Proyecto Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Raspberry pi como plataforma de algoritmos de Machine Learning: Reconocimiento de imágenes y datos financieros en streaming

Autor: Manuel Jesús Rodríguez González
Tutor: Sergio Toral Marín

Dep. de Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2018

Proyecto Fin de Grado
Ingeniería de Telecomunicación

Raspberry pi como plataforma de algoritmos de Machine Learning: Reconocimiento de imágenes y datos financieros en streaming

Autor:
Manuel Jesús Rodríguez González

Tutor:
Sergio Toral Marín
Catedrático

Dep. de Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2018

Proyecto Fin de Grado: Raspberry pi como plataforma de algoritmos de Machine Learning: Reconocimiento de imágenes y datos financieros en streaming

Autor: Manuel Jesús Rodríguez González
Tutor: Sergio Toral Marín

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2018

El Secretario del Tribunal

A mi familia
A mis maestros

Agradecimientos

Con este proyecto acaba lo que ha sido un camino lleno de esfuerzo, sacrificio, superación personal y aprendizaje. Quiero dar las gracias a todos los que han contribuido a hacer de estos años los mejores de mi vida.

Gracias, en primer lugar, a mi tutor, Sergio Toral Marín, por haberme dado la oportunidad de realizar este trabajo.

Gracias a todos los que han sido más que compañeros, Miguel, Carlos, Arturo y muchos más que han conseguido que el camino se hiciera más ameno.

Gracias a mis amigos de la infancia Óscar, Javi y Adri que me han apoyado y ayudado siempre.

Gracias a Marina, que ha sido un pilar fundamental para poder conseguir lo que he conseguido.

Y gracias, especialmente, a mi familia, que me ha apoyado desde el inicio. A mis padres, por hacer que toda esta experiencia haya sido posible.

Gracias a todos.

Manuel Jesús Rodríguez González
Sevilla, 2018

Resumen

El objetivo de este trabajo es evaluar la implementación de modelos predictivos basados en redes neuronales y técnicas de Machine Learning en dispositivos con capacidad de cómputo relativamente baja para modelos de esta índole.

Es por ello que en nuestro estudio el dispositivo seleccionado ha sido la Raspberry pi 3 y las dos aplicaciones que se verán son la clasificación de objetos mediante técnica de reconocimiento de imágenes y la predicción de valores futuros de datos financieros en procesos de datos en streaming.

Se ha optado por introducir bases teóricas afines a las aplicaciones empleadas para favorecer la lectura dinámica del documento y para la correcta asimilación de conceptos.

Abstract

The aim of this work is the implementation of predictive models base don neural networks and Machine Learning techniques on devices with low power strength to this kind of models.

That is why, in our study the selected device has been Raspberry pi 3 and the two applications integrated on it are object classification through image recognition techniques and future values prediction of financial data on streaming.

It has been chosen to introduce teorical basis of applications selected for the dynamic reading of this document and to understanding all concepts.

Índice

Agradecimientos vii
Resumen ix
Abstract x
Índice i
Índice de Tablas iii
Índice de Figuras iv
Notación v
1 Introducción 7
1.1 Motivación 7
1.2 Estructura del proyecto 8
2 Estado del arte 10
2.1 Estado actual 10
2.2 Modelos de predicción en sistemas embebidos 10
2.3 Reconocimiento de imágenes en la actualidad 12
2.4 Predicción sobre datos en streaming en la actualidad 12
2.5 Lenguajes de programación para Machine Learning 13
3 Machine Learning 15
3.1 Aprendizaje supervisado 15
3.1.1 Clasificación 15
3.1.2 Deep Learning 16
3.2 Series Temporales 21
3.2.1 Función de Autocorrelación 21
3.2.2 Tendencias 21
3.2.3 Ruido blanco 22
3.2.4 Random Walk 23
3.2.5 Estacionariedad 24
3.2.6 Modelos Autoregresivos (AR) 24
3.2.7 Modelos de medias móviles (MA) 25
3.2.8 Modelos autoregresivos de media móvil (ARMA) 25
3.2.9 Redes neuronales recurrentes (RNN) 25
3.2.10 MinMax scaling (Normalización) 26
4 Software y hardware empleado 28
4.1 Raspberry pi 3 28
4.2 Raspbian 29
4.3 Python 29
4.4 IDE: Spyder 29
4.5 Librerías 30
4.5.1 TensorFlow 30
4.5.2 Alpha-vantage 31
5 Resultados 33
5.1 Reconocimiento de imágenes 33

5.1.1 Metodología 33
5.1.2 Resultados experimentales 38
5.2 Datos financieros en streaming *38*
5.2.1 Metodología 38
5.2.2 Resultados experimentales 39
6 Conclusiones y líneas futuras **43**
7 Referencias **46**
Anexo: Código de Ejemplos **48**

ÍNDICE DE TABLAS

Tabla 1. Caracterícticas de Edge Computing y Fog Computing 11
Tabla 2. Desempeño de Raspberry Pi con reconocimiento de imágenes..... 38
Tabla 3. Desempeño Raspberry Pi datos financieros en streaming. 39

ÍNDICE DE FIGURAS

Figura - 1. Función Sigmoidea 16
Figura - 2. Estructura de una neurona. 17
Figura - 3. Representación de una neurona computacional. 18
Figura - 4. Red neuronal computacional. 18
Figura - 5. Perceptrón. 19
Figura - 6. Red neuronal Convolucional (CNN). 20
Figura - 7. Red neuronal Recurrente (RNN). 20
Figura - 8. Mean Reversion. 21
Figura - 9. Momentum. 22
Figura - 10. LSTM. 26
Figura - 11. Arquitectura Raspberry Pi 3. 28
Figura - 12. Ejemplos dataset CIFAR-10. 34
Figura - 13. Función lineal con Scores. 34
Figura - 14. Aplicación función Softmax. 35
Figura - 15. Rectified Linear Units. 36
Figura - 16. Modelo de 2 capas. 36
Figura - 17. CNN de reconocimiento de imágenes. 37
Figura - 18. Modelo de 2 capas 40
Figura - 19. Modelo de 4 capas. 40
Figura - 20. Modelo de 5 capas. 40
Figura - 21. Ejemplo de ejecución de programa(1/2) 40
Figura - 22. Ejemplo de ejecución de programa (2/2) 41

Notación

h_θ	Hipótesis
θ	Pesos del modelo
x	Variables de entrada
y	Etiquetas de salida
J	Función de coste
g	Función Sigmoidea
log	Logaritmo
$a_i^{(j)}$	Unidad de activación i en capa j
S	Función Softmax
e	Constante matemática
D	Cross Entropy
L	One-Hot Encoding
$\mathcal{L}$	Función Loss
N	Número de muestras
$corr(x, y)$	Correlación entre x e y
R^2	R squared
$sign$	Función signo
P_t	Precio en instante t
P_{t-1}	Precio en instante $t-1$
ϵ_t	Ruido blanco Gaussiano
μ	Media
ϕ	Parámetro de modelos autoregresivos
AR(n)	Modelo autoregresivo de orden n
MA(n)	Modelo de media móvil de orden n

1 INTRODUCCIÓN

It is not knowledge, but the act of learning, and not possession, but the act of getting here, which grants the greatest enjoyment
- Carl Friedrich Gauss -

Actualmente, la inteligencia artificial está siguiendo una corriente muy fuerte desde que se le acuñase este término en el año 1956 por John McCarthy. Las interesantes disciplinas que abarca la hacen atractiva a la vez que intimidante por los progresos no controlados que puedan realizarse en el futuro. La posibilidad de implementar estos algoritmos en dispositivos embebidos con limitaciones de procesamaiento para dotarles de inteligencia local es la base que impulsa este proyecto.

