor_input)
```

```
8
9  monto_capa = tf.Keras.layers.Bidirectional(tf.Keras.layers.LSTM(49,
       return_sequences =True, input_shape = (49,1)))(monto_capa)
10
11 receptor_capa = tf.Keras.layers.Bidirectional(tf.Keras.layers.LSTM(49,
       return_sequences =True, input_shape = (49,1)))(receptor_capa)
12
13 x = tf.Keras.layers.average([monto_capa, receptor_capa])
14
15 x = tf.Keras.layers.TimeDistributed(tf.Keras.layers.Dense(49))(x)
16
17 clase_pred = tf.Keras.layers.Dense(49, activation='sigmoid' ,name="
       clase")(x)
18 model = tf.Keras.Model(
19     inputs=[monto_input,receptor_input],
20     outputs=[clase_pred],
21 )
```

Listing 2.17: Modelo para la Detección de Transacciones Fraudulentas

Se puede apreciar que existen dos capas de entradas, por ende, se trata de un modelo neuronal con topología no lineal, en los cuales las primeras capas para las dos entradas son capas de enmascaramiento con valor -1, luego a cada capa se la asocia una red neuronal bidireccional LSTM, donde return_sequence = True, nos da a entender que se utilizarán todos los nodos de la capa de salida ,y no solo el último nodo de la secuencia de la variable dependiente. Además, no se retorna el último estado de la capa de olvido, debido a que, en general se lo utiliza como h_0, peso inicial de la capa oculta, cuando tenemos más de una red neuronal recurrente concatenadas entre sí.

Adicionalmente, podemos notar que (monto_capa) y (receptor_capa) son la manera de asociar capas a las diferentes ramas de la topología no lineal del modelo; así realizamos un agrupación mediante la capa de promedio, y finalmente utilizamos dos capas densas una de ellas con tiempo distribuido, ya que en cada paso de tiempo se aplicará una capa densa. Con ello, especificamos con tf.Keras.Model, las entradas al modelo, como también la salida del mismo.

2.5.1. Compilación del Modelo

La compilación del modelo consiste en establecer el proceso del aprendizaje de los parámetros, escoger la función de pérdida apropiada para el modelo en cuestión

y la métrica con la cual queremos evaluar nuestro modelo.

Proceso de aprendizaje

El proceso de aprendizaje trata sobre la estimación de los parámetros con cualquier proceso de optimización computacional, entre ellos está el proceso del descenso del gradiente estocástico, en el cual, para el cálculo de los diferentes gradientes se utiliza el proceso de retropropagación hacia atrás. En el Capítulo anterior se realizó un bosquejo o idea de como funciona la retropropagación hacia atrás, mientras que el descenso del gradiente estocástico, es una adaptación del descenso del gradiente para problemas de aprendizaje profundo mediante la implementación del método de máxima verosimilitud, es decir, estima los parámetros con una técnica estadística.

El estudio e implementación del gradiente estocástico u otros procesos de optimización y la retropropación hacia atrás están fuera del alcance de este proyecto, sin embargo, se analizará sus diferentes versiones implementadas en Keras y los atributos de esta versiones.

SGD

Este optimizador SGD[30](Descenso del Gradiente Estocástico) es la implemetación básica del gradiente estocástico, donde:

```
tf.Keras.optimizers.SGD(
    learning_rate=0.01, momentum=0.0, nesterov=False
)
```

Listing 2.18: SGD

1. **learning_rate**.- Representa el tamaño del paso en un algoritmo de optimización. El valor predeterminado es 0.01.
2. **momentum**.- Es un valor opcional que acelera el proceso de optimización realizando la sigiente actualización:

   Si momentum=0

   $$w = w - \text{learning_rate} * g$$

   Si momentum >0

   $$\text{velocity} = \text{momentum} * \text{velocity} - \text{learning_rate} * g$$

$$w = w + \text{velocity}$$

con w que representa los paramétros, es decir los pesos de las aristas y g representa el gradiente de la función de pérdida,

3. **nervestov**.- Es una variable booleana, que se activa con el valor de True, en cuyo caso se realiza la siguiente actualización:

$$\text{velocity} = \text{momentum} \ * \text{velocity} - \text{learning_rate} * g$$

$$w = w + \text{momentum} * \text{velocity} - \text{learning_rate} * g$$

RMSprop

Este optimizador implementa el algoritmo RMSprop[27] (Propagación de raíz cuadrática media), que es una mejora del gradiente estócastico, para la estimación de los parámetros de redes neuronales.

```
tf.Keras.optimizers.RMSprop(
    learning_rate=0.001,
    rho=0.9,
    momentum=0.0,
    centered=False,
)
```

Listing 2.19: RMSprop

La esencia de RMSprop es:

Mantener un promedio móvil, es decir, la esperanza del gradiente actual al cuadrado es la combinación convexa de la esperanza del gradiente anterior al cuadrado y el cuadrado del gradiente de la función de minimización.

Además, los pesos se actualizan al restar los pesos anteriores por la división del learning_rate sobre la raíz de la esperanza del gradiente por el cuadrado del gradiente de la función de minimización.

1. **learning_rate**.- Su valor predeterminado es 0.001.

2. **rho**.- Representa el valor escalar de la combinación convexa, su valor predeterminado es 0.9.

3. **momentum**.- Es el mismo parámetro del SGD.

4. **centered**.- Es un valor booleano que se activa con el valor de True e indica si se quiere normalizar los gradientes en cada paso de capa.

Adam

La optimización de Adam[8] es un método de descenso de gradiente estocástico que se basa en la estimación adaptativa de momentos de primer y segundo orden, lo que significa que calcula las tasas de aprendizaje individuales para diferentes parámetros.

La esencial de este algoritmo es el cálculo del momentum actual mediante la combinación convexa del momentum anterior y los gradientes actuales; además, del cálculo de velocity actual con la combinación convexa del velocity anterior y el gradiente al cuadrado. Luego, se calculan dos parámetros un momentum sombrero actual que es el momentum sobre 1 menos el parámetro de su combinación convexa y velocity sombrero actual que es velocity sobre q menos el parámetro de su combinación convexa. Todo ello para obtener una actualización de los pesos mediante el peso anterior restado la tasa de aprendizaje por el momentum sombrero sobre la raíz de velocity sombrero más un psilon.

