Universidad Zaragoza
1542

Facultad de Ciencias
Universidad Zaragoza

Trabajo de Fin de Máster de Física y Tecnologías Físicas

Detección de anomalías en datos secuenciales mediante técnicas de *machine learning*

Departamento de Ingeniería Electrónica y Comunicaciones

Autor:
Miguel Lahoz Muñoz

Director:
Nicolás Medrano Marqués

Septiembre 2020

Agradecimientos

Quiero dar las gracias a mi familia, los cuales me han apoyado y ayudado durante toda mi carrera para poder seguir adelante, y me han servido de apoyo durante la realización de este trabajo. También quiero agradecer a mis profesores, ya que sin su ayuda no podría haber terminado la carrera y no estaría ahora terminando mis estudios de máster. Por último, pero no menos importante, quiero agradecer a la empresa Etiqmedia, los cuales me han dejado sus equipos para la realización de este trabajo.

Resumen

La detección de anomalías es un problema que surge desde distintos campos. Desde evitar el fraude en transacciones bancarias, pasando por la detección de ataques en línea, hasta la detección de nuevas partículas en los grandes aceleradores de partículas, los algoritmos de detección trabajan en un amplio espectro de casos, cada uno con sus características peculiares. Sin embargo, la gran mayoría de estos sistemas son estáticos: funcionan para las condiciones concretas para las que fueron diseñados, pero pocos son capaces de adaptar la condición de "normalidad" conforme se obtienen nuevos datos. En este trabajo se pretenden estudiar las alternativas de algoritmos de detección de anomalías, con el objetivo de construir un sistema que permita, sin entrenamiento o con apenas entrenamiento, detectar anomalías en un sistema nuevo, e ir adaptándose a los cambios que vayan surgiendo.

Figuras y tablas

Acrónimos

- FPGA: *Field-Programmable Gate Array*
- GAN: *Generative Adversary Network*
- FFN: *Feed Forward Network*
- ADAM: *ADAptative Moment estimation*
- LSTM: *Long Short-Term Memory*
- HTM: *Hierarchical Temporal Memory*
- SDM: *Sparse Distributed Memory*
- MNIST: *Modified National Institute of Standards and Technology database*
- RAM: *Random Access Memory*
- CPU: *Central Processing Unit*
- GCP: *Graphics Processing Unit*

Índice

I. Motivación y objetivos

La motivación para la realización de este trabajo surge de la necesidad de expandir los conocimientos obtenidos durante la realización del Trabajo de Fin de Grado en *Machine Learning,* estudiando un modelo no convencional de red. Además, este trabajo se realiza desde una premisa más física, siendo posible utilizar la detección de anomalías en apartados más técnicos, como la detección de partículas en los grandes aceleradores, o medidas inusuales en diversos experimentos.

Los datos con los que se va a evaluar esta herramienta son datos públicos al acceso de todo el mundo. Los veremos con más detalle a lo largo del trabajo, pero se trata del número de personas que están utilizando el servicio de taki de Nueva York en cada momento, y el precio de la publicidad en una web. Estas entradas habrá que tratarlas de forma correcta, esto lo veremos en la sección *IV.A Datos y Codificación.*

La mayoría de algoritmos actuales están basados en una serie de reglas implementadas por el programador o por el usuario de la aplicación, que sólo pueden modificarse por configuración. Por ejemplo, se pueden tener varias secuencias de sucesos considerados normales, y cualquiera que no encaje en esa lista, ser una anomalía. El problema de estos algoritmos es la necesidad de hacer los cambios de forma manual. En nuestro caso, buscamos una forma de adecuar la situación de “normalidad” y de “anomalía” sin decirle al sistema cuales son las nuevas reglas.

Este trabajo se centra en la búsqueda, implementación, y comprobación de funcionamiento de algoritmos de detección de anomalías que sean capaces de autoadaptarse a las condiciones de los datos de cada momento. A su vez, este algoritmo usará *Machine Learning* por un principal motivo: en los últimos años esta tecnología ha avanzado a pasos agigantados, y se ha propuesto como una de las tecnologías más eficientes a la hora de aprender patrones, por lo que su utilización es perfecta para nuestro objetivo.

Este trabajo, además de usarse con finalidades académicas, también cuenta con un interes comercial, ya que se ha realizado con ayuda de la empresa Etiqmedia, para la cual me hayo trabajando en estos momentos. Esta empresa necesita un sistema igual que el planteado en este trabajo: un detector de anomalías que sea capaz de adecuar su situación de normalidad con los nuevos datos que le vayan dejando. A cambio, la empresa ha ayudado en la realización de este trabajo, ya que ha dejado a nuestra disposición diversos equipos con altas prestaciones para poder estudiar mejor los diversos algoritmos que iremos estudiando a lo largo del trabajo. El código final se ha visto modificado por esto, ya que necesitamos tener un producto comercial, es decir, tiene que ser versátil y facilmente modificable por programadores que no hayan realizado un estudio exhaustivo en las anomalías.

II. Anomalías

Para poder detectar anomalías, primero debemos definir qué consideramos como anomalía. Podemos definir las anomalías como aquellos patrones que se salen de la normalidad en datos un conjunto dado de datos. Estos patrones pueden ser de diversos tipos, dependiendo de su naturaleza.

Por ejemplo, podríamos pensar en un experimento en el que estamos caracterizando un sistema de medida electrónico dentro de un entorno controlado, caracterizando el valor del voltaje a la salida con respecto a la temperatura. Al terminar las medidas, representamos los datos obtenidos, y observamos lo siguiente (Figura 1):

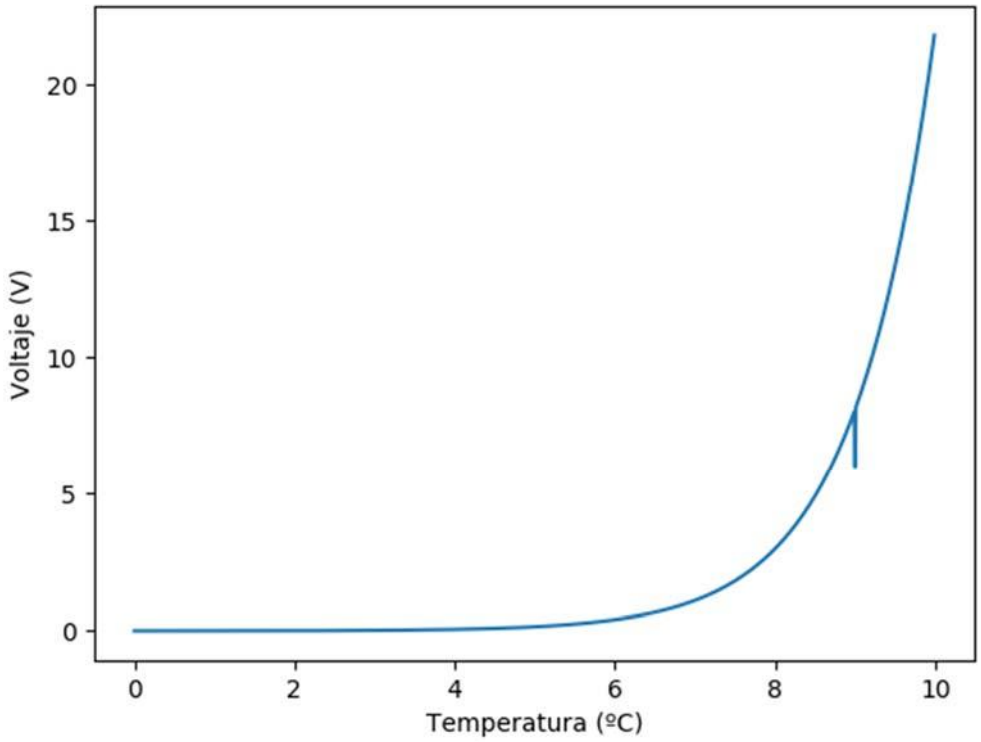

Figura 1: Ejemplo anomalía puntual. Origen: Propio

A simple vista, podemos ver que algo ha pasado en torno a los 9ºC.En este caso, se trata de una anomalía puntual, posiblemente debida a un problema del instrumental de medida.

Otro ejemplo podría ser en el campo de la informática. Suponemos una web en la que usuarios pueden registrarse, conectarse, comprobar sus mensajes y desconectarse. Si estudiamos a un usuario en concreto vemos que sus registros son (Figura 2):

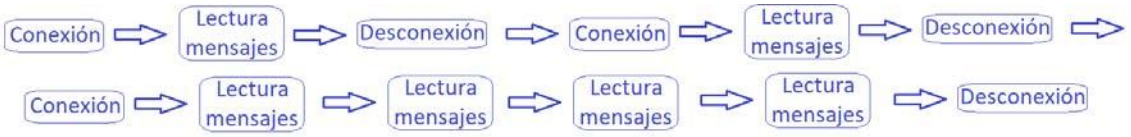

Figura 2: Ejemplo anomalía secuencial. Origen: Propio

En este caso, la anomalía no se trata de un solo dato puntual, ya que la lectura de mensajes se ha producido con anterioridad. Lo anómalo es, en una misma conexión, realizar tres lecturas consecutivas, y podría deberse a un acceso por parte de una persona no autorizada que intenta recabar información de la víctima. En este caso la anomalía es un conjunto de eventos, y no un evento en concreto.

A continuación, vamos a hacer una distinción de los distintos tipos de anomalías existentes, los cuales vamos a sacar de un meta estudio realizado por la Universidad de Minnesota, el cual agrupa un gran compendio de estudios sobre las anomalías, y los agrupa dependiendo del origen de estas [1].

II.A. Distintos tipos de anomalías

Podemos agrupar las anomalías dependiendo de varios aspectos: la naturaleza de los datos de entrada, el tipo de anomalía, la clasificación de los datos y la salida de la detección de anomalías. Es importante conocer los distintos tipos de anomalías y el cómo etiquetarlas para poder elegir el algoritmo que proporcione los mejor resultados.

Nuestros datos de entrada pueden ser de muy diversos tipos. Podemos tener datos “puros”, como los que podemos ver en el primer ejemplo, o podemos agrupar datos bajo una etiqueta. En el segundo ejemplo, los datos originales (una petición a una web, o varias) están agrupados y etiquetados, ya que en este caso las anomalías que estamos buscando trabajan con estas etiquetas. Otro ejemplo podría ser una entrada de vídeo: en vez de trabajar con el vídeo completo, podemos ejecutar un algoritmo de reconocimiento de objetos, y luego analizar anomalías dependiendo de los objetos presentes en la imagen.

Además, la relación que hay entre los distintos datos de entrada también es importante. Mientras que en el primer ejemplo no necesitamos saber qué dato va antes o después del que nos importa (la medida incluso podría haberse realizado en varios intentos), en el segundo la anomalía sólo se encuentra por el orden en el que se encuentran los datos. A este tipo de patrones se las denomina secuenciales, aunque no son los únicos en los que puede darse relación entre los datos. Por ejemplo, podríamos tener una entrada de datos con una relación espacial, en la que el orden en el que llegan los datos no es importante, pero sí saber la posición relativa que ocupan entre sí.

Dependiendo de la naturaleza de los datos que se procesan, las posibles anomalías pueden clasificarse en puntuales, contextuales y colectivas. Las anomalías puntuales son aquellas en las que el valor de un solo dato puede ser una anomalía en sí mismo; las contextuales y las colectivas requieren analizar del conjunto de datos para decidir si son realmente una anomalía o no. La principal diferencia entre las anomalías contextuales y colectivas es el tamaño del conjunto de datos que son necesarios para determinar si hay una anomalía en un patrón: las anomalías contextuales dependen de una ventana concreta de datos para la decisión, mientras que las anomalías colectivas dependen de todo el histórico de datos. Como ejemplo, la Figura 3 muestra datos de temperatura representados a lo largo de varios meses. Aquí, t_1 y t_2 tienen el mismo valor, pero el contexto de los datos que les rodean hace que t_2 sea considerado una anomalía. Si no fuese por el resto de datos, podría no ser una anomalía, ya que no sabríamos la forma que tiene la curva de temperaturas en los meses adyacentes, o incluso en años anteriores.