El objetivo principal es la necesidad de comprobar el desempeño de dispositivos de capacidad de cómputo bajo frente a modelos predictivos desarrollados con librerías que necesitan mucha potencia como TensorFlow, y medir como actúan ante la complejidad de una red neuronal

1.1 Motivación

La principal motivación ha sido ver cómo el software que hace años no podrían soportar ordenadores ahora lo soporta un dispositivo adaptable y de fácil uso como es la Raspberry pi. Actualmente, el machine learning es una tecnología que está en auge, sobre todo por la potencia que tienen las redes neuronales capaces de imitar un cerebro humano y realizar tomas de decisiones en base a un entrenamiento previo.

Hoy en día la tecnología es ubicua en muchos dispositivos que nos rodean, muchos de ellos pueden implementar algoritmos muy complejos. Por ejemplo, móviles y televisores actualmente integran modelos predictivos para mejorar la experiencia del usuario.

Por último, su integración en la industria que hace que empresas que elaboren una transformación digital de las mismas dotarán a sus sistemas de modelos basados en datos históricos para mejorar la experiencia de un usuario, personalizar un producto o realizar predicciones de cara al futuro.

1.2 Estructura del proyecto

Sobre la estructura de este proyecto, se empezará tratando información en el apartado *2. Estado del arte* sobre el estado actual de proyectos de esta índole y las diferentes aplicaciones predecesoras y actuales similares a las que se han implementado.

En el apartado *3. Machine Learning*, se verá el lado teórico de lo que es el Machine Learning y a su vez ejemplos de los aprendizajes que existen junto a una introducción teórica de las redes neuronales.

Un estudio del software y el hardware que se utilizará para realizar en el proyecto se encontrará en el apartado *4. Software y hardware empleado* haciendo énfasis en las librerías más básicas para realizar modelos predictivos y para el análisis de datos y a su vez recalcar las limitaciones que nos encontraremos debido al hardware usado.

El apartado con información experimental se encuentra en el apartado *5. Resultados,* donde además de comprobar los resultados y el desempeño de la Raspberry pi 3 frente a estos modelos se darán bases teóricas sobre las distintas aplicaciones para ayudar a la comprensión de la aplicación.

Por último, en el apartado *6. Conclusiones y líneas futuras* se hará una reflexión de las distintas aplicaciones que tiene en un futuro la implementación de modelos en dispositivos.

2 ESTADO DEL ARTE

"What is the magic trick that make us smart? The trick is thast there is no trick. The power of intelligence emanates from our vast diversity, not from a single and perfect principle."

- Marvin Minsky -

Debido a la fuerza de cómputo que hemos adquirido con el paso de los años, muchos de los algoritmos dedicados a la predicción que dejamos atrás se han ido retomando. *Machine Learning* o *aprendizaje automático* son las palabras con las que hoy en día se define este ámbito. Programamos los algoritmos matemáticos y somos capaces de evaluar sistemas expertos capaces de predecir en función de cantidades de datos al alcance de muchos.

2.1 Estado actual

El proceso de integración de modelos predictivos en dispositivos ha ido desarrollándose desde un ordenador hasta tal punto de disponer de sistemas expertos en dispositivos móviles.
La Raspberry pi es ordenador de tamaño pequeño de bajo coste. Su fuerza de cómputo ha hecho de ella un dispositivo apetecible para un abanico de proyectos entre ellos de aprendizaje automático basados en redes neuronales con una fuerza de cómputo considerable.

Actualmente, Google ha lanzado una encuesta en la cual se pregunta a todos los usuarios sobre los intereses de estos para añadir herramientas desarrolladas por ellos para enfocar la Raspberry pi en campos como: drones, robótica, IoT[1], impresoras 3D, wearables[2], domótica, o aprendizaje automático. A través de librerías como TensorFlow[3], Google quiere sacarle el máximo partido y cubrir las necesidades de la gente con dispositivos de tal calibre. [1]

Existen diversos proyectos que emplean la Raspberry pi como plataforma para reconocimiento de voz y de imágenes aprovechando las prestaciones de esta y accesorios como la Pi Camera.

2.2 Modelos de predicción en sistemas embebidos

La fuerza de computación que los sistemas embebidos han ido adquiriendo con el tiempo nos han ayudado a integrar modelos predictivos y algoritmos en estos dispositivos, un ejemplo claro son los smartphones. De hecho, un reciente estudio de ABI Research afirma que esta tecnología está destacando frente a otras, generando aplicaciones y usos de inteligencia artificial y machine

[1] IoT: *Internet of Things,* Denominación de la interrelación entre máquinas mecánicas y digitales que tienen identificadores únicos y con la capacidad de transferir datos en la red.
[2] Wearables, Prendas con capacidad de monitorizar datos biológicos y transferirlos.
[3] TensorFlow, librería desarrollada por Google Brains para el diseño de redes neuronales, véase el punto 4.5.1 .

learning. Así, según el informe, se prevé que para el año 2022 contaremos con más de 650 millones de dispositivos con aplicaciones de visión artifical. [2]

La Raspberry pi es un dispositivo muy versátil pero debemos contar con las limitaciones que nos vamos a encontrar. Por una parte, hay limitaciones a nivel de software ya que este dispositivo no es compatible con el resto de sistemas operativos que usamos porque utiliza una arquitectura ARM[4]. Por eso, tenemos que usar un sistema operativo basado en Debian llamado Raspbian. Además, sumemos el lenguaje de programación Python que no es compilado sino interpretado que consumen más tiempo de procesamiento al tener que ser traducido al lenguaje máquina con cada ejecución. Por otra parte, limitaciones a nivel de hardware ya que el dispositivo cuenta con 1 Gb de memoria RAM la cual no es muy adecuada para el entrenamiento de modelos que necesitan ingestas de datos muy grandes y capacidad de procesamiento altas para poder mostrar un carácter eficiente.

En dispositivos embebidos existen 2 posibilidades para conseguir una inteligencia local:

- Procesamiento en la propia tarjeta
- Procesamiento en la nube

En el caso de procesamiento en la nube contamos con una API[5] que nos permite subir datos a la nube y ésta nos devuelve los resultados requeridos, evadiendo así el proceso de entrenamiento y de predicción en nuestro dispositivo. Un posible ejemplo sería subir una imagen a la nube y que el modelo predictivo se encargue de darnos como resultado qué representa dicha imagen. Las empresas líderes en este tipo de procesamiento son Google y Microsoft. [3]

Actualmente, Amazon también ha desarrollado una plataforma con la capacidad de subir tus propios modelos o utilizar los de ellos a su sistema, la plataforma se llama Amazon SageMaker.

En nuestro caso, se usa el procesamiento en la propia placa ya que cada vez contamos contamos con dispositivos más rápidos y con más facilidad de implementación, en nuestro caso Python sobre Raspbian. Muchas veces el procesamiento en la nube tiene inconvenientes, como el tiempo que tarda en obtenerse una respuesta, que a veces los datos capturados pierden valor con el tiempo. Todo esto entraría en el Edge Computing vs Fog computing, en la Tabla 1 te puede identificar las principales diferencias entre ambos. [4]

Tabla 1. Caracterícticas de Edge Computing y Fog Computing

	Fog Computing	**Edge Computing**
Datos	Se recopilan, procesan y almacenan a través de la red	Parte en red y parte en local
Procesamiento	Dispositivo único y potente	Varios dispositivos
Probabilidad Fallo	Alta	Baja

[4] ARM, Arquitectura de procesador.
[5] API, *Application Programming Interface*, Conjunto de funciones que ofrece una librería para su posterior uso en otro software.