```
tf.Keras.optimizers.Adam(
    learning_rate=0.001,
    beta_1=0.9,
    beta_2=0.999,
    epsilon=1e-7
)
```

Listing 2.20: Adam

1. **learning_rate**.- Su valor prederteminado es 0.001.
2. **beta_1**.- Valor de la combinación convexa del momentum.
3. **beta_2**.- Valor de la combinación convexa de velocity.
4. **psilon**.- Valor de epsilon para la estimación de la actualización de los pesos en las aristas.

Es de destacar que dentro de Keras existen modificaciónes del algoritmo Adam, que tratan de resolver problemas como la dimensionalidad en la estimación, actualización de los pesos de las aristas por el algoritmo de retropropagación hacia atrás y la implementación de un Adam con impulso.

Dichas modificaciones son: Adadelta, Adagrad, Adamax, Nadam, Ftrl. Para mayor información consulte: [24]

Función de Pérdida

Ahora, para la función de pérdida tenemos que considar que solo necesitamos el estudio de las funciones para clasificación, pues nuestro problema no es de predicción.

Aunque entre las funciones de pérdida para problemas de estimación tenemos en Keras: MeanSquaredError, MeanAbsoluteError, MeanAbsolutePercentageError, MeanSquaredLogarithmicError, CosineSimilarity, Huber, LogCosh. Para mayor información consulte: [19]

Entropía Binaria

La entropía binaria utiliza la función de entropía cruzada[10] presentada en la sección anterior para dos categorías.

```
tf.Keras.losses.BinaryCrossentropy(
   )
```

Listing 2.21: Entropía Binaria

Pérdida de Poisson

La función de pérdida de poisson[25] utiliza la siguiente función:

$$\text{Pérdida} = \sum_{i=1}^{n} \hat{y}_i - y_i * In(\hat{y}_i)$$

```
tf.Keras.losses.Poisson(
)
```

Listing 2.22: Pérdida de Poisson

Divergencia KL

Finalmente, la función de pérdida de divergencia KL[15] utiliza la siguiente función de pérdida:

$$\text{Pérdida} = \sum_{i=1}^{n} y_i * In\left(\frac{y_i}{\hat{y}_i}\right)$$

```
1 tf.Keras.losses.KLDivergence()
```

Listing 2.23: Divergencia KL

Métricas

Cuando tenemos problemas de clasificación binaria nos basamos en la matriz de confunsión para realizar sus respectivas métricas[11]:

		Predicción	
		Positivos	Negativos
Observación	Positivos	Verdaderos Positivos (VP)	Falsos Negativos (FN)
	Negativos	Falsos Positivos (FP)	Verdaderos Negativos (VN)

Figura 2.9: Matriz de confunsión
Elaboración: Autor

En la Figura 2.8 podemos encontrar 4 medidas utilizadas como métricas en Keras, para usar dichas medidas tenemos que realizar su respectiva declaración en inglés (TruePositives, TrueNegatives, FalsePositives, FalseNegatives) y entre comillas.

Además, de estas métricas presentadas en la tabla de confusión, tenemos:

1. **Exactitud(Accuracy)**

$$Accuracy = \frac{VP + VN}{VP + FP + FN + VN},$$

2. **Presición(Precision)**

$$Precision = \frac{VP}{VP + FP},$$

3. **Sensibilidad (Sensitivity)**

$$Sensitivity = \frac{VP}{VP + FN},$$

4. **Especificidad (Especificity)**

$$Especifity = \frac{VN}{VN + FP}.$$

Asimismo, existe la medida **AUC** que es el área bajo la curva ROC. La curva ROC nos indica que tan bueno es el modelo para distinguir entre las dos clases de la variable dependiente. Dicha curva tiene como variable el corte de la clasificación, es decir el número que utilizamos para designar si nuestra predicción es 0 o 1.

Este corte puede tomar cualquier número real entre 0 y 1; con ello, utilizando este corte como parámetro, lo que se gráfica en el eje de las abcisas es 1−especificidad(corte) y en el eje de las coordenadas la sensibildad(corte). Por tanto, mientras mayor sea el AUC mejor es el modelo distinguiendo las dos clases.

Para los problemas de predicción los nombres de las métricas son exactamente los mismos que se especificaron en el subtema de función de pérdida. Con todo lo presentado en este subcapítulo utilizamos la siguiente combinación de función de perdida, optimizador y métrica, para poder ajustar de manera apropiada nuestro problema de clasificación:

```
model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate =0.002,
    beta_1 =0.9, beta_2 =0.9 ,epsilon =1e-7),
    loss=[tf.keras.losses.BinaryCrossentropy()],
    metrics=['accuracy'],
)
```

Listing 2.24: Compilación del modelo de transacciónes fraudulentas

2.5.2. Ajuste del Modelo

Por último, realizamos el ajuste del modelo[37] que es el proceso en el cual se estiman los parámetros, dentro de Keras para este ajuste encontramos los siguientes atributos:

```
Model.fit(
    x=None,
    y=None,
    batch_size=None,
    epochs=1,
    validation_split=0.0,
    class_weight=None,
```