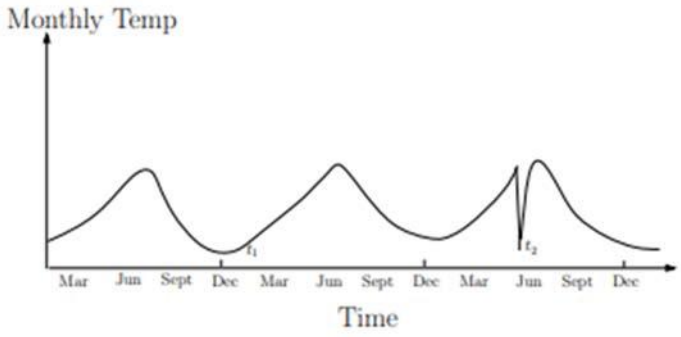

Figura 3: Anomalía contextual. Origen: [1]

Una vez sabemos qué tipo de anomalías queremos detectar, necesitamos obtener un conjunto de datos de entrenamiento y test para poder definir, implementar y verificar la solución adecuada. Dependiendo de cómo se obtenga la clasificación estos datos, podemos hablar de etiquetado no supervisado, semi-supervisado y supervisado.

- El etiquetado no supervisado supone que nuestro algoritmo no necesita una base de datos inicial para ajustar su comportamiento y decidir si está detectando las anomalías correctamente, por lo que no disponemos de un conjunto de datos y el programa aprenderá a detectar anomalías conforme vaya recibiendo datos y construyendo con ellos lo que es "normal".

- Por el contrario, podríamos necesitar un conjunto de datos para mostrar a nuestro algoritmo qué es anómalo. Si este conjunto de datos está catalogado con etiquetas, teniendo datos "normales" y datos "anómalos", hablamos de etiquetado supervisado.

- Finalmente, si sólo disponemos de datos etiquetados como "normales" hablamos de una técnica semi-supervisada.

La catalogación de anomalías puede hacerse de diversas formas. Los dos métodos más comunes son las etiquetas y la puntuación. En el caso de etiquetas, el algoritmo cataloga cada dato de entrada en una de las dos posibles opciones: "normal" o "anómalo". En el caso de catalogación por puntuación, el sistema proporcionará una salida numérica, con un valor mayor cuando la probabilidad de ese dato ser anómalo es mayor. Este segundo caso se podría convertir en un sistema de etiquetado simplemente declarando un umbral o valor límite (*threshold*), a partir del cual catalogamos un dato como anómalo. También, si disponemos de un conjunto de datos etiquetado de cualquier forma de las anteriores, podemos usarlo para el entrenamiento de un sistema por puntuación, simplemente asignado a cada etiqueta una puntuación (0 para caso normal, 100 para anómalo en el caso más sencillo). Este sistema tiene más flexibilidad que el etiquetado tradicional, ya que nos permite hacer etiquetas escalonadas: podemos tener datos normales, de baja anomalía, de media anomalía y de alta anomalía, dependiendo del valor de salida.

II.B. Técnicas y algoritmos actuales

Una vez que hemos visto los distintos tipos de anomalías, podemos hacer un estudio de los algoritmos más usados actualmente en la detección de anomalías, para luego poder decidir cuál es mejor para nuestra solución.

Una de las primeras aproximaciones que se intentan para detectar anomalías consiste en obtener un modelo estadístico que nos permita calcular la probabilidad de, dado un conjunto de datos anteriores y el dato a evaluar, suceso no fuese anómalo, es decir, intenta predecir el siguiente dato dado la secuencia anterior.

Una de las primeras aproximaciones que se intentan para detectar anomalías consiste en obtener un modelo estadístico que nos permita calcular la probabilidad, dado un conjunto de datos anteriores y el dato a evaluar, de que un suceso no sea anómalo. Es decir, que prediga el siguiente dato de la secuencia. Estos algoritmos pueden ser usados tanto para detección supervisada [2] como no supervisada [3]. El sistema no supervisado intenta crear un modelo que permita ajustar la definición de "normalidad" si los datos varían con el tiempo. Un ejemplo muy sencillo de un modelo así sería usar la media de las anteriores medidas, y comprobar la desviación del nuevo dato con respecto a los anteriores. De esta forma, un dato que se "aleje" demasiado de los valores anteriores sería considerado una anomalía, mientras que, si los datos se modifican de una forma suave, se llega a una situación en la que la normalidad ha cambiado, haciendo que nuevos valores que con anterioridad serían anómalos ahora no lo sean.

En el caso presentado en [2], el valor que utilizan para establecer la normalidad de los patrones es la correlación entre el valor más actual y los valores anteriores hasta una distancia τ, haciendo que esta sea la ventana de datos a tomar, y la función usada para calcular la probabilidad de que sea una anomalía es:

$$\rho(\tau) = Corr(X_t, X_{t+\tau}) = \frac{E[(X_t - E[X_t])(X_{t+\tau} - E[X_{t+\tau}])]}{\sqrt{Var[X_t]VAR[X_{t+\tau}]}} \quad (1)$$

En este ejemplo en concreto, el tamaño de la ventana varía dependiendo de t, ya que los datos con los que trabajan son periódicos, e intentan que dentro de la ventana haya un ciclo completo, por lo que hay momentos en los que el error asociado a esta medida es mayor, ya que la ventana va creciendo con el tiempo hasta llegar a un nuevo ciclo.

Otros estudios intentan usar otros valores estadísticos para calcular la probabilidad de anomalía. En el caso de [4], se hace un estudio de qué valores son más útiles, y se analiza cómo el empleo de la media o la mediana puede llevar a falsos positivos, y por lo tanto no se deberían usar de forma única para estos cálculos.

Otro tipo de técnica es la llamada *Kernel Nearest Neighbors*, también conocida como *K-Nearest Neighbors*, basada en el empleo de una función distancia (no tiene por qué ser una distancia euclídea) para obtener los K vecinos más cercanos al nuevo dato. De esta forma, se pueden hacer grupos de datos considerados normales, y todo dato que esté fuera de estos grupos se puede marcar como anómalo (Figura 4).

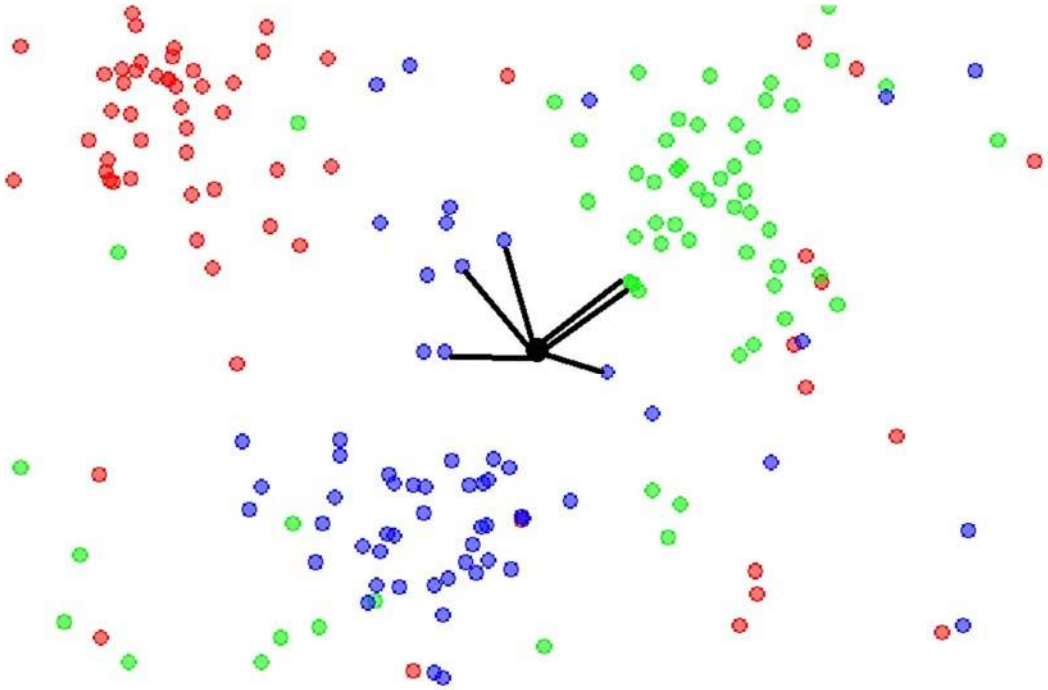

Figura 4: K-Nearest Neighbors. Origen: unite.ai

Estos algoritmos llevan aplicándose desde 1991 [5], siendo actualmente usados junto con algoritmos de *Machine Learning* en un amplio rango de casos. Por poner un ejemplo, son muy utilizados en tecnologías de reconocimiento facial, en el que una red neuronal convierte una imagen de una cara en una representación 128-dimensional, y se utiliza un algoritmo de *K-Nearest* para obtener a qué persona pertenece esa cara.

Estos algoritmos tienen una principal desventaja, y es su coste computacional, ya que cuantos más datos deben procesarse más complicado se hace el cálculo. Hay diversos estudios que intentan mitigar este efecto, por ejemplo, dividiendo el espacio en pequeños subespacios para no comprobar todos los posibles vecinos, o implementando estos algoritmos en FPGAs para que su velocidad sea mayor [6].

Otros algoritmos más actuales hacen uso de la tecnología del *Machine Learning* para poder detectar anomalías. Este es el caso de [7], donde se propone un sistema basado en una red GAN (*Generative Adversarial Network*). Estos sistemas constan de dos redes neuronales antagónicas, encargándose una del trabajo a realizar, y la segunda se usa para comprobar los resultados que la primera genera. En el caso de la detección de anomalías, se tiene una red que recibe los datos de entrada, y decide si estos son anómalos o no. Una segunda red recibe estos mismos datos más la predicción y decide si es correcta. Si estas dos redes se dejan entrenar, pueden llegar a aprender de forma no supervisada el comportamiento correcto.

Otro método de aplicar *Machine Learning* en la detección de anomalías en datos es parecido al usado en los algoritmos de reconocimiento facial explicados con anterioridad. En este caso, una red recibe unos datos de entrada, y los transforma, proyectándolos en un segundo espacio (de la misma dimensión o menor), donde se aplican algoritmos como los vistos con anterioridad (por ejemplo, un *K-Neighbors*).

III. *Machine Learning*

Como hemos visto en la sección anterior, últimamente se están utilizando diversas técnicas de *Machine Learning* para resolver los problemas de anomalías. En este apartado vamos a hacer una pequeña introducción a qué es el *Machine Learning*, explicando las redes más sencillas, y veremos un tipo de red que está dando muy buenos resultados en la detección de anomalías.

El *Machine Learning* o Aprendizaje Automático es un campo de la informática centrado en la obtención de resultados mediante el aprendizaje, en lugar de buscar la solución de forma analítica. Estos algoritmos están basados en los estudios de las neuronas reales, modelizadas de una forma sencilla para poder implementarlas en un ordenador.

Una primera aproximación a estos algoritmos la encontramos en [8], trabajo en el que se nos muestran distintas aproximaciones, dese las más realistas hasta las más ideales, de cómo funciona una neurona.