Por otra parte, hay que diferenciar entre el entrenamiento del modelo y la implementación, ya que cuando se entrena un modelo podemos implementarlo en cualquier dispositivo, aunque no se haya entrenado dicho modelo. En este caso, los modelos son entrenados en un ordenador y posteriormente se implementan ya entrenados en la Raspberry Pi.

2.3 Reconocimiento de imágenes en la actualidad

Existen diversas aplicaciones de reconocimiento de imágenes, podemos destacar *Google Goggles,* una aplicación capaz de reconocer imágenes a partir de fotos que realicemos con nuestro móvil y devolver información detallada y de la cual Google se ha deshecho después de desarollar Nougat. [3]

En dispositivos con sistema operativo iOS[6] se integran aplicaciones embebidas capaces de discretizar personas en función del reconocimiento facial y cuantificando la frecuencia en la que aparecen en las fotos. Además, podemos destacar la funcionalidad que ofrecen varios dispositivos de desbloqueo en función de la cara del poseedor.

Con la integración de modelos basados en redes neuronales y el lanzamiento de librerías como *TensorFlow* de Google o *Keras*[7]*,* de las cuales hablaré en apartados posteriores, se ha conseguido integrar en sistemas embebidos diversas herramientas que hacen uso de estas librerías para realizar reconocimientos.

Respecto a aplicaciones en Rasperry Pi, existen diversas aplicaciones con librerías como OpenCV[8] capaces de reconocer imágenes de todo tipo y haciendo uso de la Pi Camera.

2.4 Predicción sobre datos en streaming en la actualidad

Actualmente es un campo en continua investigación ya que con la generación de grandes cantidades de datos nació la necesidad de obtener dichos datos y aprovecharlos eficientemente. Por eso, este concepto está íntimamente relacionado con el *Big Data*[9] ya que se precisa de herramientas para la ingesta de dichos datos.

Dentro de la obtención de datos, podremos clasificar entre dos tipos. Por una parte, la ingesta de datos estática, en la cuál obtenemos los datos y los almacenamos con herramientas diseñadas y en Bases de Datos no relacionales para su posterior análisis y uso. Por otra parte, la ingesta de datos dinámica o en streaming, mediante la cual obtenemos datos en tiempo real, los analizamos y usamos modelos predictivos que sean capaces de ir aprendiendo dinámicamente para realizar una correcta identificación de patrones o para realizar predicciones.

Además, actualmente existen distintas APIs capaces de obtener datos *real-time.* Entre estas están Alpha Vantage, Polygon.io, etc…

[6] iOS, Sistema operativo desarrollado por Apple Inc.
[7] Keras, Librería para el diseño de redes neuronales.
[8] OpenCV, Librería para el tratamiento de imágenes.
[9] Big Data, Engloba toda tecnología encargada del procesamiento y tratamiento de grandes ingestas de datos.

2.5 Lenguajes de programación para Machine Learning

Los lenguajes de programación más usados son Python, R, C++, Java y Scala. R se usa debido a que es un lenguaje muy orientado al análisis estadístico y cuenta con muchas librerías desarrolladas para el análisis de datos y creación de modelos. A su vez, Python destaca por su sintaxis intuitiva y una gran variedad de librerías para el desarrollo de aplicaciones de Machine Learning, las más relevantes son:

- *Pandas*: Nos ofrece una forma versátil de recoger estos datos en streaming para poder procesarlos posteriormente en dataframes.
- *SciPy*: Herramientas de matemáticas, ciencia e ingeniería.
- *Statsmodel*: Nos provee herramientas dedicadas para el análisis de series temporales.
- *Scikit*-learn: Para algoritmos de Machine Learning.
- *Numpy*: Para vectores y matrices.
- *TensorFlow*: Para integración de redes neuronales.
  Keras: Al igual que TensorFlow, es una librería destinada al desarrollo de redes neuronales.

3 MACHINE LEARNING

"Machine learning is a core, transformative way by which we're rethinking everything we're doing".

- Google -

Es una disciplina científica que se enfoca en crear sistemas que aprenden de datos y son capaces de realizar predicciones e identificar patrones complejos en ellos. Esta definición implica que estos sistemas se mejoran de forma autónoma. *Tom Mitchell* ofrece una definición moderna de lo que sería el Machine Learning:

"*Se dice que un programa de ordenador aprende de la experiencia E con respecto a alguna clase de tareas T y la medida de rendimiento P, si su desempeño en tareas en T, medido por P, mejora con la experiencia E*."

En él podemos identificar dos tipos: Aprendizaje supervisado y aprendizaje no supervisado. La principal diferencia entre ambos está en el conjunto de datos que usaremos para entrenar, si usamos datos etiquetados o con un valor real estaremos hablando de aprendizaje supervisado mientras que, si el conjunto de datos no está etiquetado, nuestro algoritmo deberá buscar un patrón en estos, por lo que estaremos ante un problema de aprendizaje no supervisado. [4]

3.1 Aprendizaje supervisado

Necesitamos un conjunto de datos de los cuáles un porcentaje de ellos será destinado al entrenamiento de la máquina, otro a la validación del modelo y el restante al test en el que medimos cuánta precisión tiene nuestro modelo. En el aprendizaje supervisado podemos diferenciar entre 2 dos tipos de problemas: Regresión y clasificación. Se hará más énfasis en la clasificación.

3.1.1 Clasificación

La principal diferencia entre los problemas de regresión y los de clasificación es que nuestra predicción será un valor discreto mientras que en problemas de regresión será un valor continuo.

Nos centraremos en un **problema de clasificación binaria**[10]. Si tuviéramos las analíticas de varios pacientes y quisiéramos diagnosticar si tienen cáncer o no estaríamos ante un problema de

[10] Clasificación binaria, problema mediante el cual las salidas son binarias, es decir, 1 ó 0.

clasificación ya que podríamos tratarlo como un problema binario en el que la salida *tiene cáncer* se correspondería con un 1 y *no tiene cáncer* sería un 0.

Para estos problemas es usado un modelo de regresión logística como el siguiente:

$$h_\theta(x) = g(\theta^T x) \qquad (3.1)$$

$$z = \theta^T x \qquad (3.2)$$

$$g(z) = \frac{1}{1+e^z} \qquad (3.3)$$

Podemos diferenciar entre la hipótesis (fórmula 3.1), nuestros parámetros multiplicados por los pesos (fórmula 3.2) y nuestra función sigmoidea (fórmula 3.3).

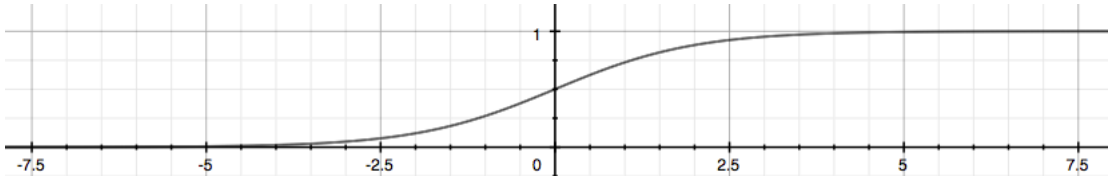

Figura - 1. Función Sigmoidea

En la figura 1 podemos comprobar como los valores de esta función van de 0 a 1. Nuestra hipótesis $h_\theta(x)$ nos dará la probabilidad de que sea 1 ó 0. Por ejemplo, un valor de $h_\theta(x) = 0.4$ nos está diciendo que hay un 40% de que sea 1.

Como en la regresión, necesitamos una función de coste para medir el error, la función es la siguiente:

$$J(\theta_0, \theta_1) = \frac{1}{m}\sum_{i=1}^{m}(y^i \log\left(h_\theta(x^i)\right)) + (1 - y^i)\log\left(1 - \left(h_\theta(x^i)\right)\right) \qquad (3.4)$$

La fórmula 3.6 nos muestra la función de coste dónde m es el número de ejemplos, $h_\theta(x_i)$ es la predicción para una entrada x_i y y_i el valor real para ese ejemplo i.