```
    sample_weight=None
)
```

Listing 2.25: Ajuste del modelo

1. **x**.- Son los datos de entrada. Puede ser: Una matriz Numpy, Un tensor de TensorFlow, o un data set.
2. **y** : datos de de la variable dependiente. Al igual que los datos de entrada x, pueden ser matrices Numpy o tensores de TensorFlow. Debe ser coherente con x (no puede tener entradas Numpy y objetivos de tensor, o viceversa).
3. **batch_size** .- Entero o None. Número de muestras por actualización de gradiente. Si no se especifica batch_size, el valor predeterminado será 32.
4. **epochs** .- Entero. Número de épocas para entrenar el modelo. Una época es una iteración sobre la totalidad datos proporcionados .
5. **validation_split** .- Un númer real entre 0 y 1. Dicha fracción de los datos de entrenamiento se utilizará como datos de validación. El modelo separará esta fracción de los datos de entrenamiento, no se entrenará en ella y evaluará la pérdida y cualquier métrica del modelo en estos datos al final de cada epochs. Los datos de validación se seleccionan de las últimas muestras en los datos proporcionados, antes de realizarse un muestreo.
6. **validation_data** : datos sobre los que evaluar la pérdida y cualquier métrica del modelo al final de cada época. El modelo no se entrenará con estos datos.
7. **shuffle**.- Valor booleano que indica si mezclar los datos de entrenamiento antes de cada época o no.
8. **class_weight** .- Diccionario opcional que asigna índices de clase (enteros) a un valor de peso (flotante), que se utiliza para ponderar la función de pérdida (solo durante el entrenamiento). Esto puede ser útil para decirle al modelo que "preste más atención.ª las muestras de una clase subrepresentada.
9. **sample_weight** : Matriz Numpy opcional de pesos para las muestras de entrenamiento, que se utiliza para ponderar la función de pérdida (solo durante el entrenamiento). Puede pasar una matriz plana de Numpy con la misma longitud que las muestras de entrada , o en el caso de datos temporales, puede pasar una matriz 2D con forma (samples, sequence_length), para aplicar una diferente peso para cada paso de tiempo de cada muestra.

Con la información proporcionada, como primer paso realizamos la ponderación de los pesos (class_weight). Para ello, utilizaremos class_weight.compute_class_weight de la librería Sklearn para la estimación de los pesos. Este atributo está inspirada en la regresión logística en datos de eventos raros[41]. Por tanto para cada elemento de la serie, haremos la estimación de 0 y 1, recordando que la entrada a class_weight es un diccionario de listas.

```
from sklearn.utils import class_weight
class_weights = []
for i in range(len(y_train[1,0,:])):
    class_weights.append(
    class_weight.compute_class_weight(
    class_weight = 'balanced',
    classes = np.unique(y_train[:,0,i]),
    y=np.array(y_train[:,0,i])))
```

Listing 2.26: Ponderación para class_weight

Transformando esta lista a un diccionario cuyas claves son el número de nodos de la capa de salida, empezando desde 0, procedemos ha hallar el tamaño de datos de entrenamiento y evaluación; por ende, aplicamos la siguiente técnica de Boostrap[38], que utiliza la siguiente fórmula:

$$N \geq \frac{8}{\epsilon^2} ln \left(\frac{4m_h(2N)}{\delta} \right) \tag{2.3}$$

- N = tamaño de muestra.
- ϵ = Representa el error.
- δ = Representa la confianza.
- $m_h(2N)$ = Representa la formas de etiquetar el doble del número de los datos.

Para nuestro caso N=5856, $m_h(2N)$= 98, ϵ y δ son parámetros a estimar; la forma de etiquetar nuestros datos es 49 pues las demás combinaciones no tienen importancia, ya que queremos saber si el titular tiene transacciones fraudulentas, mas no en que posición. Con ello obtenemos los siguientes resultados con los parámetros a estimar:

ε	δ	FÓRMULA (2.2)	N-FÓRMULA(2.2)	$\frac{\text{FÓRMULA } (2.2)}{\text{N}}$	$\frac{1 - \text{FÓRMULA } (2.2)}{\text{N}}$	TOTAL
0,15	0,8	1709,550588	3948,449412	30%	70%	100%
0,2	0,8	961,622206	4696,377794	17%	83%	100%
0,25	0,8	615,4382118	5042,561788	11%	89%	100%
0,15	0,85	1687,995167	3970,004833	30%	70%	100%
0,2	0,85	949,4972816	4708,502718	17%	83%	100%
0,25	0,85	607,6782602	5050,32174	11%	89%	100%
0,15	0,9	1667,672176	3990,327824	29%	71%	100%
0,2	0,9	938,0655989	4719,934401	17%	83%	100%
0,25	0,9	600,3619833	5057,638017	11%	89%	100%
0,15	0,95	1648,448275	4009,551725	29%	71%	100%
0,2	0,95	927,2521546	4730,747845	16%	84%	100%
0,25	0,95	593,441379	5064,558621	10%	90%	100%
0,15	0,99	1633,784112	4024,215888	29%	71%	100%
0,2	0,99	919,0035629	4738,996437	16%	84%	100%
0,25	0,99	588,1622803	5069,83772	10%	90%	100%

Figura 2.10: Estimación del tamaño de muestra
Elaboración: Autor

Por tanto, de la inecuación anterior requerimos una cota superior para N, por lo cual, escogemos con un error de 15 % , una confianza al 80 % y un tamaño de muestra de 30 %. Con ello, conseguimos el siguiente ajuste:

```
model.fit(
    {"monto": x_train1, "receptor": x_train2},
    {"clase": y_train},
    epochs = 200,
    batch_size = 32,
    validation_split = 0.30,
    class_weight = final
)
```

Listing 2.27: Ajuste del Modelo

En este ajuste realizamos el paso de datos al modelo tanto de las variables independientes y dependiente del modelo, asimimso el porcentaje de validación y cuantas veces se va a ajustar nuestro modelo.

Capítulo 3

Resultados, conclusiones y recomendaciones

3.1. Resultados

Para los resultados presentamos como uno de nuestros resultados principales el modelo neuronal, que contiene tanto la descripción de cada capa como también su número de párametros:

```
Model: "model_2"
____________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
====================================================================================================
monto (InputLayer)              [(None, 1, 49)]      0
____________________________________________________________________________________________________
receptor (InputLayer)           [(None, 1, 49)]      0
____________________________________________________________________________________________________
masking_4 (Masking)             (None, 1, 49)        0           monto[0][0]
____________________________________________________________________________________________________
masking_5 (Masking)             (None, 1, 49)        0           receptor[0][0]
____________________________________________________________________________________________________
bidirectional_4 (Bidirectional) (None, 1, 98)        38808       masking_4[0][0]
____________________________________________________________________________________________________
bidirectional_5 (Bidirectional) (None, 1, 98)        38808       masking_5[0][0]
____________________________________________________________________________________________________
average_2 (Average)             (None, 1, 98)        0           bidirectional_4[0][0]
                                                                 bidirectional_5[0][0]
____________________________________________________________________________________________________
time_distributed (TimeDistribut (None, 1, 49)        4851        average_2[0][0]
____________________________________________________________________________________________________
clase (Dense)                   (None, 1, 49)        2450        time_distributed[0][0]
====================================================================================================
Total params: 84,917
Trainable params: 84,917
Non-trainable params: 0
____________________________________________________________________________________________________
```

Figura 3.1: Modelo Neuronal
Elaboración: Autor

Podemos notar que las capas de entrada tienen las mismas dimensiones de [(None, 1, 49)], mientras que después de la capa de bidireccionales este número cambia por [(None, 1, 98)], ya que se realiza una LSTM tanto de la secuencia de 1 a 49, como la de la secuencia del 49 al 1. Luego de aquello se realiza una media como capa de agrupación para poder obtener al final una capa densa que clasifica en 0 o 1, considerando nuestras nos variables independientes del modelo.

Además de ello, presentamos la medida con que evaluamos el modelo y la pérdida total después de la última epoch, tanto para su muestra de entrenamiento como para su muestra de testeo:

```
Epoch 200/200
40/40 [==============================] - 0s 12ms/step - loss: 0.0209 - accuracy: 0.6263 - val_loss: 0
.0606 - val_accuracy: 0.8270
<keras.callbacks.History at 0x207a8e5c550>
```

Figura 3.2: Exactitud del modelo
Elaboración: Autor

Por tanto, notamos que el ajuste de los datos tiene un ajuste del 62,63 %, que para la tarea planeada y por el desbalanceo de los datos es apropiado para utilizar el modelo; de este modo presentamos algunos resultados utilizando el atributo predict() y un corte de 0.2, pues este corte es proporcional al numero de observaciones de la clase 1 sobre el total de todas las observaciones, así procedemos a presentar diferentes situaciones que se pueden presentar en las entidades financieras para realizar pronósticos con nuestro modelo.

En un primer caso presentamos a un titular de tarjeta de crédito que pertenece a los datos con el cual se entrenaron el modelo, específicamente el titular 2171, pero en este caso el titular 2171 realizó 5 transacciones nuevas, es decir se le agregan 5 nuevas transacciones, lo que provoca que se elimine las dos primeras transacciones de su historial, pues al inicio el titular 2171 tenía como variables independientes, sucesiones de 46 observaciones y para pronosticar sus nuevas 5 transacciones necesitamos como máximo sucesiones de longitud 49,por ende tenemos que:

```
Detección de una transacción legítima (Verdaderos Positivos):  44
Transacción legítima incorrectamente detectatada (Falso Negativo):  0
Transacción fraudulenta incorrectamente detectada (Falso Positivo):  0
Transacción fraudulenta detectada:  5
Total de transacciones del titular:  5
```

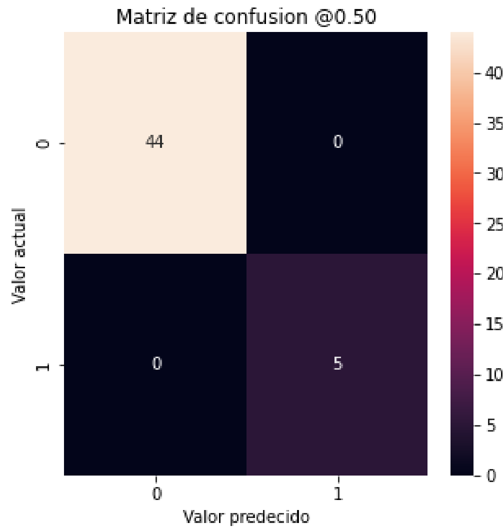

Figura 3.3: Resultado 1
Elaboración: Autor

Podemos notar que en la Figura [3.3] la clasificación es exacta, ya que las transacciones legítimas son 44, mientras que las transacciones fraudulentas detectadas fueron 5, del Total de transacciones fraudulentas, dicho total de transacciones fraudulentas se lo representa con el Total de transacciones del titular. Por lo cual, los falsos negativos y falsos positivos nos da un resultado de 0, lo cual es ideal para nuestra detección.

El segundo caso que podemos encontrar es para nuevos titulares que empezaron a realizar transacciones con nuestra entidad financiera. Por ejemplo para el titular de tarjetas de crédito con identificación 5000, notamos que dicho titular no se utilizó ni para el entrenamiento del modelo, ni para su evaluación, con lo cual podemos observar el siguiente resultado:

```
Detección de una transacción legítima (Verdaderos Positivos):  40
Transacción legítima incorrectamente detectatada (Falso Negativo):  0
Transacción fraudulenta incorrectamente detectada (Falso Positivo):  1
Transacción fraudulenta detectada:  8
Total de transacciones del titular:  9
```

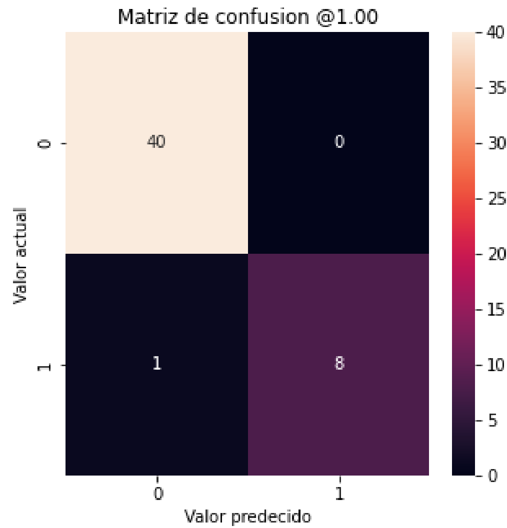

Figura 3.4: Resultado 2
Elaboración: Autor

En esta clasificación podemos notar que dicho titular tuvo 49 transacciones de las cuales 9 fueron fraudulentas y 40 transacciones fueron legítimas, de ellas tenemos que existe una transacción fraudulenta que no pudo detectar nuestro modelo. Es decir, existe un falso negativo de las 9 transacciones que no fueron legítimas, lo que representa un 11,11 % de la clasificación de transacciones que son fraudulentas.

Finalmente al ser un titular de tarjeta de crédito nuevo podemos analizar su Curva Roc para analizar el rendimiento de nuestro modelo para este tipo de clientes.

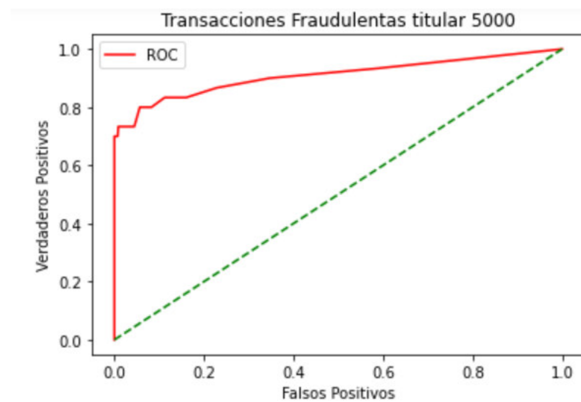

Figura 3.5: Resultado 3
Elaboración: Autor

En dicha grafica su AUC= 0.7675, lo cual nos da la posibilidad de colegir que nuestro modelo clasifica a nuevos clientes de manera correcta para determinar si existe o no un a transacción fraudulenta.

3.2. Conclusiones

Los modelos matématicos de aprendizaje profundo han revolucionado el mundo financiero, dando márgenes que han logrado el crecimiento de la industria privada; con ello, estudiar estos modelos incluyendo sus relaciones de dependencia como lo hacen las Redes Neuronales Recurrentes, logran un instrumento capas de evaluar problemas mucho más complejos que aquellos en los que se asume independencia en las variables. Además, el estudio teórico de estos instrumentos matemáticos proporciona una perspectiva diferente a la habitual de los modelos de aprendizaje profundo, en otras palabras, la ideal de que los modelos de aprendizaje profundo son una caja negra queda obsoleta y proporciona mayor robustes a nuestros resultados.

Con ello, otro aspecto importante es la comprensión del problema para poder aplicar estos modelos de aprendizaje profundo de manera eficiente; en este caso el problema de implementar un modelo para la detección de transacciones fraudulentas con tarjeta de crédito resulta ser esencial por la continua innovación de las formas de pago y el progreso de la tecnología en este mundo globalizado. Como conclusión, además de la implementación de un modelo eficiente; las compañías deben gestionar los problemas que se generan con los consumidores de las tarjetas al momento de adquirir este servicio. Este problema se produce cuando las empresas que venden este producto (el crédito) no informan de manera correcta a sus consumidores de como mantener la seguridad en sus tarjetas. Además, este problema se produce cuando las empresas no mantienen un control sobre sus sistemas de seguridad o por la falta de comunicación entre las áreas encargas de otorgar el crédito a los consumidores y las personas que se encargan de la seguridad de dichas tarjetas.

Finalmente, otro aspecto fundamental es conocer las herramientas computacionales, como los metódos de minería de datos para ajustar de manera correcta nuestros modelos. Por ello, conocer Keras y las funciones que hay dentro de esta librería, nos da la oportunidad de crear estos modelos de manera intuitiva; la creación de capas, las conecciones entre capas, la compilación y ajuste del modelo son funciones que tienen atributos fáciles de interpretar e implementar, de forma que el proble-

ma de la complejitud computacional y la creación de objetos son un inconveniente menos para los desarrolladores de inteligencias artificiales.

3.3. Recomendaciones

Dentro del mundo de las redes neuronales existe una gran aplicabilidad de métodos estádisticos, de optimización y computacionales que ayudan al desarrollo de la investigación dentro del campo de la matématica aplicada; por tal motivo se recomienda estudiar de manera detallada los métodos del gradiente estócastico, tanto su implementación como su base estadístico y de optimización, el algoritmo de retropropagación hacia atrás que implica el estudio de programación dinámica y por último, la creación de redes neuronales propias como clases de la librería Keras, lo que supone el estudio a profundidad de la programación orientada a objetos y estructuras de datos.

Asimismo, se recomienda el estudio de otras configuraciones arquitectónicas, como también otros métodos de minería de datos para contrastar estos nuevos resultados con los presentados en este trabajo. Estos métodos pueden ser desde el preprocesamiento de datos hasta la utilización de nuevas herramientas para la evaluación de los resultados, haciendo énfasis en el entendimiento del problema a resolver como en este caso el problema de las transacciones fraudulentas con tarjetas de crédito.

Finalmente, el problema de las transacciones fraudulentas conlleva grandes pérdidas a las compañías que tienen como producto financiero al crédito; estos problemas desembocan en la creación de áreas que se dediquen solo al seguimiento del proceso de crédito, lo cual puede ser tedioso y consuma muchos recursos a las empresas. Por tal motivo, la creación de estos modelos de clasificación deben estar acompañados de otros modelos matématicos complementarios que contribuyan a una detección temprana de estas transacciones fraudulentas; como también de programas computacionales y bases de datos que agiliten estos procesos.

Capítulo 4

Anexos

4.1. Apéndice A