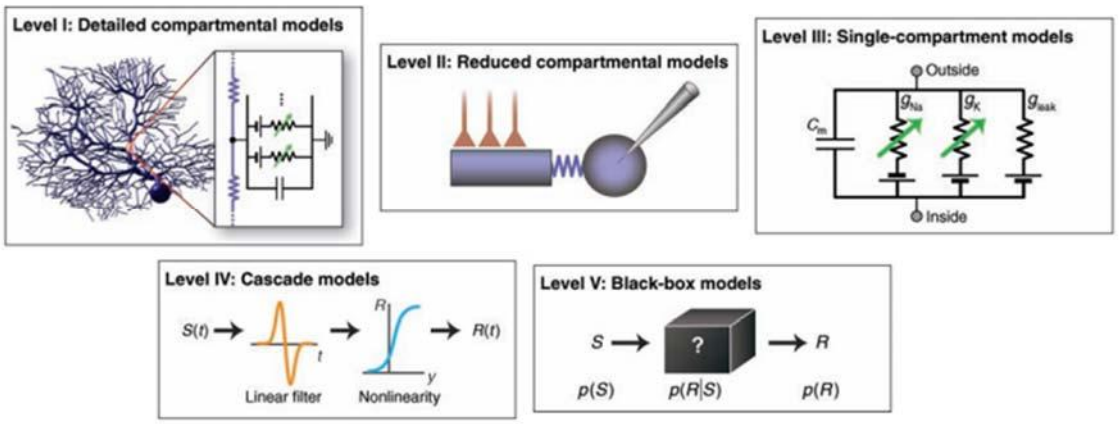

Figura 5: Modelos de neuronas. Fuente: [8]

Estos modelos son (Figura 5):

- Nivel 1: Cada parte de la neurona es representada por un equivalente eléctrico, convirtiendo las distintas partes en resistencias dependientes de variables biológicas, generadores de potencial, etc. Estos modelos dan muy buenos resultados si se pretende estudiar el comportamiento de una o pocas neuronas, ya que una sola neurona puede necesitar hasta 1000 componentes distintos para poder modelizarse, por lo que es necesario un gran poder de cómputo para poder simular su comportamiento.
- Nivel 2: En estos modelos se intentan agrupar las diferentes partes de la neurona en pequeños compartimentos, permitiendo así la simulación de un mayor número de neuronas. Sin embargo, la simplificación de estos modelos resulta en un mayor error al aumentar el número de neuronas, dejando de dar resultados correctos alrededor de las 3000 neuronas.
- Nivel 3: La siguiente aproximación es tratar a la neurona como un solo componente, centrándonos en su comportamiento general. En este caso se muestra el modelo clásico de Hodking-Huxley [9], que modela los distintos potenciales que aparecen entre el interior y el exterior de una neurona como distintas resistencias cuyo valor depende de la concentración de los diferentes iones que crean estas diferencias de potencial (Sodio y Potasio), junto con un pequeño voltaje de fuga. Estos modelos permiten estudiar los picos de potencial que da una neurona ante distintos estímulos de entrada.

- Nivel 4: En esta aproximación perdemos los componentes biológicos que tenía la anterior, convirtiéndose la neurona en un simple filtro que aplica diferentes funciones a una entrada. Estas funciones suelen ser un filtro lineal, seguido de una función no lineal para no perder la naturaleza de las neuronas. Estos modelos neuronales son los más usados en el *Machine Learning*, ya que dan buenos resultados, además de permitir el uso de algoritmos muy eficientes para su cálculo. Lo estudiaremos con más detalle en la siguiente sección.
- Nivel 5: Por último, tenemos una aproximación puramente probabilística, en la que una neurona es una caja negra, la cual almacena las distintas salidas que puede dar ante una entrada y sus distintas probabilidades. En este caso, la salida se elige aleatoriamente de entre todas las posibles, con los pesos ajustados para cumplir con las probabilidades guardadas.

Con estos modelos, ya podemos estudiar los algoritmos de *Machine Learning* más usados.

III.A. Redes neuronales convencionales

Como hemos visto en el anterior apartado, las redes neuronales convencionales usadas en el *Machine Learning* están basadas en modelos de filtros. Así, podemos definir una neurona dentro de estas redes como una operación matemática, siendo la neurona más sencilla una suma pesada de todas sus entradas, a la que posteriormente se le aplica un filtro no lineal (Figura 6).

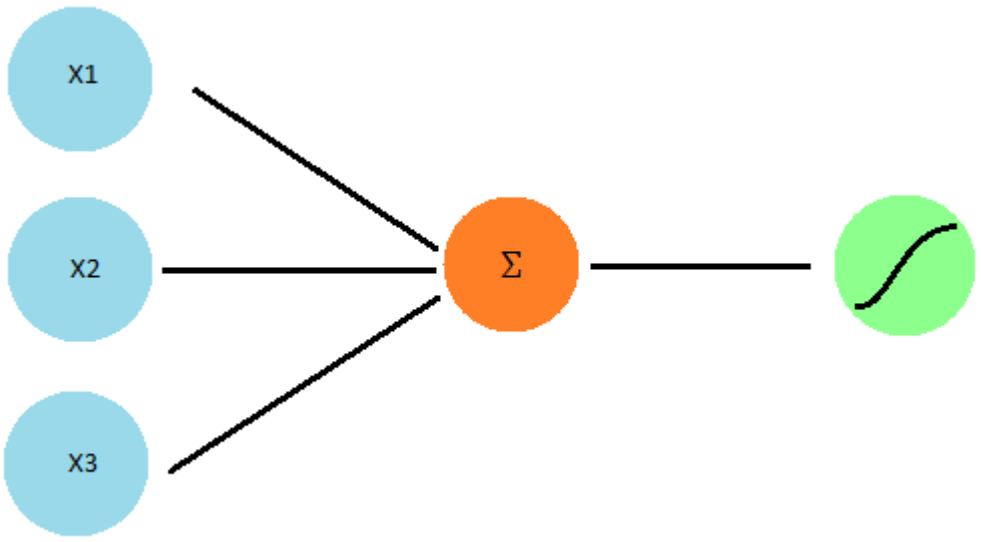

Figura 6: Modelo neurona sencilla. Fuente: Propia

En este caso, tenemos una neurona con tres entradas (x_1, x_2, x_3). Cada una de estas entradas se multiplica por un peso θ_i, individual de cada entrada. Adicionalmente, algunos modelos usan una entrada extra, llamada de *Bias*, cuyo valor es fijo e igual a 1, siendo lo único que varía su peso.

Tras multiplicarse sus entradas por sus respectivos pesos, se suman todos los resultados, y se hacen pasar por una función no lineal. Esta función suele ser una función sigmoide

$$f(x) = \frac{1}{1+e^{-x}} \tag{2}$$

Cuya salida se encuentra acotada entre 0 y 1. De esta forma, la salida de una neurona viene dada por:

$$y = f(\sum_n x_n \theta_n) = \frac{1}{1+e^{-\sum_n x_n \theta_n}} \tag{3}$$

Para modificar cómo se comporta una neurona, basta con modificar su vector de pesos $(\theta_0, \dots, \theta_n)$.

Una red neuronal está formada por un número de neuronas organizadas en capas. La red neuronal más sencilla que se puede formar es una *Feed Forward Network*, en la que su característica más importante es que las salidas de todas las neuronas de una capa están conectadas a las entradas de los procesadores de la siguiente capa. De esta forma, cada neurona de una capa tiene tantas entradas como neuronas haya en la capa anterior (más una, asociada al *Bias*). Las redes *Feed Forward* tienen una capa de entrada, una capa de salida, y una o varias capas intermedias u ocultas. La capa de entrada debe tener tantas neuronas como el número de componentes de los vectores que constituyen los datos que se proporcionan a la red. La capa de salida debe tener tantas neuronas como componentes de salida queramos de la red, ya que ésta puede ser un número (una componente, una neurona) o un vector. El número de capas intermedias, así como de procesadores por capa deberá ajustarse atendiendo a la complejidad de problema que se pretende resolver.

La principal ventaja de estos modelos es que podemos representar las distintas capas como matrices, en las que cada columna (o fila) son los pesos de una neurona para sus entradas. Así, estas matrices tendrán de dimensión el número de entradas de cada neurona por el número de neuronas de esa capa. De esta forma, podemos organizar los pesos de la red en diferentes matrices, y aplicando únicamente operaciones matriciales, calcular el resultado. Esto es muy importante, ya que actualmente existen librerías muy potentes de cálculo tensorial, muy bien optimizadas para trabajar con el hardware actual, lo que permite realizar miles de iteraciones en una red en apenas unos segundos.

Una vez que podemos calcular la salida de una red, necesitamos saber cómo modificar los pesos para que esta "aprenda" las respuestas adecuadas. Para esto se utilizan técnicas de minimización de errores, en las que se calculan las diferencias entre el resultado obtenido y el resultado deseado, y se aplican correcciones en los pesos que más han influido en ese error. Uno de los algoritmos más usados es el algoritmo ADAM [10], ya que obtiene muy buenos resultados en un tiempo relativamente pequeño.

Este tipo de redes relativamente sencillas son las más usadas actualmente en la detección de anomalías puntuales. Sin embargo, no resultan adecuadas en problemas que presenten anomalías secuenciales (contextuales y colectivas), ya que no son capaces de relacionar una iteración de los datos con las anteriores. Es decir, carecen de "memoria". Una posible solución sería utilizar neuronas LSTM, o *Long Short Term Memory* [11] (Figura 7).

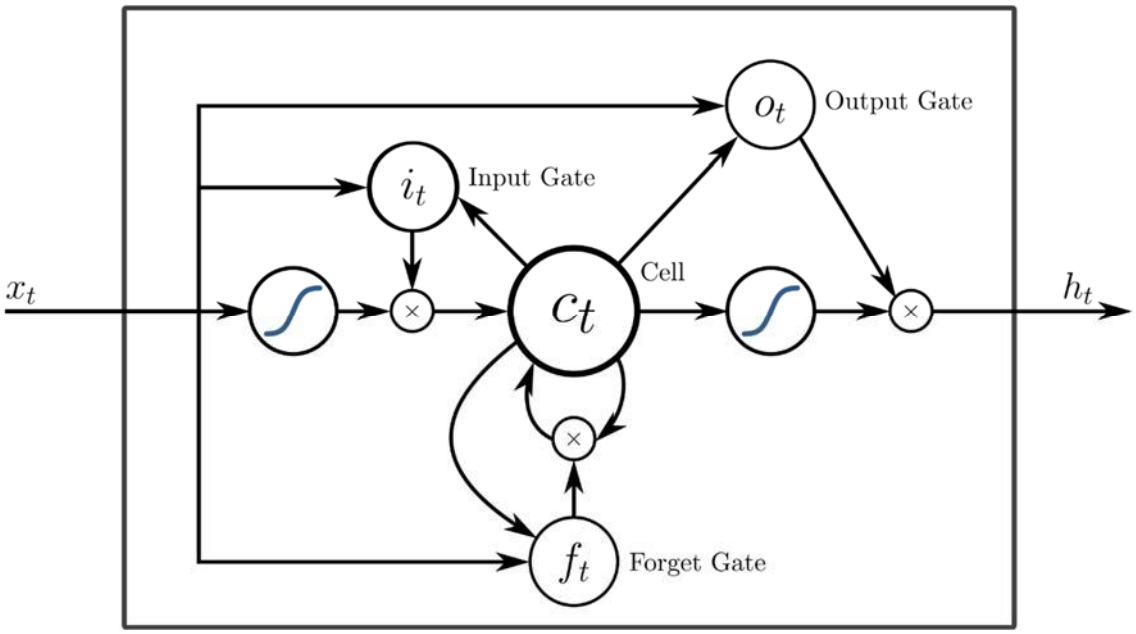

Figura 7: Neurona LSTM. Fuente: Wikipedia

Las neuronas LSTM constan de unas pequeñas "puertas" que almacenan el valor anterior de la neurona, y el resultado actual se ve modificado por este valor. De esta forma, se obtiene una relación entre la salida actual y los valores devueltos con anterioridad. El principal problema de estos tipos de redes es la velocidad de aprendizaje, ya que el hecho de guardar un valor anterior hace que los algoritmos utilizados sean mucho más lentos, ya que el cálculo del gradiente de la función de error se hace muy costoso, por lo que cada vez este tipo de redes se usan menos, mientras se buscan alternativas con redes que, aunque sean más grandes en tamaño, permitan un entrenamiento más corto.