Nuestro objetivo es minimizar el error como en los problemas de regresión, así que usaremos también técnicas metaheurísticas como el gradiente descendente[11]. [5]

3.1.2 Deep Learning

El Deep Learning nació de la creación de redes neuronales profundas, por eso la denominación de Deep Learning o aprendizaje profundo. Es una materia que actualmente crea diversidad de opiniones ya que puede encontrarse bien como una parte englobada por el Machine Learning o bien por separado. Su objetivo es la imitación del cerebro humano y en concreto de las redes

[11] Gradiente descendente, Algoritmo de optimización.

neuronales humanas. Muchas aplicaciones de las redes neuronales están relacionadas con el NLP[12] y en concreto reconocimiento de imágenes y reconocimiento de voz. [6]

3.1.2.1 Historia

En 1943, Warren McCulloh y Walter Pitts dieron los primeros fundamentos, explicando la forma de trabajar de las neuronas y modelaron una red neuronal mediante un circuito eléctrico.

En 1957, Frank Rosenblatt desarrolló el Perceptrón que es la red neuronal más antigua, actualmente usada en el reconocimiento de patrones. 10 años después, en 1967, Marvin Minsk y Seymour Papert demostrarón que el Perceptrón no era robusto ya que no podía resolver problemas no lineales. [7]

En 1974, se desarrolló el algoritmo Backpropagation[13] por Paul Werbos.

A partir de 1985, las investigaciones sobre las redes neuronales se incentivaron. Se utilizaron en los 80s y 90s. Con el objetivo de crear máquinas capaces de simular el cerebro humano. Cayeron en desuso a finales de los 90 debido a que las operaciones que realizaban eran demasiado complejas y no existía potencia de cálculo suficiente para implementarlas.

El nombre viene del símil con las neuronas humanas en las que podemos identificar 3 partes:

- **Axón**: Salida de la neurona. Sirven para mandar impulsos o señales a otras neuronas.
- **Dendritas**: Entradas de la neurona. El axón se conecta con las dendritas de otras neuronas.
- **Núcleo**: Cuerpo de la neurona.

En la figura 2 podemos encontrar una ilustración con las partes de una neurona:

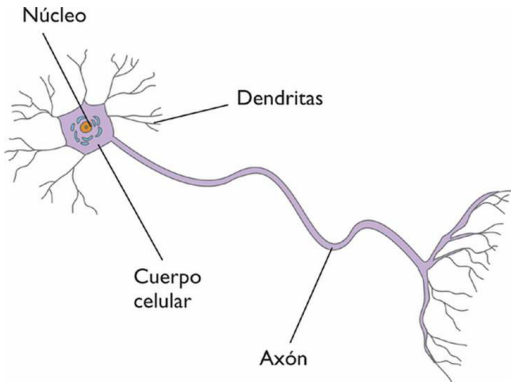

Figura - 2. Estructura de una neurona.

[12] NLP, *Natural Language Processing*, Es una disciplina de la Inteligencia Artificial que se centra en las interacciones humano-máquinas.
[13] Backpropagation, Algoritmo de las redes neuronales para encontrar los valores del modelo predictivo.

3.1.2.2 Representación

Modelamos las neuronas como unidades lógicas de la siguiente forma:

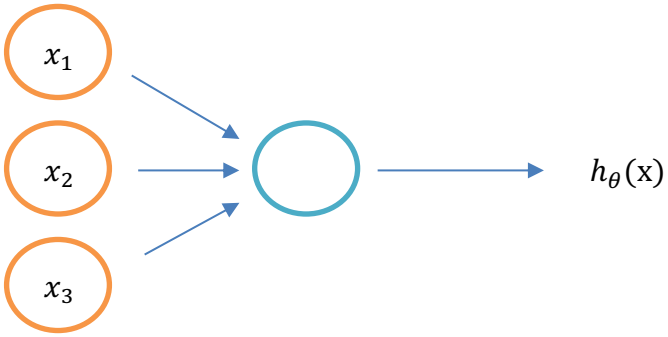

Figura - 3. Representación de una neurona computacional.

En la figura 3 podemos ver la representación de una neurona computacional donde las x representan las entradas, el círculo azul representa una función de activación y $h_\theta(x)$ es la salida.

Además de las x representadas aún faltaría x_0 que representaría la unidad de oscilación porque su valor es 1.

La figura representada muestra lo que es una neurona, si visualizamos una red de neuronas con un ejemplo de 3 capas sería asi:

pas sería asi:

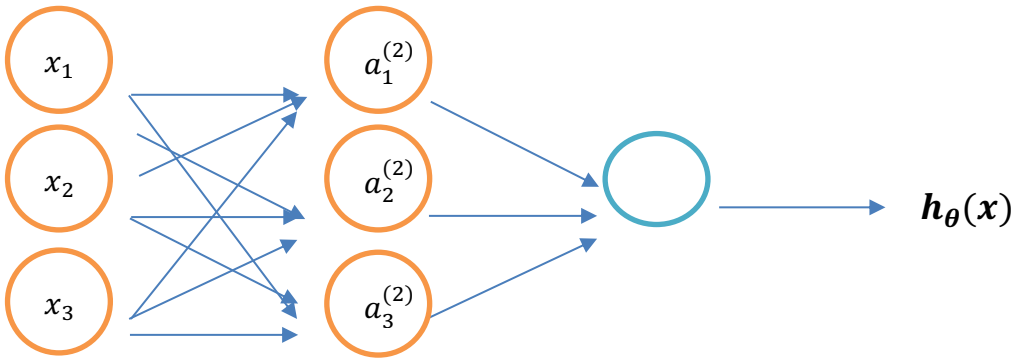

Figura - 4. Red neuronal computacional.

En la figura 4 podemos ver una red neuronal con 3 capas diferenciadas. La capa 1 es la capa de entrada ya que es la capa donde vamos a introducir las variables de entrada, la capa 3 es la capa de salida ya que es dónde se realiza la predicción y la capa 2 es la capa oculta; si tuviéramos más de 1 capa entre la 1 y la 3, éstas serían ocultas. [5]

Podemos identificar los siguientes parámetros:

- $a_i^{(j)}$: Representa a la unidad de activación i en la capa j.
- θ^j : Matriz de parametrización de capa j.

Los valores para las unidades de activación son los siguientes:

$$a_1^2 = g\left(\theta_{10}^{(1)}x_0 + \theta_{11}^{(1)}x_1 + \theta_{12}^{(1)}x_2 + \theta_{13}^{(1)}x_3\right) \quad (3.5)$$

$$a_2^2 = g\left(\theta_{20}^{(1)}x_0 + \theta_{21}^{(1)}x_1 + \theta_{22}^{(1)}x_2 + \theta_{23}^{(1)}x_3\right) \quad (3.6)$$

$$a_3^2 = g(\theta_{30}^{(1)}x_0 + \theta_{31}^{(1)}x_1 + \theta_{32}^{(1)}x_2 + \theta_{33}^{(1)}x_3) \quad (3.7)$$

Y el valor de la hipótesis es:

$$\boldsymbol{a_1^3 = g\left(\theta_{10}^{(2)}a_0 + \theta_{11}^{(2)}a_1 + \theta_{12}^{(2)}a_2 + \theta_{13}^{(2)}a_3\right) = h_\theta(x)} \quad (3.8)$$

3.1.2.3 Tipos

Entre las más conocidas están los siguientes tipos de redes neuronales:

- **Perceptrón:** Es una neurona artificial capaz de recoger todas las entradas y multiplicarlas por sus respectivos pesos y realizar una función de activación para poder dar un resultado.