```
import pickle as pkl
import pandas as pd
import tensorflow as tf #pythorch
import numpy as np
with open("2018-04-01.pkl", "rb") as f:
    object = pkl.load(f)

df1 = pd.DataFrame(object)
df1.to_csv(r'file.csv')

with open("2018-04-02.pkl", "rb") as f:
    object = pkl.load(f)

df2 = pd.DataFrame(object)
df2.to_csv(r'file.csv')

with open("2018-04-03.pkl", "rb") as f:
    object = pkl.load(f)

df3 = pd.DataFrame(object)
df3.to_csv(r'file.csv')

with open("2018-04-04.pkl", "rb") as f:
    object = pkl.load(f)

df4 = pd.DataFrame(object)
df4.to_csv(r'file.csv')
```

```
28
29 with open("2018-04-05.pkl", "rb") as f:
30     object = pkl.load(f)
31
32 df5 = pd.DataFrame(object)
33 df5.to_csv(r'file.csv')
34
35 with open("2018-04-06.pkl", "rb") as f:
36     object = pkl.load(f)
37
38 df6 = pd.DataFrame(object)
39 df6.to_csv(r'file.csv')
40
41 with open("2018-04-07.pkl", "rb") as f:
42     object = pkl.load(f)
43
44 df7 = pd.DataFrame(object)
45 df7.to_csv(r'file.csv')
46
47 with open("2018-04-08.pkl", "rb") as f:
48     object = pkl.load(f)
49
50 df8 = pd.DataFrame(object)
51 df8.to_csv(r'file.csv')
52
53 with open("2018-04-09.pkl", "rb") as f:
54     object = pkl.load(f)
55
56 df9 = pd.DataFrame(object)
57 df9.to_csv(r'file.csv')
58
59 with open("2018-04-10.pkl", "rb") as f:
60     object = pkl.load(f)
61
62 df10 = pd.DataFrame(object)
63 df10.to_csv(r'file.csv')
64
65 with open("2018-04-11.pkl", "rb") as f:
66     object = pkl.load(f)
67
68 df11 = pd.DataFrame(object)
69 df11.to_csv(r'file.csv')
70
71 with open("2018-04-12.pkl", "rb") as f:
```

```
72      object = pkl.load(f)
73
74  df12 = pd.DataFrame(object)
75  df12.to_csv(r'file.csv')
76
77  with open("2018-04-13.pkl", "rb") as f:
78      object = pkl.load(f)
79
80  df13 = pd.DataFrame(object)
81  df13.to_csv(r'file.csv')
82
83  with open("2018-04-14.pkl", "rb") as f:
84      object = pkl.load(f)
85
86  df14 = pd.DataFrame(object)
87  df14.to_csv(r'file.csv')
88
89  with open("2018-04-15.pkl", "rb") as f:
90      object = pkl.load(f)
91
92  df15 = pd.DataFrame(object)
93  df15.to_csv(r'file.csv')
94
95  with open("2018-04-16.pkl", "rb") as f:
96      object = pkl.load(f)
97
98  df16 = pd.DataFrame(object)
99  df16.to_csv(r'file.csv')
100
101 with open("2018-04-17.pkl", "rb") as f:
102     object = pkl.load(f)
103
104 df17 = pd.DataFrame(object)
105 df17.to_csv(r'file.csv')
106
107 with open("2018-04-18.pkl", "rb") as f:
108     object = pkl.load(f)
109
110 df18 = pd.DataFrame(object)
111 df18.to_csv(r'file.csv')
112 with open("2018-04-18.pkl", "rb") as f:
113     object = pkl.load(f)
114
115 df19 = pd.DataFrame(object)
```

```
116 df19.to_csv(r'file.csv')
117 with open("2018-04-19.pkl", "rb") as f:
118     object = pkl.load(f)
119 
120 df20 = pd.DataFrame(object)
121 df20.to_csv(r'file.csv')
122 with open("2018-04-20.pkl", "rb") as f:
123     object = pkl.load(f)
124 
125 df21 = pd.DataFrame(object)
126 df21.to_csv(r'file.csv')
127 with open("2018-04-21.pkl", "rb") as f:
128     object = pkl.load(f)
129 
130 df22 = pd.DataFrame(object)
131 df22.to_csv(r'file.csv')
132 with open("2018-04-22.pkl", "rb") as f:
133     object = pkl.load(f)
134 
135 df23 = pd.DataFrame(object)
136 df23.to_csv(r'file.csv')
137 with open("2018-04-23.pkl", "rb") as f:
138     object = pkl.load(f)
139 
140 df24 = pd.DataFrame(object)
141 df24.to_csv(r'file.csv')
142 with open("2018-04-24.pkl", "rb") as f:
143     object = pkl.load(f)
144 
145 df25 = pd.DataFrame(object)
146 df25.to_csv(r'file.csv')
147 with open("2018-04-25.pkl", "rb") as f:
148     object = pkl.load(f)
149 
150 df26 = pd.DataFrame(object)
151 df26.to_csv(r'file.csv')
152 with open("2018-04-26.pkl", "rb") as f:
153     object = pkl.load(f)
154 
155 df27 = pd.DataFrame(object)
156 df27.to_csv(r'file.csv')
157 with open("2018-04-27.pkl", "rb") as f:
158     object = pkl.load(f)
159 
```