III.B. Hierarchical Temporal Memory

Otra posible solución es, en vez de modelar una sola neurona y construir una red a partir de ella, modelar el comportamiento de la red neuronal completa. Esto es lo que se pretende conseguir con la "Memoria Jerárquica Temporal", o HTM por sus siglas en inglés [12].

La teoría HTM intenta centrarse en el comportamiento del neocórtex de los mamíferos, fijándose en cómo funcionan las conexiones de las diferentes neuronas en lugar del comportamiento de una neurona puntual. De esta forma, se define un modelo de celda que intenta modelar una neurona junto con cómo se conecta al resto. La Figura 8 muestra un esquema de funcionamiento de este modelo, donde los datos del *Feedforward*, que son los datos de salida de la celda anterior se representan en color verde. Los datos representados en azul provienen de dendritas lejanas, las cuales se activan si su entrada supera un cierto umbral. Estos datos representan el contexto, ya que se "conectan" a distintas neuronas, por lo que almacenan datos de qué neuronas estaban activas o inactivas en los anteriores pasos.

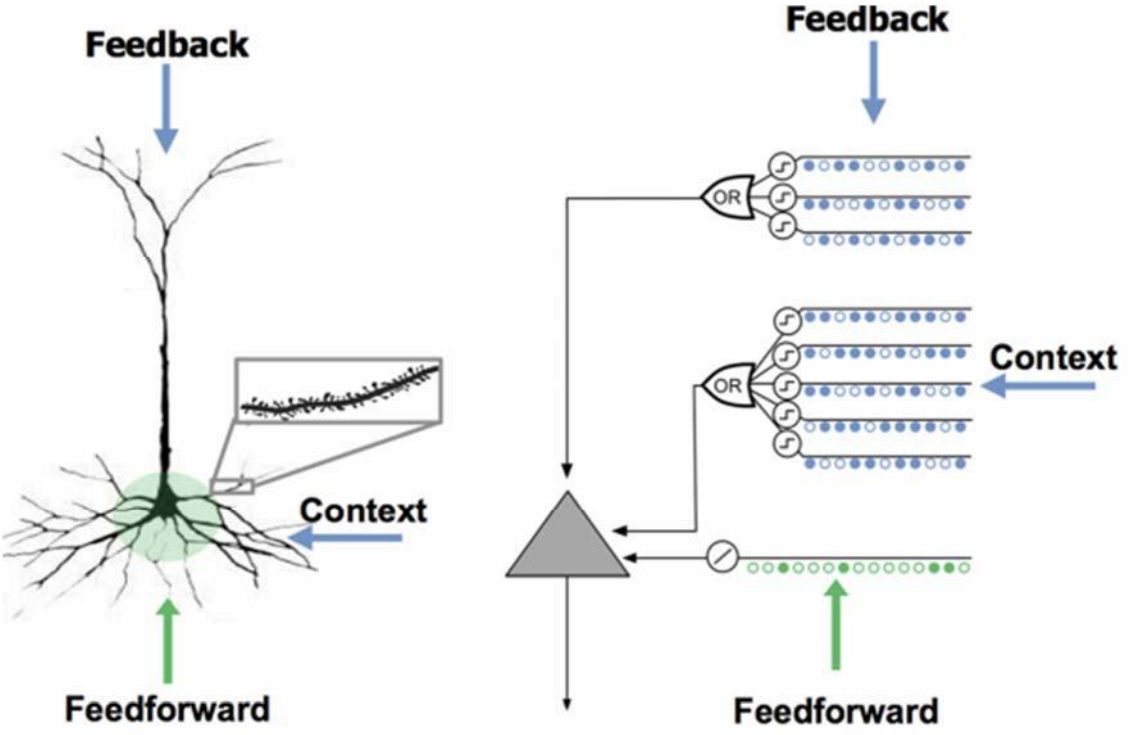

Figura 8: Comparación neurona y celda HTM. Fuente: [12]

Al tener un carácter tan temporal (las entradas dependen de los anteriores pasos), estos modelos son muy buenos en el reconocimiento de patrones en datos temporales, y por eso se están empezando a usar en el reconocimiento de anomalías. Además, tienen un conjunto de características que les diferencian de las redes tradicionales:

- Aprenden de forma continua, ya que los pesos, que en este caso son los umbrales de activación de las distintas neuronas, se ven modificados a cada paso temporal del algoritmo.
- Predicciones de alto orden. El "alto orden" se utiliza en referencia a las cadenas de alto orden de Markov [13], modelos matemáticos que intentar modelar secuencias de datos de gran tamaño. Es decir, permiten la predicción de anomalías utilizando datos en un gran abanico temporal.
- Permiten el aprendizaje de reglas locales, es decir, se forman grupos de celdas centradas en la predicción de una sola regla, mientras que en las redes convencionales todas las neuronas de la red funcionan de manera indistinta, ya que no hay una relación topológica entre ellas.

Este modelo, sin embargo, precisa que los datos sean representados de forma binaria, lo que supone una complicación. Esto puede no parecer un problema, ya que en computación todos los datos se representan así. Sin embargo, son necesarias ciertas reglas con respecto a cómo se almacenan estos datos.

III.C. Sparse Distributed Memory

Para explicar por qué no podemos utilizar una representación binaria normal para representar los datos para un modelo HTM, pondremos un ejemplo concreto. Imaginemos que queremos

representar un dato de entrada numérico, con números enteros para, por ejemplo, representar los días del mes.

El cerebro humano relaciona días que están uno al lado del otro, de tal forma que el día 5 de un mes es muy parecido (próximo) al día 6, un poco parecido al día 7, y muy poco parecido al día 15. Si utilizamos una representación binaria de 8 bits para representar los días 5, 6 y 13, obtenemos la representación mostrada en la Figura 9:

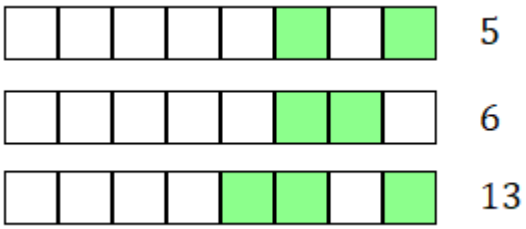

Figura 9: Representación binaria. Fuente: Propia

Aquí podemos ver cómo, tanto el número 5 como el número 6 activarían 2 entradas cada uno, de las cuales sólo una es compartida, mientras que el número 13 activaría las dos mismas neuronas que el número 5, más una tercera, por lo cual el modelo entendería que el número 5 se parece más al número 13 que al número 6.

Una solución sería representar cada número como un único bit, de tal forma que tendríamos 31 (por los distintos días del mes) bits, estando activo solamente el bit del día que corresponda (Figura 10).

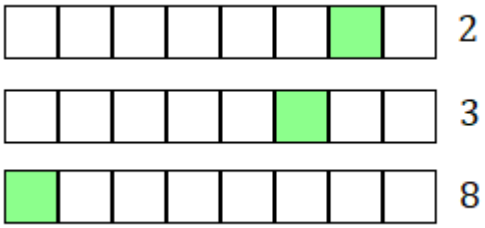

Figura 10: Posible solución. Fuente: Propia

Sin embargo, esta solución tiene un problema parecido: aunque se están activando entradas muy alejadas entre sí, no se están activando entradas iguales entre números que son muy parecidos. Esto se arregla si hacemos que cada número sean varios bits, de tal forma que tendríamos, en un caso de 3 bits por número, la representación mostrada en la Figura 11.

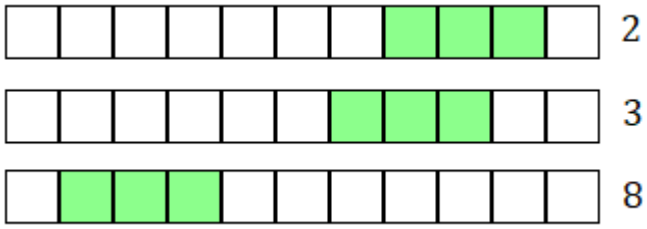

Figura 11: Solución. Fuente: Propia

En esta solución, se puede observar como el número 2 y el 3 son muy parecidos, ya que casi activan las mismas entradas, mientras que el 8 no activa ninguna. A este tipo de memoria se le

denomina *Sparse Distributed Memory* [14], y es la utilizada en las redes HTM, aunque también se ha utilizado con buenos resultados en redes tradicionales [15].

Una característica útil de este tipo de codificación es que es muy sencillo representar valores que de otra forma podría ser complicado. Siguiendo el ejemplo de codificar fechas, codificar el día de la semana puede ser complicado a la hora de entrenar una red neuronal, ya que si categorizamos a cada día con una etiqueta por su nombre ("Lunes", "Martes"…) la red no puede aprender las relaciones entre estas etiquetas. Sin embargo, usando SDM, podemos codificar los días en forma de anillo (Figura 12).

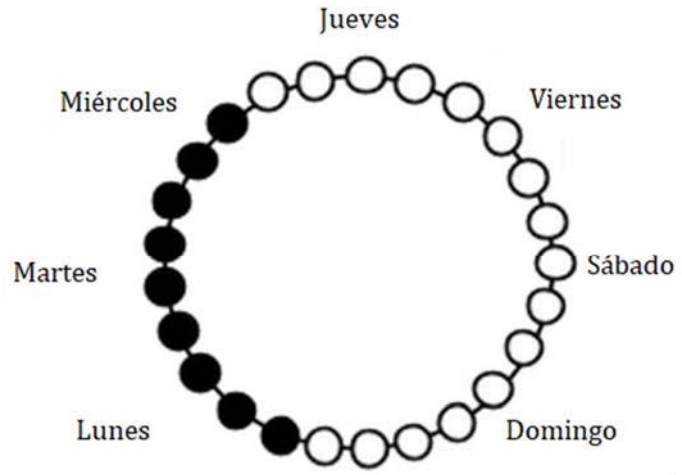

Figura 12: Rueda codificación días. Fuente: Propia

Al igual que antes, esta codificación permite que el día "Martes" comparta entradas con el día "Lunes" y "Miércoles", pero también permite que los días "Lunes" y "Domingo" compartan, ya que, realmente, están uno al lado del otro.

Otra característica importante de este tipo de codificación es su robustez ante el ruido. Esta capacidad se puede estudiar matemáticamente teniendo en cuenta alguna de las características de estas codificaciones. En primer lugar, es importante la dimensionalidad, es decir, el número de bits total del que disponemos para representar los datos, al igual que el número de bits activos que hay en cada vector de datos. En el ejemplo anterior, tenemos unos datos con una dimensionalidad de 25 bits, con vectores de 9 bits

Vamos a calcular ahora la probabilidad de que, dados un conjunto de vectores aleatorios, alguno de ellos tenga muchos bits solapados con un vector cualquiera. Para esto, empezamos calculando $\Omega^n(x_i, b, k)$, es decir, el conjunto de todos los vectores de tamaño k que coinciden en b bits con el vector x_i en un espacio de dimensión n. Definimos $|x_i|$ como el número de bits activos en x_i, es decir, el número de celdas activas dentro del vector. De esta forma, podemos escribir Ω^n como:

$$\Omega^n(x_i, b, k) = \binom{|x_i|}{b}\binom{n - |x_i|}{k - b} \tag{4}$$

A este número podemos aplicarle ahora un valor mínimo de solapamientos, es decir, podemos quedarnos con los vectores que coincidan en un número mínimo de θ bits, es decir, los que cumplan que $x_i \cdot x_j \geq \theta$ como:

$$\sum_{b=\theta}^{|x_i|} \left|\Omega^n\left(x_i, b, \left|x_j\right|\right)\right| \tag{5}$$

Si ahora elegimos vectores distribuidos uniformemente, podemos calcular la probabilidad de que estos se solapen de forma cuantitativa como

$$P(x_i \cdot x_j \geq \theta) = \frac{\sum_{b=\theta}^{|x_i|} |\Omega^n(x_i, b, |x_j|)|}{\binom{n}{|x_j|}} \quad (6)$$

En [15] se estudia el comportamiento de esta función. Para ello, generan un vector con 24 bits aleatorios como base, que luego comparan con varios vectores aleatorios con a bits activos. Utilizan un número mínimo de solapamientos de $\theta = 12$ bits, es decir, que con sólo la mitad de bits iguales ya se tomarían como vectores muy solapados. Con estos datos, y variando la dimensionalidad n y el número de bits activos en el segundo vector, se obtienen los resultados mostrados en la Figura 13.