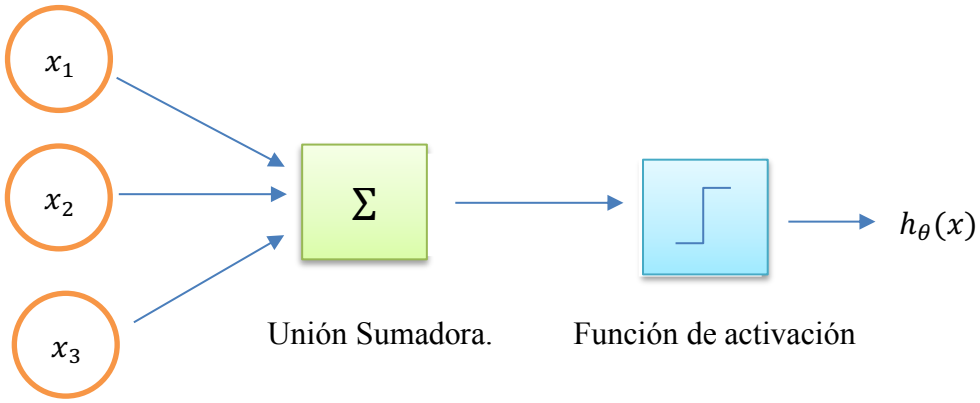

Figura - 5. Perceptrón.

- **CNN (Convolutional Neural Network) o CovNet**: Es una variación del perceptrón multicapa, son muy efectivas para aplicaciones de visión artificial.

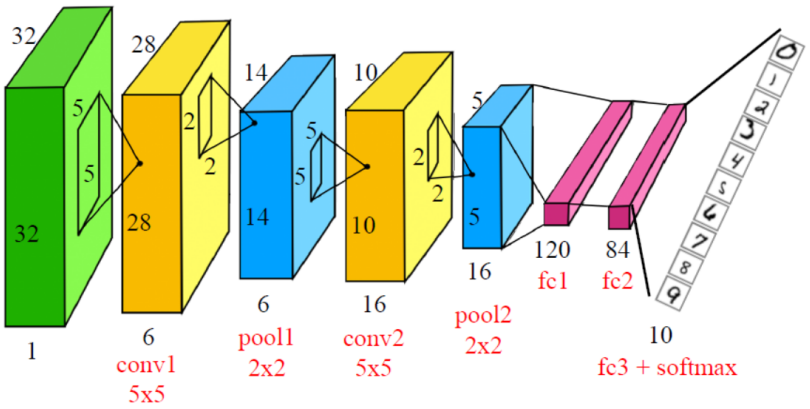

Figura - 6. Red neuronal Convolucional (CNN).

- **RNN (Recurrent Neural Network):** Es un tipo de red en el que las unidades tiene memoria lo que le permite exhibir un comportamiento temporal dinámico para una secuencia de tiempo.

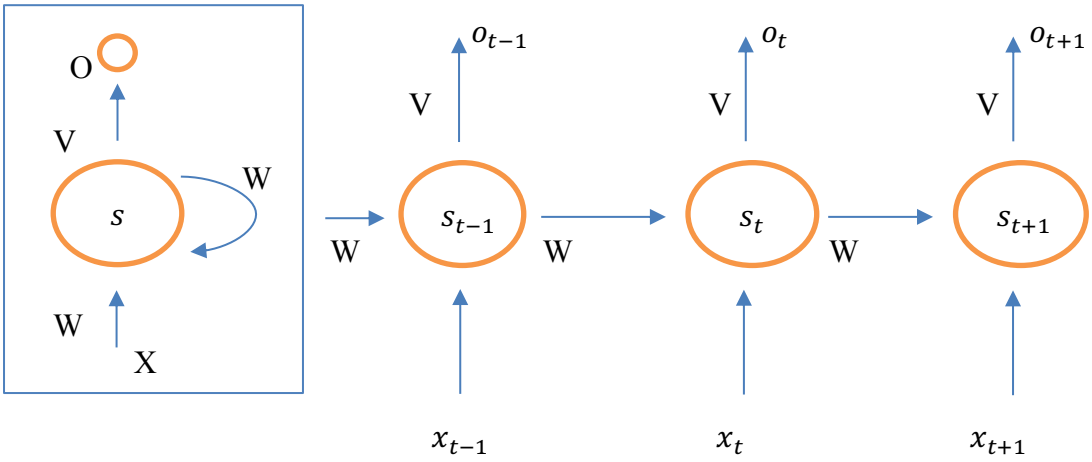

Figura - 7. Red neuronal Recurrente (RNN).

En este caso, para la aplicación de reconocimiento de imágenes se usarán dos tipos de modelos: clasificador logístico (Machine Learning) ya que es un problema discreto en el que las salidas son clases (Ej: Avión, rana, gato…) y se usarán también redes neuronales convolucionales basadas en TensorFlow, ya que ofrecen un gran rendimiento en este tipo de problemática.

Para la segunda aplicación, datos financieros en streaming, usaremos dos tipos de modelos también, se aplicará un modelo ARMA (Autoregresivo de media móvil) basado en la librería `statsmodel,` ya que los datos financieros son series temporales, se usarán redes neuronales recurrentes en TensorFlow explotando asi su principal funcionalidad (cada neurona tiene un estado de memoria).

3.2 Series Temporales

3.2.1 Función de Autocorrelación

La función de autocorrelación simple de una serie proporciona la estructura de dependencia lineal de la misma. Si denominamos z_t a la serie temporal, los valores que se observan van a ser:

$$z_1, z_2, \ldots z_{t-2}, z_{t-1}, z_t, z_{t+1}, \ldots$$

donde z_1 representa el primer valor de la serie, z_2 el segundo, y z_t será el valor actual de la serie. De este modo z_{t+1} representa el valor de la serie para el próximo periodo, es decir un valor futuro.

La función de autocorrelación tiene por objetivo como se influyen las diversas observaciones.

3.2.2 Tendencias

Si analizamos las tendencias de una serie temporal podemos identificar dos:

- **Mean reversión**: Momento en el que el valor alcanza un pico y nos indica que el patrón siguiente es una caída. Tiene autocorrelación negativa.

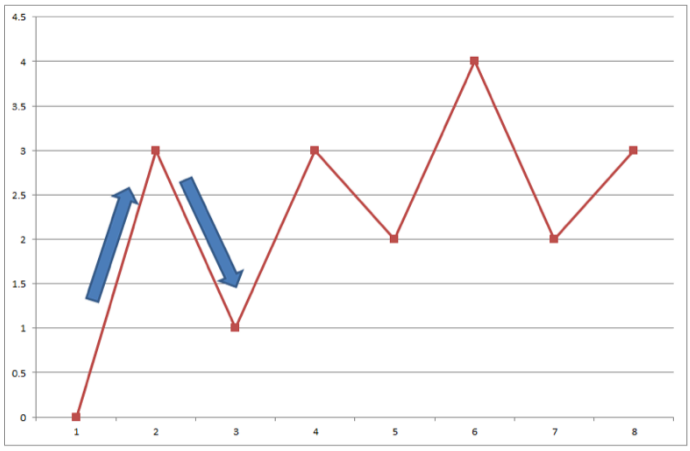

Figura - 8. Mean Reversion.

- **Momentum:** Nos indica que durante un periodo de tiempo el valor continua manteniendo esta ventaja a lo largo del tiempo. Tiene autocorrelación positiva.