```
df28 = pd.DataFrame(object)
df28.to_csv(r'file.csv')
with open("2018-04-28.pkl", "rb") as f:
    object = pkl.load(f)

df29 = pd.DataFrame(object)
df29.to_csv(r'file.csv')
with open("2018-04-29.pkl", "rb") as f:
    object = pkl.load(f)

df30 = pd.DataFrame(object)
df30.to_csv(r'file.csv')
with open("2018-04-30.pkl", "rb") as f:
    object = pkl.load(f)
X = pd.concat([df1,df2,df3,df4,df5,df6,df7,df8,df9,df10,df11,df12,df13
    ,df14,df15,df16,df17,df18,df19,df20,df21,df22,df23,df24,df25,df26,
    df27,df28,df29,df30])
T = pd.Series(X.groupby('CUSTOMER_ID')['TX_AMOUNT'].mean())
X=X[['CUSTOMER_ID','TERMINAL_ID','TX_AMOUNT','TX_FRAUD']]
X['TERMINAL_ID']=(X['TERMINAL_ID']-X['TERMINAL_ID'].min())/(X['
    TERMINAL_ID'].max()-X['TERMINAL_ID'].min())
X['TX_AMOUNT']=(X['TX_AMOUNT']-X['TX_AMOUNT'].min())/(X['TX_AMOUNT'].
    max()-X['TX_AMOUNT'].min())

a = []
b = []
c = []
for i in T.index:
    a.extend([X[X['CUSTOMER_ID'] == i].to_numpy()[:,1]])
    b.extend([X[X['CUSTOMER_ID'] == i].to_numpy()[:,2]])
    c.extend([X[X['CUSTOMER_ID'] == i].to_numpy()[:,3]])
e = []
f = []
g = []
for i in range(len(a)):
    if len(a[i])>40 and len(a[i])<50:
        e.append(a[i])
        f.append(b[i])
        g.append(c[i])
    elif len(a[i])>= 50:
        e.append(a[i][0:49])
        f.append(b[i][0:49])
        g.append(c[i][0:49])
a1=[]
```

```
200 a2=[]
201 for i in range(len(g)):
202     if sum(g[i])==0:
203         a1.append(i)
204     else:
205         a2.append(i)
206 
207 print(len(g))
208 print(len(a1))
209 print(len(a2))
210 e1 = []
211 f1 = []
212 g1 = []
213 j=1
214 while j<8:
215     for i in range(len(a2)):
216         e1.append(e[a2[i]])
217         f1.append(f[a2[i]])
218         g1.append(g[a2[i]])
219     j +=1
220 for i in range(len(e1)):
221     e.append(e1[i])
222     f.append(f1[i])
223     g.append(g1[i])
224 import matplotlib.pyplot as plt
225 #plot each series
226 plt.plot(f[30],label="[31]")
227 plt.plot(f[567],label="[568]")
228 plt.plot(f[2003],label="[2004]")
229 plt.plot(f[1023],label="[1024]")
230 plt.plot(f[3006],label="[3007]")
231 plt.legend(title='Montos')
232 plt.show()
233 padded_inputs1 = tf.keras.preprocessing.sequence.pad_sequences(
234     e, padding="post", maxlen=None, dtype='int32', value=-1.0)
235 padded_inputs2 = tf.keras.preprocessing.sequence.pad_sequences(
236     f, padding="post", maxlen=None, dtype='float32', value=-1.0)
237 padded_outputs = tf.keras.preprocessing.sequence.pad_sequences(
238     g, padding="post", maxlen=None, dtype='int32', value=1.0)
239 x_train1 = padded_inputs1.reshape(-1,1,49)
240 x_train2  = padded_inputs2.reshape(-1,1,49)
241 y_train  = padded_outputs.reshape(-1,1,49)
242 from tensorflow.keras.models import Model
243 from tensorflow.keras.layers import Masking,Dense,Bidirectional,Input,
```

```
    LSTM,Embedding,SimpleRNN
244 monto_input = tf.keras.Input(shape=(1,49,), name="monto")
245
246 receptor_input = tf.keras.Input(shape=(1,49,), name="receptor")
247
248 monto_capa = tf.keras.layers.Masking(mask_value=-1.0)(monto_input)
249
250 receptor_capa = tf.keras.layers.Masking(mask_value=-1.0)(
    receptor_input)
251
252 monto_capa = tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(49,
    return_sequences =True, input_shape = (1,49,)))(monto_capa)
253
254 receptor_capa = tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(49,
    return_sequences =True, input_shape = (1,49,)))(receptor_capa)
255
256 x = tf.keras.layers.average([monto_capa, receptor_capa])
257
258 #x = tf.keras.layers.Dense(49)(x)
259
260 #x= tf.keras.layers.LSTM(49,return_sequences=True)(x)
261
262 x = tf.keras.layers.TimeDistributed(tf.keras.layers.Dense(49))(x)
263
264 clase_pred = tf.keras.layers.Dense(49, activation='sigmoid' ,name="
    clase")(x)
265
266 model = tf.keras.Model(
267     inputs=[monto_input,receptor_input],
268     outputs=[clase_pred],
269 )
270 model.compile(
271     optimizer=tf.keras.optimizers.Adam(),
272     loss=[tf.keras.losses.BinaryCrossentropy()],
273     metrics=['accuracy'],
274 )
275 model.fit(
276     {"monto": x_train1, "receptor": x_train2},
277     {"clase": y_train},
278     epochs = 1000,
279     batch_size = 32,
280     validation_split = 0.3,
281     class_weight = final
282 )
```