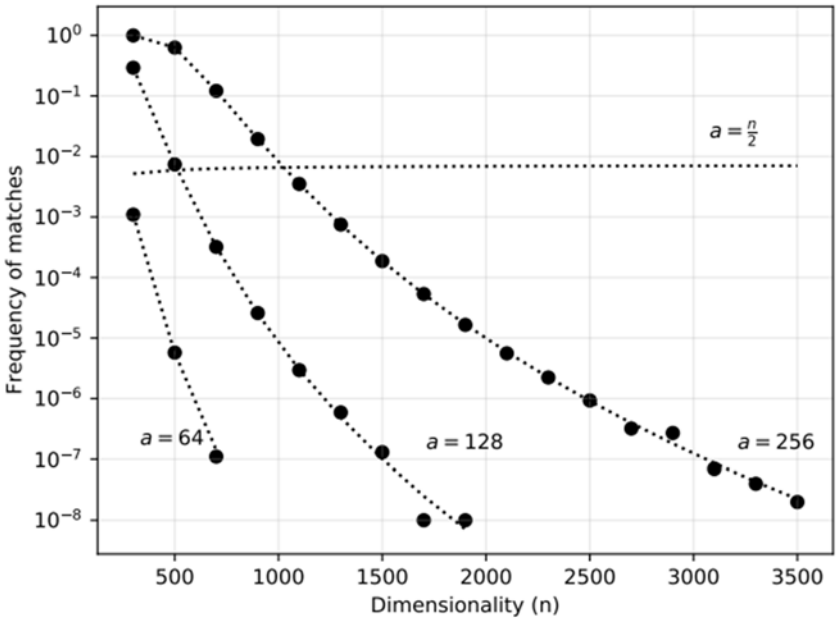

Figura 13: Probabilidad de solapamiento fuerte. Fuente: [15]

Aquí se pueden ver dos comportamientos. En el primero, cuanto mayor es la dimensión, más complicado es encontrar solapamiento. El segundo, cuanto menor es el valor de a, menor dimensión es necesaria para llegar a un solapamiento concluyente. Esto es debido a que, al activar más bits de los vectores aleatorios, es más sencillo que estos se solapen con algún bit del vector usado como base, por lo que para números más pequeños es más complicado que haya coincidencias.

Por lo tanto, es necesario encontrar un equilibrio entre la dimensionalidad el problema y el número de bits que queremos tener activos a la vez. Si aumentamos la dimensionalidad o dejamos el número de bits activos muy bajo, apenas encontraremos vectores repetidos, pero la red no será capaz de encontrar relaciones entre vectores "parecidos", ya que se encontrarán muy lejos entre ellos. Por otro lado, si elegimos una dimensionalidad baja o un número de bits activos alto, la probabilidad de un solapamiento grande entre dos vectores muy diferentes es muy alta, lo que llevaría a perder la robustez de estos sistemas frente al ruido.

Un ejemplo de esta robustez la encontramos en el mismo estudio, en el que comparan dos redes neuronales clásicas, usando una codificación tradicional y SDM. Para esto, se utiliza la base de

datos de MNIST, una base de datos de imágenes con dígitos escritos a mano, cuya finalidad es entrenar a redes para conversión de imagen a texto. A estas imágenes se les aplica un nivel de ruido, en el que simplemente se cambia el color de blanco a negro o viceversa a un porcentaje de los píxeles de forma aleatoria (Figura 14).

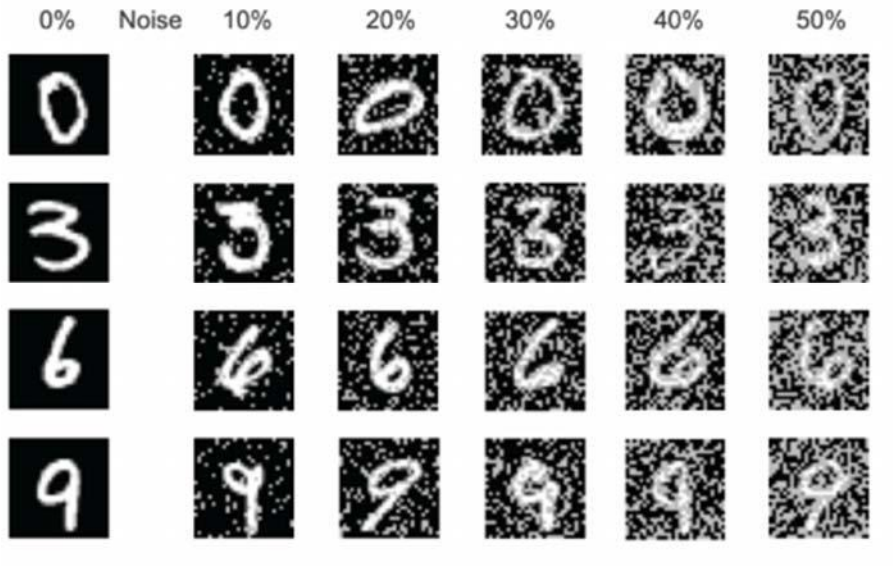

Figura 14: Imágenes MNIST con ruido. Fuente: [15]

La red debe dar como resultado el dígito correspondiente a la imagen. La Figura 15 representa la precisión de las dos redes usadas.

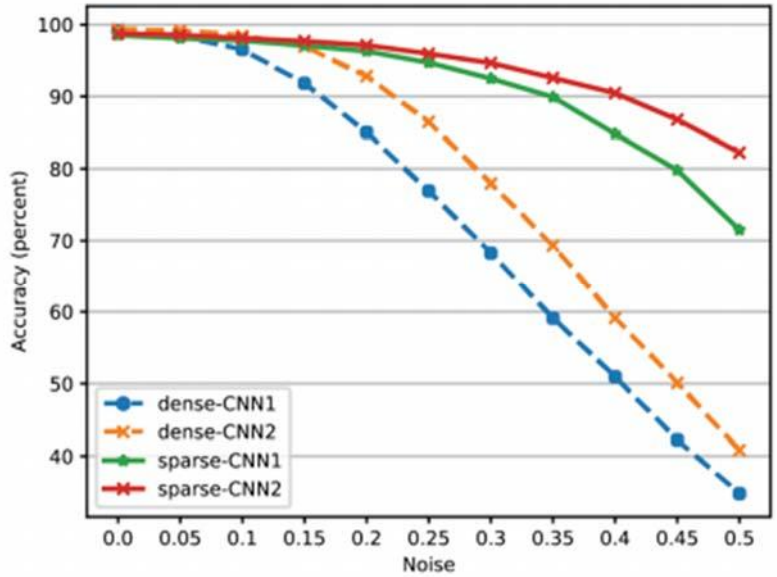

Figura 15: Resultados MNIST. Fuente: 15

Aquí podemos ver cómo las dos representaciones empiezan con una precisión cercana al 99%, pero al añadir un 25% de ruido las redes que emplean codificación tradicional reducen la precisión por debajo del 90%. Esta diferencia se hace más notable en torno a un ruido del 40%, en el que la precisión de las codificaciones tradicionales ha caído hasta un 50% (aciertan tantas como fallan), mientras que las que utilizan un sistema de memoria disperso se mantienen por encima del 80%.

IV. *Modelo para detección de anomalías*

En este apartado mostraremos la solución que hemos desarrollado en este trabajo para la detección de anomalías. Empezaremos mostrando el conjunto de datos que emplearemos para desarrollarlo y la codificación que emplearemos, explicaremos el marco de desarrollo o *framework* usado para el entrenamiento, y terminaremos con una pequeña discusión con los problemas que nos hemos ido encontrando y las soluciones que hemos planteado para solventarlos.

IV.A. Framework

Este trabajo se ha realizado utilizando un marco de desarrollo ya existente. Es decir, no se ha implementado un nuevo núcleo del algoritmo, sino que se ha hecho uso de uno existente. Las principales razones para tomar esta decisión son:

- Programar las funciones más básicas es costoso, a pesar de no tener ninguna dificultad. Por ejemplo, el código de codificación de los datos no es complicado de programar, pero hacerlo para todos los tipos de entradas disponibles puede llevar tiempo.
- Utilizar una alternativa con años de desarrollo será más eficiente. Utilizando un programa hecho por una empresa especializada en este tipo de problemas hará que los algoritmos sean más eficientes, y no perdamos tiempo esperando más de lo debido.

En cuanto a la elección del *framework* a utilizar, no disponemos de muchas opciones. Este tipo de redes neuronales son relativamente nuevas, lo que hace que apenas haya código desarrollado para este sistema, y mucho menos de acceso libre. Sin embargo, la empresa Numenta, ha dado acceso libre al código de varios sistemas de HTM. Esta empresa es una de las más comprometidas con este tipo de redes, como se deduce del hecho de que que la mayoría de publicaciones técnicas sobre estos sistemas incluyen uno o varios autores que trabajan en Numenta. Los códigos que ha dejado libres son: dos implementaciones en las librerías más conocidas de *machine learning* del sistema HTM, y un *framework* que sirve como base para sistemas de anomalías. En nuestro caso, usaremos este último [16], ya que cuenta con herramientas útiles a la hora de trabajar con detectores de anomalías.

Este framework está escrito en Python, un lenguaje de programación interpretado. Esto quiere decir que no es necesario compilar para ejecutar el código, lo cual nos permite hacer cambios en el código y probarlos en un tiempo pequeño. Por esto mismo, el código escrito para este trabajo se ha realizado también en Python. A pesar de usar un *framework*, necesitamos escribir el código que hay por "encima" de toda la carga de datos, salida de datos y dibujado, además de definir los valores de la red usada, ya que el framework no nos define un modelo estándar.

IV.B. Datos y codificación

El primero conjunto de datos, lo podemos obtener desde la página web del ayuntamiento de Nueva York [17]. Estos datos representan la cantidad de pasajeros totales que ha habido en los taxis de la ciudad de Nueva York. Estos datos sufren un pequeño preprocesado como

recomiendan en [18] para disminuir la cantidad de datos e intentar reducir el ruido. Este preprocesado simplemente agrupa los datos en grupos de 30 minutos, teniendo así la suma total de pasajeros cada media hora (Figura 17).

El segundo grupo de datos consiste en información del precio de la publicidad en distintas webs a lo largo del tiempo. Estos datos, de 2011, constan de dos métricas importantes: el coste por *click* (visita publicitaria en una web) y el coste por cada mil visitas (Figura 18). Estas métricas son las más utilizadas por empresas de marketing para decidir cuándo lanzar una campaña publicitaria. Con estos datos, podremos comprobar si existen anomalías para evitar la compra en momentos en los que haya un precio elevado, o hacerla automática si el precio ha bajado en picado durante un tiempo.