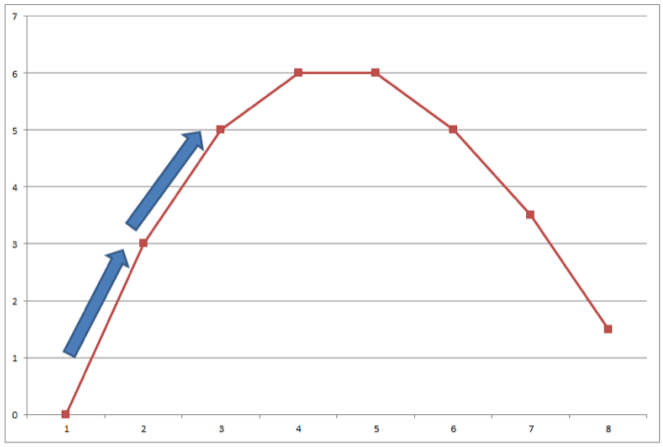

Figura - 9. Momentum.

A continuación, las principales características entre el mercado individual y los productos básicos y monedas:

Individual stocks

- Historicamente tiene autocorrelacion negativa.
- Medidas sobre un horizonte corto (días).
- Es un buen momento para vender.

Productos básicos y monedas

- Historicamente tiene autocorrelacion positiva.
- Medidas sobre horizontes largos (meses).
- Es un buen momento para comprar.

Un aspecto importante a tener en cuenta en las series temporales es el ruido blanco, que se verá en el siguiente apartado.

3.2.3 Ruido blanco

El ruido blanco es una serie con:

- Media constante.
- Varianza constante.
- Autocorrelación cero en todos los retrasos.

Existe un caso especial en el que si la distribución es normal sería ruido blanco Gaussiano.

Una forma de generar ruido blanco es con la librería numpy y en concreto con `random.normal` Donde los parámetros serían `loc` que sería la media y `scale` que es la desviación típica. [13]

3.2.4 Random Walk

Básicamente se define como que el precio de hoy es igual al de ayer más un ruido. El cambio en el precio en Random Walk es simplemente ruido blanco:

$$P_t = P_{t-1} + \epsilon_t \qquad (3.9)$$

$$\epsilon_t = P_t - P_{t-1} \qquad (3.10)$$

El balance final es que si el precio de las acciones sigue un Random Walk, entonces los valores de las acciones son ruido blanco. Una de las propiedades del Random Walk es que no se puede predecir, es decir, la mejor predicción para el precio mañana es el de hoy.

En un Random Walk con derivas la diferencia en el precio sigue siendo ruido blanco pero con una media μ:

$$P_t = \mu + P_{t-1} + \epsilon_t \qquad (3.11)$$

$$P_t - P_{t-1} = \mu + \epsilon_t \qquad (3.12)$$

Para comprobar si una serie sigue un Random Walk, podemos hacer una regresión con precios actuales y precios con retraso temporal:

$$P_t = \alpha + \beta P_{t-1} + \epsilon_t \qquad (3.13)$$

Comprobaremos el valor de β y tendremos dos hipótesis:

$$H_0: \beta = 1\ (random\ walk)$$
$$H_1: \beta < 1\ (no\ random\ walk)$$

Si el valor de β no es diferente de 1, no podemos rechazar la hipótesis nula por lo que sería un random walk, por el contrario, si β es distinto de 1, rechazaríamos la hipótesis nula y no sería un random walk.

Una forma idéntica de hacerlo sería haciendo la regresión en la diferencia del precio. Ahora tendríamos una regresión idéntica donde el eje de coordenadas sería la diferencia de precios y donde las hipótesis en lugar de ser 1 serían 0:

$$P_t - P_{t-1} = \alpha + \beta P_{t-1} + \epsilon_t \qquad (3.14)$$

$$H_0: \beta = 0\ (random\ walk)$$
$$H_1: \beta < 0\ (no\ random\ walk)$$

Esta comprobación es conocidas como **Dickey-Fuller Test.** Para realizarla, statsmodel tiene una función llamada `adfuller.` [13]

Si lo realizamos nos tenemos que fijar en el p-valor:

- **p-valor < 5%:** Entonces podemos rechazar la hipótesis nula con un intervalo de confianza del 95%.
- **En otro caso**: No podemos rechazar la hipótesis nula.

3.2.5 Estacionariedad

Existen dos tipos: fuerte y débil. La estacionariedad fuerte en una distribución se define como aquella en la que los datos son invariantes temporalmente, es decir, las observaciones no dependen del tiempo. En cambio, la estacionariedad débil es menos restrictiva y establece que la media, varianza y autocorrelación no son dependientes del tiempo (para la autocorrelación, $corr(X_t, X_{t-\tau})$ es solo una función de τ y no del tiempo).

Si un proceso no es estacionario, entonces es difícil de modelar. La modelización envuelve la estimación de un conjunto de parámetros, y si un proceso no es estacionario y los parámetros son diferentes en cada punto en el tiempo, entonces hay muchos parámetros a estimar.

Random Walk es un ejemplo de serie no estacionaria, la varianza crece con el tiempo, pero si cogemos las primeras diferencias la convertiremos en una serie estacionaria ya que será ruido blanco.

En ocasiones necesitamos dos pasos para transformar una serie en estacionaria, si tuviéramos una serie con un patrón exponencial, primero necesitaríamos aplicarl el logaritmo para deshacernos de este comportamiento y una vez realizado esto haríamos la diferencia. [13]

3.2.6 Modelos Autoregresivos (AR)

En los modelos autoregresivos, el valor de hoy equivale a una media más una fracción phi del valor de ayer más ruido:

$$P_t = \mu + \phi P_{t-1} + \epsilon_t \qquad (3.15)$$

Ya que solo hay un valor de retraso temporal en el lado derecho, esto es llamado como *modelo autoregresivo de orden 1* ó *modelo AR(1)*. [13]

Si el parámetro AR (ϕ) es 1, entonces el proceso es un Random Walk. En cambio, si ϕ es 0 entonces es ruido blanco. Para que el proceso sea estable y estacionario, entonces el valor tendrá que estar entre -1 y +1, en este caso tendríamos dos casos:

- $\phi < 0$: Mean Reversion
- $\phi > 0$: Momentum

Podríamos tener modelos autoregresivos de distinto orden:

- *AR(1):* $P_t = \mu + \phi P_{t-1} + \epsilon_t$
- *AR(2) :* $P_t = \mu + \phi_1 P_{t-1} + \phi_2 P_{t-2} + \epsilon_t$
- …

3.2.7 Modelos de medias móviles (MA)

En los modelos de media móviles, el valor de hoy equivale a una media más ruido más una fracción theta del ruido del valor de ayer:

$$R_t = \mu + \epsilon_t + \theta\epsilon_{t-1} \qquad (3.16)$$

Ya que solo hay un valor de retraso temporal en el lado derecho, esto es llamado como *modelo de medias móviles de orden 1* ó *modelo MA(1)*.

Si θ es 0 entonces estamos ante ruido blanco. Los modelos MA son estacionarios para todos los valores de θ. En este caso tenemos dos casos:

- $\theta < 0$: Mean Reversion en un periodo.
- $\theta > 0$: Momentum en un periodo.