```
283 import matplotlib as mpl
284 import matplotlib.pyplot as plt
285 import sklearn
286 from sklearn.metrics import confusion_matrix
287 from sklearn.model_selection import train_test_split
288 from sklearn.preprocessing import StandardScaler
289 import seaborn as sns
290 def plot_cm(labels, predictions, p=9.9997638e-01):
291   cm = confusion_matrix(labels, predictions > p)
292   plt.figure(figsize=(5,5))
293   sns.heatmap(cm, annot=True, fmt="d")
294   plt.title('Matriz de confusion @{:.2f}'.format(p))
295   plt.ylabel('Valor actual')
296   plt.xlabel('Valor predecido')
297 
298   print('Detecci n de una transacci n leg tima (Verdaderos
     Positivos): ', cm[0][0])
299   print('Transacci n leg tima incorrectamente detectatada (Falso
     Negativo): ', cm[0][1])
300   print('Transacci n fraudulenta incorrectamente detectada (Falso
     Positivo): ', cm[1][0])
301   print('Transacci n fraudulenta detectada: ', cm[1][1])
302   print('Total de transacciones del titular: ', np.sum(cm[1]))
303   import numpy as np
304 import pandas as pd
305 import matplotlib.pyplot as plt
306 import seaborn as sns
307 from sklearn.datasets import make_classification
308 from sklearn.ensemble import RandomForestClassifier
309 from sklearn.model_selection import train_test_split
310 from sklearn.metrics import roc_curve
311 
312 def plot_roc_curve(fper, tper):
313     plt.plot(fper, tper, color='red', label='ROC')
314     plt.plot([0, 1], [0, 1], color='green', linestyle='--')
315     plt.xlabel('Falsos Positivos')
316     plt.ylabel('Verdaderos Positivos')
317     plt.title('Transacciones Fraudulentas titular 5000')
318     plt.legend()
319     plt.show()
320 plot_roc_curve(fper, tper)
```

Listing 4.1: Código Completo

Bibliografía

[1] Aggarwal, C. C. *Neural Networks and Deep Learning*. International Business Machines. Springer, Cambridge University Press,Estados Unidos, 2015.

[2] A.Karpathy, J. Johnson, L. F.-F. *Visualizing and understanding recurrent networks.* Department of Computer Science, Stanford University, 2016.

[3] C. Phua, V. L. A comprehensive survey of data mining-based fraud detection research. págs. 1–10, 2010.

[4] Coates, A. *Neural networks: Tricks of the Trade, Second Edition*. Springer, Technische Universitat Berlin, Departament of Computer Science, 2012.

[5] Hochreiter, S. *Gradient flow in recurrent nets:the difficulty of learning long-term dependencies*. Dept. Informatique et Recherche Opérationnelle Universit´e de Montr´eal, CP 6128, Succ. Centre-Ville, Montréal, Québec, Canada, 2001.

[6] K. Paliwal, M. S. . K. K. Bidirectional recurrent neural networks.ieee transactions on signal processing, Vol. 45, No. 11, November. 1997.

[7] Kaufmann, M. *The Morgan Kaufmann Series in Data Management Systems, third edition*. University of Illinois at Urbana–Champaign, 2011.

[8] Keras. Adam. `https://keras.io/api/optimizers/adam/`, 2016.

[9] Keras. Bidirectional layer. `https://keras.io/api/layers/recurrent_layers/bidirectional/`, 2016.

[10] Keras. Binary cross entropy. `https://keras.io/api/losses/probabilistic_losses/#binarycrossentropy-class`, 2016.

[11] Keras. Classification metrics based on true/false positives negatives. `https://keras.io/api/metrics/classification_metrics/`, 2016.

[12] Keras. Concatenate layer. https://keras.io/api/layers/merging_layers/concatenate/, 2016.

[13] Keras. Convolutional layer. https://keras.io/api/layers/convolution_layers/, 2016.

[14] Keras. Dense layer. https://keras.io/api/layers/core_layers/dense/, 2016.

[15] Keras. Divergence kl. https://keras.io/api/losses/probabilistic_losses/#kldivergence-class, 2016.

[16] Keras. Embedding layer. https://keras.io/api/layers/core_layers/embedding/, 2016.

[17] Keras. Input layer. https://keras.io/api/layers/core_layers/input/, 2016.

[18] Keras. Keras layers api. https://keras.io/api/layers/, 2016.

[19] Keras. Loss. https://keras.io/api/losses/, 2016.

[20] Keras. Lstm layer. https://keras.io/api/layers/recurrent_layers/lstm/, 2016.

[21] Keras. Masking layer. https://keras.io/api/layers/core_layers/masking/, 2016.

[22] Keras. Merge layer. https://keras.io/api/layers/merging_layers/, 2016.

[23] Keras. The model class. https://keras.io/api/models/model/, 2016.

[24] Keras. Optimizers. https://keras.io/api/optimizers/, 2016.

[25] Keras. Poisson. https://keras.io/api/losses/probabilistic_losses/#poisson-class, 2016.

[26] Keras. Pooling layer. https://keras.io/api/layers/pooling_layers/, 2016.

[27] Keras. Rms. https://keras.io/api/optimizers/rmsprop/, 2016.

[28] Keras. Rnn layer. https://keras.io/api/layers/recurrent_layers/simple_rnn/, 2016.

[29] Keras. The sequential class. https://keras.io/api/models/sequential/, 2016.

[30] Keras. Sgd. https://keras.io/api/optimizers/sgd/, 2016.

[31] Keras. Timedistributed layer. https://keras.io/api/layers/recurrent_layers/time_distributed/, 2016.

[32] Programador Clic. Una comprensión fácil de entender de tensor, rango y forma. https://programmerclick.com/article/4529888432/, 2016.

[33] Report, T. N. Trade Publication on Consumer Payment Systems. *Issue 1193*, págs. 15–31, 2021.

[34] Solé-Casals, J. *Machine Learning Methods with Noisy, Incomplete or Small Datasets*. Department of Electrical and Electronic Engineering, Tokyo University of Agriculture and Technology,Tokyo 184-8588, 2018.

[35] Sundermeyer, M. Translation modeling with bidirectional recurrent neural networks. *Human Language Technology and Pattern Recognition Group, Spoken Language Processing Group*, 2014.

[36] Tensorflow. Preprocessing sequence pad sequences. https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/sequence/pad_sequences, 2016.

[37] Tensorflow. tf.keras.model. https://keras.io/api/metrics/classification_metrics/, 2016.

[38] Tobar, A. *Nueva metodología para la detección de anomalías usando técnicas de minería de datos y Bootstrap. Caso de aplicación eficiencia energética* . Ecuador, 2021.

[39] Yannael. Fraud detection handbook. https://github.com/Fraud-Detection-Handbook/simulated-data-raw/tree/main/data, 2017.

[40] Y.Bengio. *Justifying and generalizing contrastive divergence. Neural Computation*. Dept. IRO, University of Montreal, August 5th, 2008.

[41] Zen, K. *Logistic Regression in Rare Events Data*. Department of Political Science, George Washington University,Funger Hall, 2201 G Street NW, Washington, DC 20052, 2001.