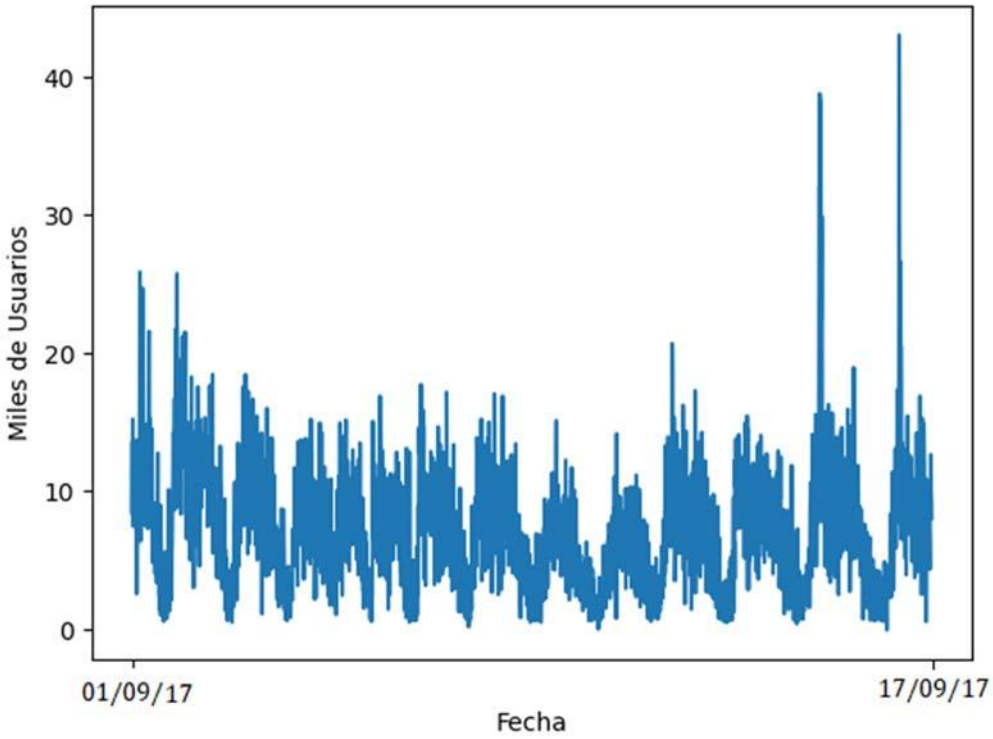

Figura 16: Usuarios del taxi de Nueva York. Fuente: [17]

En la Figura 16 podemos observar representados los datos de los taxis correspondientes a las dos primeras semanas de septiembre. Como podemos ver, los últimos dos días hay un pico de usuarios, duplicando a lo normal el resto de días. Usaremos estas anomalías para comprobar el funcionamiento del programa. El tratamiento de los datos está explicado en el Anexo I.

En el ejemplo del precio de *clicks* las anomalías son más claras, costando un día normal las 1000 visitas entre 0.5 y 2 dólares, pero con picos de hasta 10 dólares, por lo que también podemos usarlo para comprobar la eficacia del programa.

Ambos conjuntos de datos se han hecho pasar por el programa de detección de anomalías propuesto. Este modelo consta de dos entradas, la fecha usando el formato explicado con anterioridad, y un valor numérico que representa el valor en ese punto. Como salida, obtenemos una probabilidad de este valor de ser anómalo. Esta probabilidad se nos expresa como un número en el rango 0-1, luego deberemos decidir a partir de que probabilidad lo consideramos anomalía.

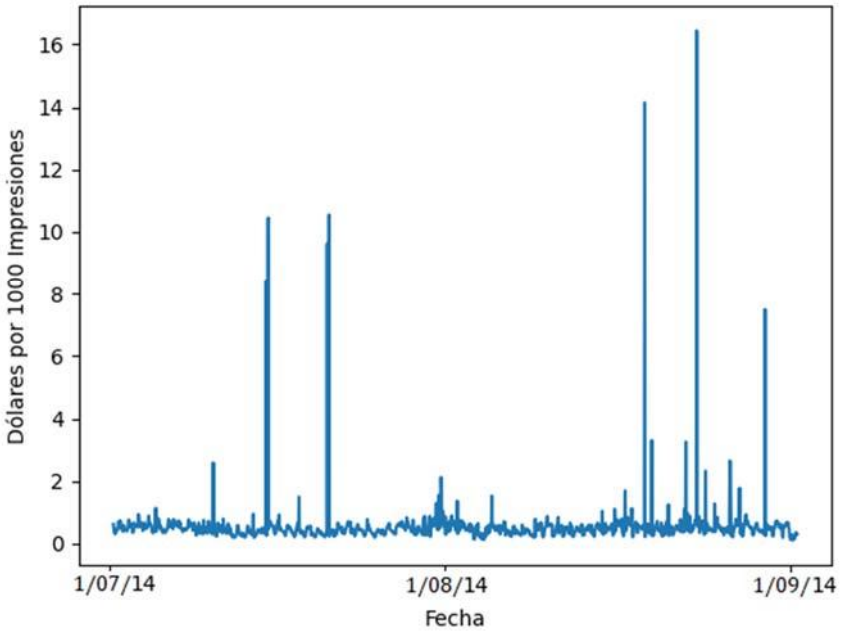

Figura 17: Precio por impresiones. Fuente: [16]

El modelo utilizado consta de una capa de 2048 x 2048 columnas, teniendo cada una de estas columnas 32 neuronas. Al contrario que en las redes tradicionales, aquí no tenemos capas conectadas unas con otras, si no que tenemos una matriz tridimensional de neuronas, las cuales se conectan por pesos a las entradas como hemos visto, pero también se conectan entre neuronas, ya sean de su misma "capa" o distintas (Figura 18), donde vemos dos matrices bidimensionales sacadas contiguas de la matriz tridimensional. De esta forma, una neurona que no se activa en un paso, pero está conectada a una neurona activa, se vuelve una neurona "receptiva", y es más fácil que se active en siguientes pasos. Estas conexiones también varían durante el aprendizaje, al igual que los pesos. Sin embargo, cada neurona tiene un número máximo de conexiones, o sinapsis, que puede tener. En este caso, lo hemos limitado a 255, ya que a más conexiones los resultados no mejoraban, pero había un incremento sustancial en el uso de memoria del programa (al pasar de los 8 bits, era necesario modificar el algoritmo para usar el siguiente contenedor más apropiado, siendo este de 16 bits, por lo que la memoria se duplicaba).

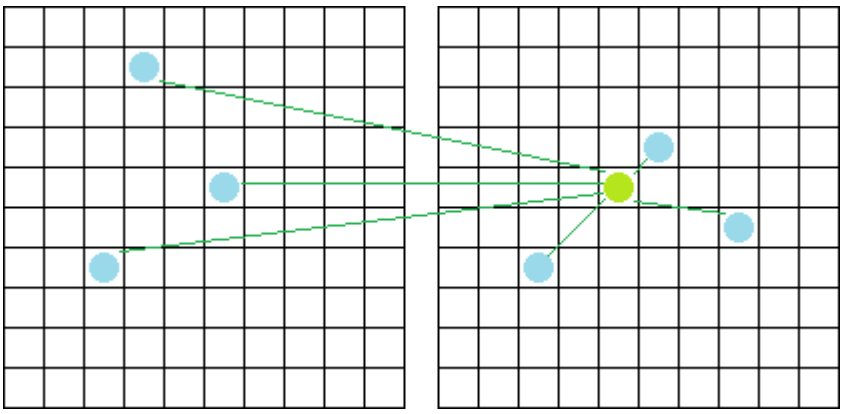

Figura 18: Conexiones de una neurona activa (verde) con otras (azules). Fuente: Propio

Con esto, podemos hacer un diagrama explicativo del programa. Esto lo podemos ver en la Figura 19.

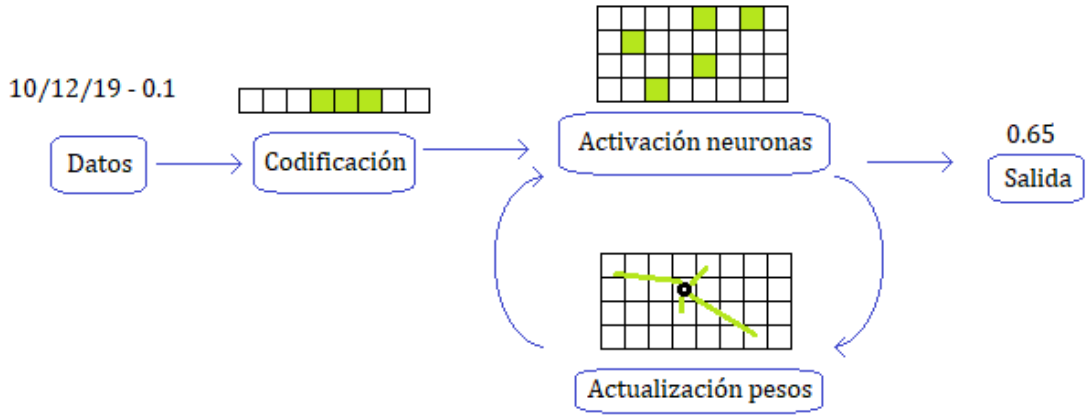

Figura 19: Diagrama de flujo. Fuente: Propia

V. Resultados

En esta sección vamos a estudiar los resultados obtenidos. Empezaremos estudiando el algoritmo en sí, tiempo que tarda en ejecutarse, facilidad de configuración, etc. A continuación, estudiaremos los resultados obtenidos.

Las pruebas se han realizado en un portátil Lenovo B50-70 con sistema operativo Ubuntu 20.04.1 LTS. Este sistema cuenta con 4GB de RAM y un Intel Core i5 con 4 núcleos a 1.7GHz. No es una máquina muy potente, además de no disponer de GPU (unidad de procesado gráfico) para la aceleración de los cálculos. Sin embargo, los algoritmos terminan la ejecución en un tiempo relativamente corto, por lo que podemos usarlo para comprobar la velocidad en el peor de los casos, y ver si es suficiente para lo que necesitamos.

V.A. Rendimiento

Como hemos explicado, el algoritmo se ejecuta en un portátil que sólo tiene CPU. Por suerte, el algoritmo de detección de anomalías está optimizado para utilizar todos los núcleos de los que dispone un ordenador, en nuestro caso, 4 núcleos. Como el producto final será usado frente a un flujo constante de datos, el parámetro que nos interesa es el tiempo medio que tarda en obtener la puntuación de anomalía por entrada. Para esto, medimos el tiempo total que tarda en calcular los resultados para los datos anteriores (Tabla 3).

Tabla 1: Tiempos de ejecución para cada conjunto de datos

	Número de entradas	Tiempo (s)
Taxis	2380	32±1
Publicidad	1643	18±1

Con estos datos, podemos determinar que el tiempo medio que tarda en procesar cada ejemplo es de:

$$t = 0.012 \pm 0.002\ s = 12 \pm 2\ ms \tag{7}$$

Este tiempo puede ser elevado dependiendo de la aplicación para la que se vaya a realizar. Sin embargo, para aplicaciones en las que la entrada de datos sea menor a unas 80 por segundo, este sistema puede responder sin problemas. En el caso de usarse para el cálculo en un conjunto de datos histórico, el tiempo de ejecución podría ser elevado si el número de datos es alto, por lo que para una utilización como esta sería recomendable utilizar un ordenador con más potencia de cálculo.

V.B. Detección de anomalías: resultados experimentales

Vamos a estudiar ahora los resultados de los algoritmos, si han sido capaces de detectar las anomalías presentes en los datos de entrada. Empezamos con los datos de la ocupación de los taxis de Nueva York (Figura 20).