La autocorrelación en un periodo no es θ, es:

$$\frac{\theta}{(1+\theta)^2} \qquad (3.17)$$

Como en los modelos autoregresivos, podemos encontrarnos con modelos MA de distinto orden:

- *MA(1)* : $R_t = \mu + \epsilon_t - \theta_1\epsilon_{t-1}$
- *MA(2)* : $R_t = \mu + \epsilon_t - \theta_1\epsilon_{t-1} - \theta_2\epsilon_{t-2}$
- …

3.2.8 Modelos autoregresivos de media móvil (ARMA)

Los modelos ARMA son modelos que combinan modelos autoregresivos y modelos de medias móviles. Su fórmula es la siguiente:

$$R_t = \mu + \phi R_{t-1} + \epsilon_t + \theta\epsilon_{t-1} \qquad (3.18)$$

La cual es una combinación de los dos modelos (AR y MA). Podemos convertir un modelo ARMA en un AR puro o un MA puro. [13]

3.2.9 Redes neuronales recurrentes (RNN)

En un problema con series temporales las redes neuronales más usadas son las recurrentes (RNN) y en concreto las LSTM (Long-Short Term Memory) ya que las neuronas tienen memoria y en este caso necesitamos saber el estado anterior de la neurona para poder predecir el siguiente valor, los factores más importantes de una LSTM son los siguientes: [14]

- c_t, **estado de célula**: Representa la memoria interna de la célula en la que se almacena el término de corto plazo como el de largo plazo.
- h_t, **estado oculto**: Esta es la información de estado de salida calculada con el valor actual de la entrada, estados ocultos previos y la entrada actual de la célula usada eventualmente para predecir el valor. El estado oculto puede decidir si solo quedarse con el valor a corto plazo, el de largo plazo o ambos.
- i_t, **puerta de entrada:** Decide cuanta información de la entrada actual fluye hacia el estado de la célula.
- f_t, **estado de olvido:** Decide cuanta información de la entrada actual y del estado previo de la célula fluye hacia el estado actual de la célula.
- o_t, **puerta de salida:** Decide cuanta información del estado actual de la célula fluye hacia el estado oculto.

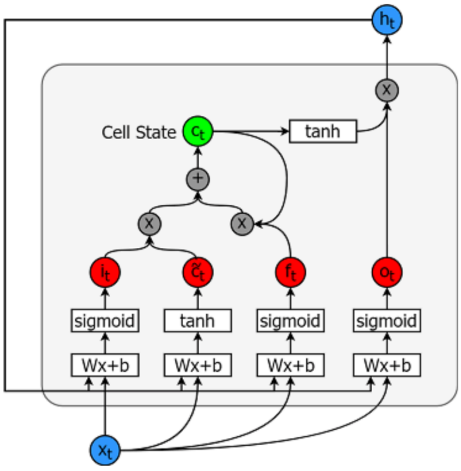

Figura - 10. LSTM.

3.2.10 MinMax scaling (Normalización)

El MinMax scaling es un método usado para la normalización de los datos que no es más que procesar dichos datos para obtenerlos en un intervalo entre 0 y 1. Esto se realiza ya que ayuda al modelo predictivo a la hora de entrenar

Esta normalización se hace mediante la siguiente ecuación:

$$Xsc = \frac{X - Xmin}{Xmax - Xmin} \qquad (3.19)$$

4 SOFTWARE Y HARDWARE EMPLEADO

"Computers are useless. They can only give you answers".

- Pablo Picasso -

Los modelos de predicción en los que nos basamos necesitan de una fuerza computacional robusta por lo que el desarrollo de hardware consistente ha ido avanzando para poder satisfacer este tipo de integración. Además del desarrollo hardware, el software se ha sometido a cambios y lanzamientos de nuevas librerías capaces de cubrir estas necesidades. El carácter *open-source* de Python ha hecho que se desarrollen múltiples fuentes de librerías capaces de simular todo tipo de modelos.

4.1 Raspberry pi 3

Raspberry pi 3 es un dispositivo de tamaño pequeño que consta de una placa en la que se monta un procesador, chip gráfico y memoria RAM. Lanzado en 2006 por la fundación Raspberry Pi. [8]

A continuación, podemos identificar en la siguiente imagen sus componenetes principales:

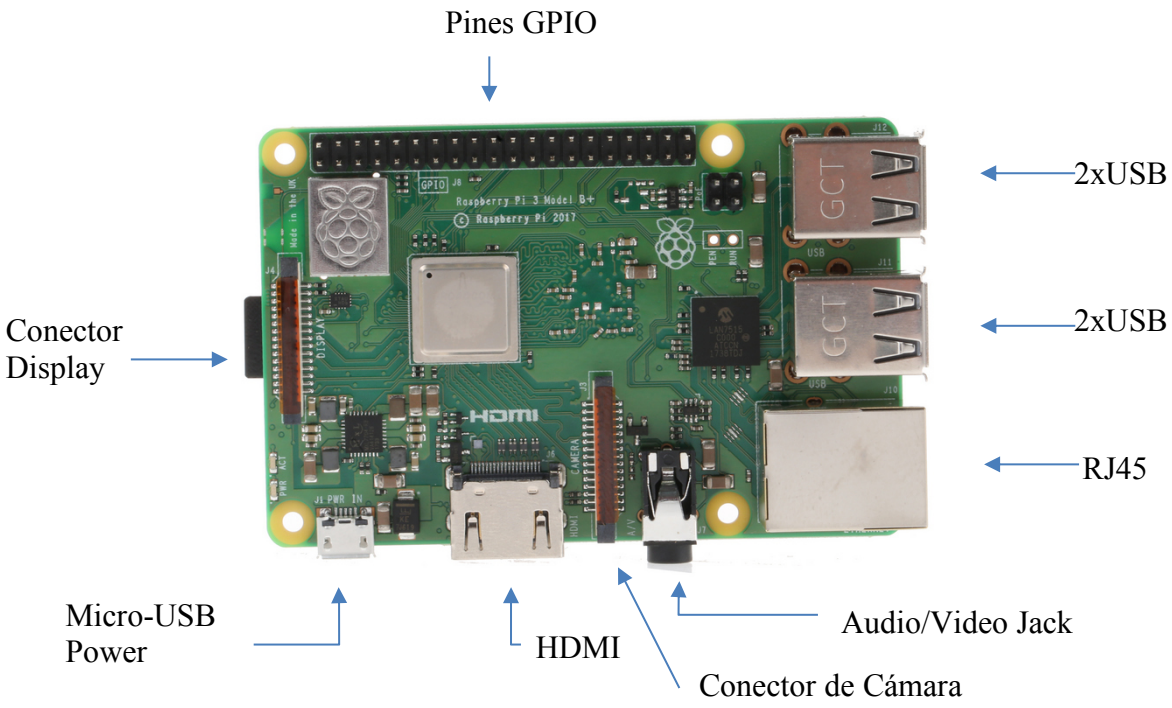

Figura - 11. Arquitectura Raspberry Pi 3.

Es una múltiple fuente de proyectos debido a su potencia de cómputo (1 Gb de RAM), además es modular ya que se le puede proporcionar diferentes funcionalidades como el uso de la Pi Camera o conexión de arduinos.

Entre sus principales características:

- Procesador Broadcom BCM2837, Cortex A53(ARMv8) 64-bit SoC.
- Disipador capaz de ayudar a controlar las temperaturas generadas por la CPU.
- Frecuencia de reloj de 1.2GHz.
- Conectividad inalámbrica 2.4GHz IEEE 202.11b/g/n Bluetooth 4.1.
- Fast Ethernet 10/100 Gbps.
- GPIO 40 pines, HDMI, 4xUSB 2.0, MicroSD, Micro USB.

4.2 Raspbian

Es una distribución de GNU/Linux basado en Debian y optimizado para el hardware de la Raspberry Pi, su lanzamiento fue el 12 de junio de 2012.

La distribución usa LXDE como escritorio y Midori como navegador web. Destaca el menú *raspi-config* que permite configurar el sistema operativo sin tener que cambiar los ficheros manualmente. [9]

Además, incluye herramientas de desarrollo muy interesantes, como IDE[14] para Python, Scratch para programar videojuegos la tienda de aplicaciones denominada Pi Store, etc…

4.3 Python

Python es un lenguaje de programación Creado por Guido Van Rossum en 1991. Se trata de un lenguaje interpretado por lo que los errores se dan en tiempo de ejecución.