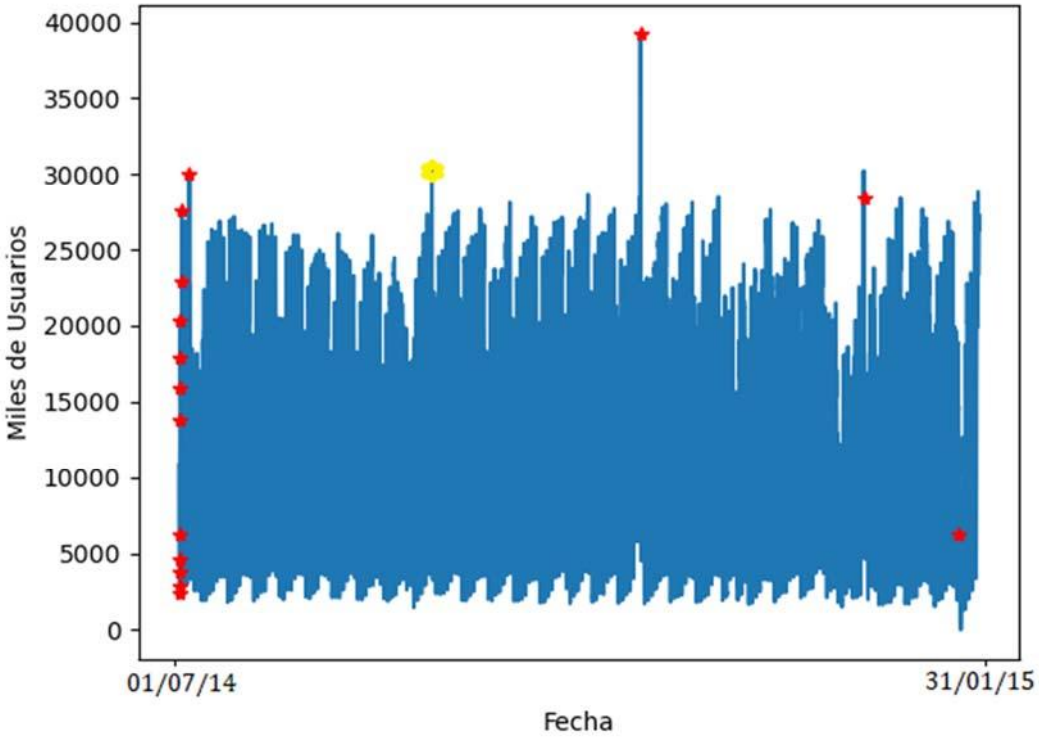

Figura 20: Anomalías detectadas en el alquiler de taxis en NY. Fuente: Propia

Para representar las anomalías, elegimos el umbral para decidir si un evento es anómalo no normal, ya que el algoritmo nos da una probabilidad de anomalía. En este caso, hemos elegido 0.8 como el valor de corte, por lo que los puntos rojos corresponden a puntos donde el detector da un mínimo de 80% de confianza de anomalía.

Como podemos ver, ha detectado como anómalos los primeros datos. Esto es debido a que, al contrario que una red neuronal convencional, la cual es entrenada y luego se puede usar, estas redes van acomodando su respuesta conforme reciben nuevos patrones, "aprenden" lo que es normal. En este caso, la red estaba configurada para que vaya modificando sus parámetros ante nuevos datos, mejorando en principio su estimación. Una alternativa, similar a la de los modelos neuronales convencionales, sería ajustar los parámetros sólo al principio, y luego dejarla en modo autónomo cuando ya haya aprendido el concepto de "normalidad". Este método sin embargo cuenta con un inconveniente, y es que no se adaptará a nuevas situaciones de normalidad que puedan surgir con el tiempo.

En cuanto al resto de picos, ha encontrado dos de los tres picos centrales más grandes a lo largo de la figura. El primero no lo ha considerado anómalo (marcado a mano con un indicador amarillo en la figura) ya que, justo al principio, hay otro pico de características similares, lo que lo hace "normal". Para evitar estos problemas se podrían hacer unas rondas de "entrenamiento" en las que no haya anomalías, para que pudiese detectar correctamente el resto.

Si comprobamos ahora los resultados con los datos de los costes de mil *clicks*, obtenemos los resultados mostrados en la Figura 21.

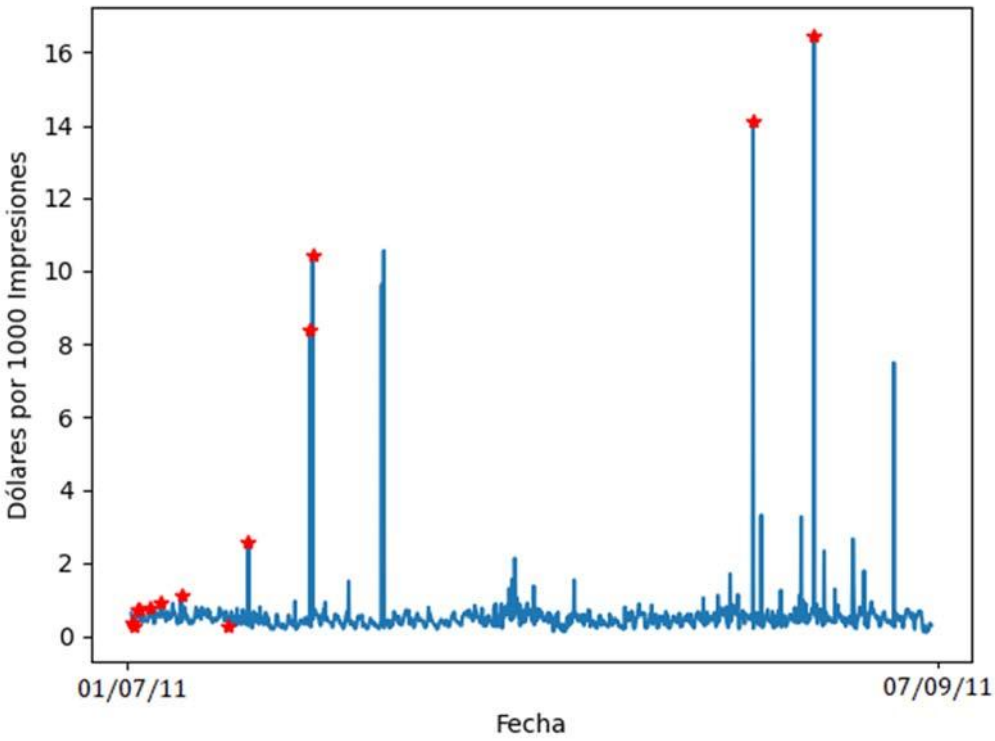

Figura 21: Anomalías detectadas en precios por cada mil clicks publicitarios en web. Fuente: Propia

Aquí podemos observar algo similar a lo visto en la Figura 20 para el ejemplo de los taxis, son necesarios unos pocos datos iniciales para poder detectar anomalías correctamente. Si nos fijamos en los picos más grandes, nos encontramos con dos a la izquierda de la figura, del que sólo ha detectado como anómalo uno. Al igual que en el ejemplo anterior, esto es debido a que el segundo pico es muy parecido al primero, por lo cual ya no es "anómalo" al decirle a la red que aprendiese del primero.

Por el contrario, volvemos a tener dos grandes picos a la derecha, los cuales ahora sí que se han detectado como anomalías. Estos picos ya no son tan parecidos a los anteriores, por lo que el algoritmo nos da una probabilidad mayor de anomalía, y por lo tanto son detectados como tal. A modo de curiosidad, el segundo pico de la izquierda, a pesar de no ser detectado como anómalo, da una puntuación de anomalía más alta que el resto, de 0.5 (50% de probabilidad de ser anómalo). Es decir, si nos interesase que ese pico también fuese detectado, bastaría con modificar ligeramente el valor de corte. La Figura 22 muestra los datos y la probabilidad de anomalía (multiplicada por 10 para poder verla fácilmente). En la figura se puede observar mejor el comportamiento de falsos positivos al inicio. La primera parte del gráfico, presenta saltos entre una probabilidad del 0% y una del 100%, dando muchos falsos positivos. Cuando se normalizan un poco los datos en la siguiente parte, las probabilidades disminuyen, pero sigue habiendo saltos al 100%, aunque podemos observar que hay muchas probabilidades en torno al 1%. Por último, en la última parte, el sistema ha aprendido lo que es "normal", y la probabilidad se parece más a la esperada para los datos de entrada.

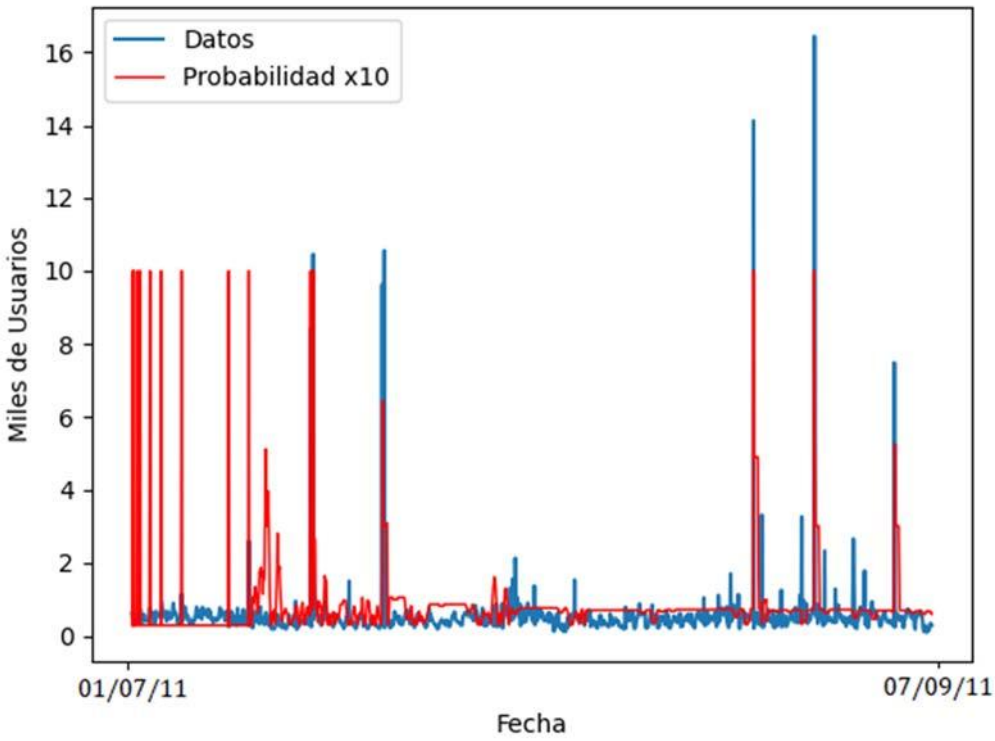

Figura 22: Probabilidad de anomalía. Fuente: Propia

VI. Aplicación en desarrollo

En un principio, el trabajo aquí presentado iba a trabajar con otro tipo de datos, más centrados en el trabajo en la empresa. Sin embargo, por la situación actual, el proyecto inicial ha sufrido demoras, por lo que a la hora de realizar este trabajo aún no disponemos de los datos necesarios para esa aplicación. Sin embargo, como una parte importante es su codificación, y el tipo de datos ya es conocido, vamos a incorporar aquí una breve presentación de cómo son, lo cual también nos servirá para seguir todo el flujo necesario para definir cómo codificarlos, ya que puede no ser trivial.

Lo primero, necesitamos definir cuáles serán nuestras entradas. Aquí, tenemos una o varias señales de vídeo, las cuales serán procesadas por otro sistema autónomo, el cual tiene como salida la etiquetación de los elementos que aparecen en pantalla. En nuestro caso, estos elementos pueden ser contenedores de cargueros, o personas, al igual que el nombre de la persona que aparece (o desconocida si no está en el sistema). Además, tenemos datos sobre la hora del día actual, el día, y si es festivo o laboral.

Para la codificación del día, utilizaremos 5 entradas: día de la semana, si es fin de semana, si es festivo, la hora del día, y la estación del año. El número de bits encendido será de 12 en todas las entradas, y cada una tendrá su propia dimensión:

Tabla 2: Información datos temporales

	Día de la semana	Fin de semana	Día laboral	Hora	Estación
Nº Datos	7	2 (Si/No)	2 (Si/No)	24	4
Dimensión	147	42	42	54	85
Cociente	21	21	21	-	21.25

Esta variabilidad en la dimensión es debida a la propia naturaleza de estas redes. Para que todos los datos tengan la misma importancia, es necesario que tengan la misma “dispersidad”, es decir, que tengan el mismo (o muy parecido) número de bits encendidos por dimensión total. Para esto, las redes HTM definen un parámetro llamado “tamaño de cubo”, el cual se calcula como el cociente entre la dimensión del vector para un dato y el número total de datos que puede representar. Un tamaño alto indicaría un gran número de bits totales para representar muy pocos datos, por lo que estos datos apenas tendrían relación entre ellos. Por otro lado, un número muy bajo indicaría que valores parecidos tendrían un solapamiento muy fuerte. En este caso se ha elegido un tamaño de 21 que proporciona muy buenos resultados cuando tenemos datos numéricos y entradas de fecha. A la hora de decidir el número de bits activados por vector, se suele usar la mitad del tamaño de cubo, por lo que en nuestro caso tendremos 12 bits por vector.

Además de estos datos, contamos con la información extraída de las distintas cámaras de vídeo. De aquí sacaremos: la presencia o no de contenedor, la presencia o no de persona, y si esa persona es conocida o desconocida. Los datos de la identidad de esa persona no van a ser introducidos, ya que esos datos se usan en otros sistemas que permiten comprobar si una persona puede o no puede estar en un lugar. Con esto, los datos de entrada de vídeo serán los mostrados en la Tabla 3.

Tabla 3: Información datos de vídeo

	Hay contenedor	Hay persona
Nº Datos	2 (Si/No)	3 (No/Conocida/Desconocida)
Dimensión	42	63
Cociente	21	21

Aquí, al igual que con los datos anteriores, hemos calculado la dimensión teniendo en cuenta el número de datos posibles para ambas entradas y el tamaño de cubo que hemos decidido para las anteriores entradas. A continuación, todos estos vectores de entrada se colocan en un vector único, el cual será la entrada real de la red, cuya dimensión es igual a la suma de las dimensiones de todos los vectores de entrada. En la Figura 23 vemos un pequeño ejemplo de la unión de dos datos. Hay que tener en cuenta que, a pesar de haber dos colores, sólo hay un vector de salida, y los colores sólo representan que esa parte del vector corresponde a un vector de entrada específico. A la hora de representarlo, se suele colocar en forma matricial para poder verlo de forma más cómoda. Podemos ver un ejemplo de representación en la Figura 24.

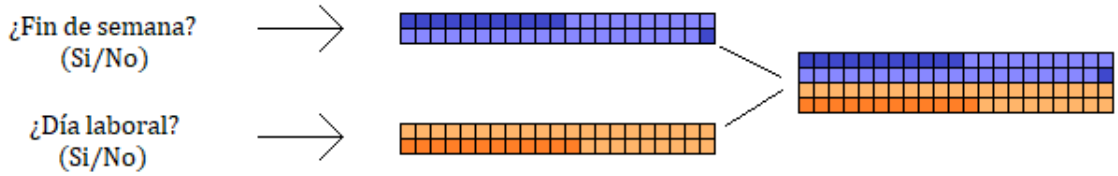

Figura 23:Ejemplo unión de dos datos. Fuente: Propia.

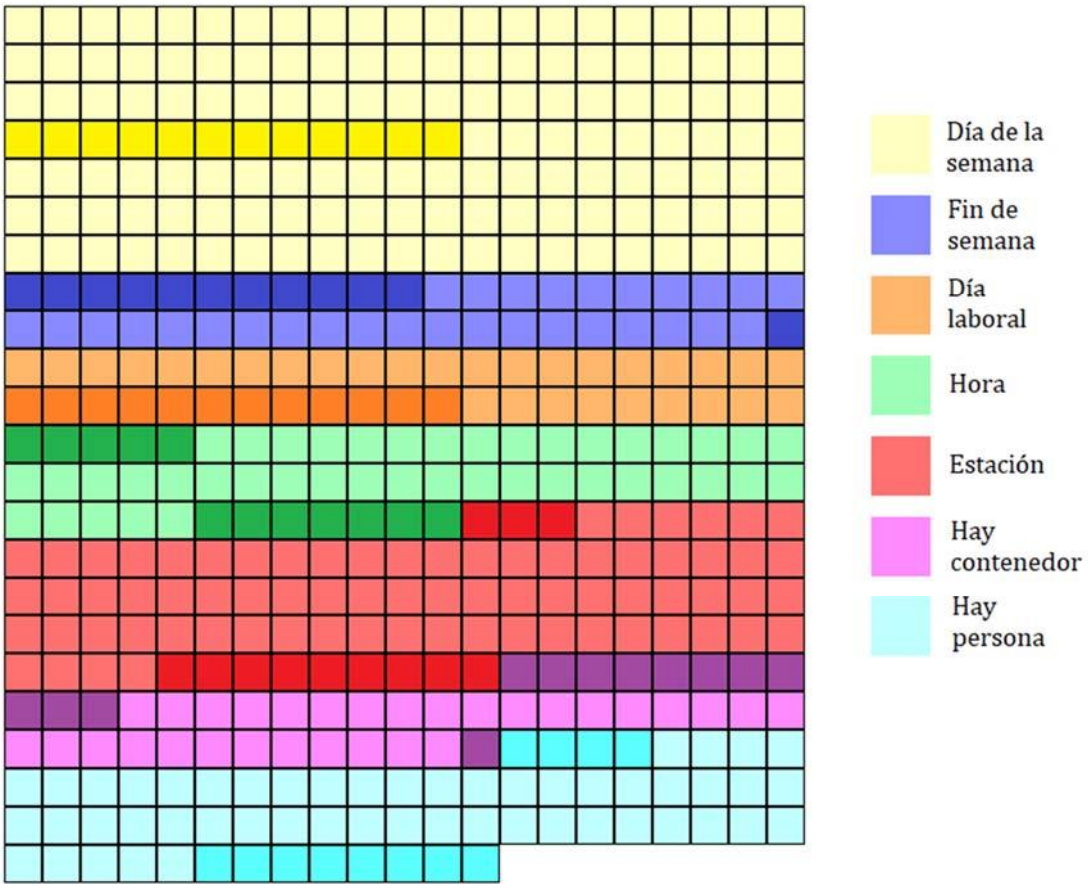

Figura 24: Datos de entrada. Fuente: Propia

Aquí, hemos separado por colores los distintos vectores de las distintas entradas. Además, hemos marcado con un color más oscuro los bits encendidos, para así poder ver que hay datos, como el fin de semana, mucho más poblados que el día de la semana. Esto es debido a que el primero tiene únicamente 2 posibles datos, mientras que el segundo tiene que ser capaz de representar 7 distintos. De esta forma, cada neurona estará conectada a todas las entradas por un valor de peso, y se activará sólo si ese peso es mayor a un valor de corte, el cual puede variar si la neurona se encuentra en un estado receptivo, y esa entrada está encendida.

VII. Conclusiones

En este trabajo, hemos estudiado las anomalías en un conjunto de datos: qué son, cuáles son sus cualidades más características, y cómo identificarlas. Hemos elegido la tecnología que más se adecuaba a nuestro problema, además de buscar los parámetros óptimos para la detección correcta, incluyendo la codificación de los datos de entrada, los parámetros de la red y los parámetros del entrenamiento.

Como trabajo futuro, debería modificarse el algoritmo de forma que permitiera elegir entre un modo de aprendizaje y uno de explotación (o detección de anomalías), ya que actualmente sólo se puede configurar una de las dos posibilidades en toda la sesión de presentación de datos. Sería interesante poder ejecutar comprobaciones en un dato, de manera que se le indicara al algoritmo si ha sido mal interpretado (anomalía a un ejemplo que no lo es, o al revés), para que lo añadiera al aprendizaje adecuadamente etiquetado y así hacer más robusto el sistema.

Además, esto nos permitiría ajustar la normalidad manualmente. Si hay una persona encargada de decidir si las anomalías son anómalas o no, se podrían usar las anomalías no marcadas para aprender que esa es la nueva normalidad. Usando uno de los ejemplos estudiados, si tenemos datos del precio de mil visitas de publicidad, y descubrimos una anomalía porque un sábado el precio aumenta, podríamos decidir a mano que no es una anomalía, ya que se trata de una franja publicitaria en el que el precio es mayor por llegar a un tipo de audiencia, y de esta forma el programa aprendería a esperar esos picos en el precio los sábados.

Otro apartado a realizar en un futuro es el uso de este sistema en la aplicación en desarrollo. Ya hemos hecho una primera aproximación con el ejemplo de cómo codificar los datos de entrada, pero será necesario ajustar los parámetros de entrenamiento para que funcione de forma óptima, ya que estos datos varían entre aplicación y aplicación.

VIII. Bibliografía

1. Varun Chandola, Arindam Banerjee, Vipin Kumar. *Anomaly Detection: A Survey.* ACM Computing Surveys, September, 2009.

2. Meir Toledano, Ira Cohen, Yonatan Ben-Simhon, Inbal Tadeski. 2017. *Real-time Anomaly Detection System for Time Series at Scale.* Proceedings of Machine Learning Research, 2017.

3. Max Landauer, Markus Wurzenberger, Florian Skopik, Giuseppe Settanni, Peter Filzmoser. *Time Series Analysis: Unsupervised Anomaly Detection Beyond Outlier Detection.* Austrian Institute of Technology.

4. Peter J. Rousseeuw, Mia Hubert. *Anomaly Detection by Robust Statistics.* WIREs Data Mining Knowl Discov, 2018.

5. N. S. Altman. *An introduction to Kernel and Nearest Neighbor Nonparametric Regression.* Cornell University, Ithaca, 1991

6. Yong Tat Tan, Bakhtiar Affendi Rosdi. *FPGA-Based Hardware Accelerator for the Prediction of Protein Secondary Class via Fuzzy K-Nearest Neighbors with Lempel-Ziv Complexity Based Distance Measure.* Neurocomputing 145, 2015.

7. Houssam Zenati, Chuan-Sheng Foo, Bruno Lecouat, Gaurav Manek, Vijay Ramaseshan Chandrasekhar. *Efficient GAN-Based Anomaly Detection.* ArXiv 1802.06222v2, 2019

8. Andreas V. M. Herz, Tim Gollisch, Christian K. Machens, Dieter Jaeger. *Modeling Single-Neuron Dinamics and Computations: A Balance of Detail and Abstraction.* Science, 314, 80, 2006.

9. Alan Lloyd Hodking, Andrew Fielding Huxley. *Propagation of Electrical Signals Along Giant Nerve Fibres.* The Royal Society Publishing, 1952.

10. Diedrik P. Kingma, Jimmy Lei Ba. *ADAM: A Method for Stichastic Optimization*. arXiv 1412.6980v9, 2017.

11. S. Hochreiter, J. Schmidhuber. *Long Short-Term Memory.* Neural Computation, 9(8):1735-1780, 1997.

12. Jeff Hawkins, Subutai Ahmad. *Why Neurons Have Thousands of synapses, a Theory of Sequence Memory in Neocortex.* Frontiers in Neural Circuits, 2016.

13. Adrian E. Raftery. *A Model for High-Order Markov Chains.* Royal Statistical Society, 1985.

14. Scott Purdy. *Encoding Data for HTM Systems.* Numenta

15. Subutai Ahmad, Luiz Scheinkman. *How Can We Be So Dense? The Benefits of Using Highle Sparse Representations.* arXiv 1903.11257v2, 2019.

16. *The Numenta Anomaly Benchmark.* GitHub: https://github.com/numenta/NAB (2019)

17. *Taxi and Limousine Commission, New York.* https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page (2014)

18. Subutai Ahmad, Alexander Lavin, Scott Purdy, Zuha Agha. *Unsupervised Real-Time Anomaly Detection for Streaming Data.* Elsevier, Neurocomputing 262, 2017.