Es un lenguaje multiplataforma (Windows, Linux, Mac). Las distribuciones de Linux suelen venir con el intérprete de Python ya incorporado. Administrado por la Python Software Foundation. Posee una licencia de código abierto, denominada como Python Software Foundation License.

Está ganando mucha importancia los últimos años debido a las múltiples librerías diseñadas basadas en Machine Learning, Big Data, Inteligencia Artificial y diseño web.

Trabajaremos con Python 3.6.

4.4 IDE: Spyder

Es un potente entorno de desarrollo interactivo Open-Source para programación en Python. Cuenta con funciones avanzadas de edición, pruebas interactivas, depuración e introspección y un entorno informático numérico.

[14] IDE, *Integrated Development Environment,* Aplicación que proporciona servicios integrales para facilitar la programación.

También se puede usar como una biblioteca que proporciona widgets potentes relacionados con la consola. [10]

Se liberó bajo la licencia del MIT[15]. Entre sus características más importantes están:

- El editor que integra es multilenguaje.
- Consola interactiva.
- Visor de documentación.
- Exploración de variables.
- Dispone de un explorador de archivos.

Es un IDE muy utilizado debido a su fácil integración y su interfaz intuitiva.

4.5 Librerías

4.5.1 TensorFlow

Es una librería de código abierto para el cálculo numérico de alto rendimiento. Se creó para diseñar, construir y entrenar modelos de Deep Learning.

Su arquitectura permite una fácil implementación de computación en una variedad de plataformas (CPU, GPU, TPU) y desde escritorios hasta clústeres de servidores y dispositivos móviles. Desarrollado por Google Brain dentro de la organización AI de Google, cuenta con un sólido respaldo para Machine Learning y Deep Learning. [11]

Además, nos permite una vez creados nuestro modelo, guardarlo y desplegarlo en otro dispositivo.

4.5.1.1 Instalación

Debemos tener instalado pip y Virtualen:

```
sudo apt-get install python-pip python-dev python-virtualenv
```

una vez instalados, debemos crear un *virtual environment* y activarlo usando los siguientes comandos:

```
virtualenv --system-site-packages ~/tensorflow
source ~/tensorflow/bin/activate
```

cuando realicemos estos dos comandos nos aparecerá a principio de línea `(tensorflow)` esto nos indicará el nombre del virtual environment en el que estamos trabajando.

[15] MIT, *Massachusetts Institute of Technology*.

A continuación, procederemos a instalar tensorflow a través del siguiente comando:

```
(tensorflow)$ pip3 install --upgrade tensorflow
```

Con esto ya dispondremos de Tensorflow en nuestra máquina.

4.5.1.2 Tensores

Para entender qué es un tensor es fundamental tener conocimientos de álgebra lineal y cálculo vectorial.

Un tensor se define como un array multidimensional, por ejemplo, un vector no es más que un Tensor de rango 1. De la misma manera, un escalar es un tensor de rango 0 ya que los escalares no tienen indicador direccional.

En un tensor de rango 2 en lugar de asociar un número para cada base vectorial, asociamos un número con todas las posibles combinaciones de dos bases vectoriales. Lo mismo para un tensor de rango 3.

4.5.2 Alpha-vantage

Alpha-vantage es un API capaz de recoger valores históricos de la bolsa y en tiempo real en diferentes periodos temporales desde datos anuales, mensuales, diarios y en tiempo real cada 5 minutos. Necesitaremos una key para poder usar la API, para obtenerla es necesario rellenar un formulario en la página oficial con el nombre, apellidos y correo electrónico.

En este caso necesitamos los valores del Bitcoin, tanto históricos como en tiempo real. En primer lugar, necesitamos crearnos un objeto de tipo `CryptoCurrencies` para poder tener acceso a los datos, una vez realizado esto podemos obtener los históricos con el método `get_digital_currency_daily` y para obtener datos en tiempo real el método `get_digital_currency_intraday` que nos ofrece los datos en tiempo real cada 5 minutos.

Para la instalación se necesitará ejecutar el siguiente comando:

```
sudo pip install alpha_vantage
```

5 RESULTADOS

"Insanity is doing the same thing over and over again and expecting different results".

- Albert Einstein -

La integración de sistemas de aprendizaje automático en Raspberry pi no está exenta de capacidad de cómputo ya que como hemos visto, Raspberry pi es muy potente pero la necesidad de recursos de Python y la librería TensorFlow es considerable. En este capitulo desarrollaremos dos aplicaciones integradas en este dispositivo: Reconocimiento de imágenes y datos financieros en streaming.

Comprobaremos la eficiencia de la Raspberry Pi ante estas dos aplicaciones y se abordará un marco teórico de ambas.

5.1 Reconocimiento de imágenes

5.1.1 Metodología

El reconocimiento de imágenes es un proceso complejo en el cuál un sistema debe clasificar una imagen entre un conjunto de clases con la mayor precisión posible.

Para esta aplicación usaremos como conjunto de datos CIFAR-10 [16], este dataset es uno de los más conocidos y se constituye de 50.000 imágenes para training y 10.000 imágenes para evaluar el modelo. El tamaño de cada imagen es de 32x32x32 (RGB) y contiene 10 clases diferentes: avión, automóvil, pájaro, gato, ciervo, perro, rana, caballo, barco y camión.

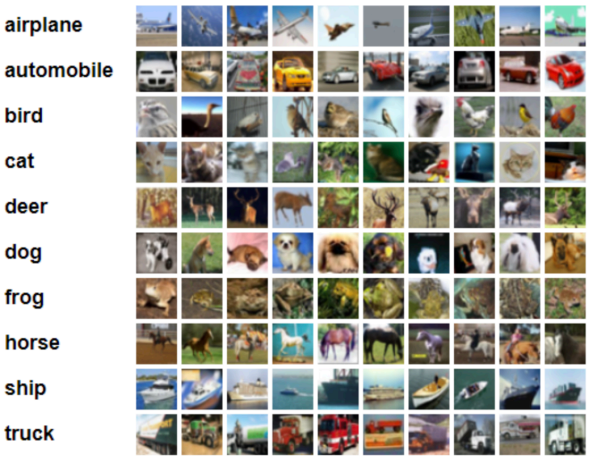

Figura - 12. Ejemplos dataset CIFAR-10.

5.1.1.1 Clasificador Logístico

El clasificador logístico no es más que una función que recibe unas entradas que en este caso son los píxeles de nuestras imágenes y aplica una función lineal para realizar las predicciones:

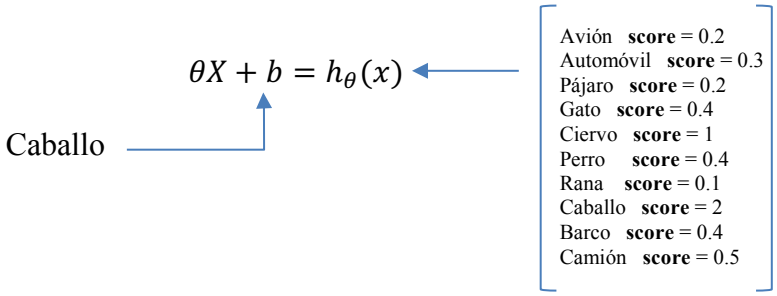

Figura - 13. Función lineal con Scores.

Una función lineal solo es una matriz de multiplicación, multiplica el vector de entradas *X* por una matriz de pesos o *weigths W* para generar las predicciones. El valor *b* es considerado como el bias y es un parámetro de ajuste para nuestra función al igual que *W* ya que en esto consistirá el Machine Learning, adaptando en cada aprendizaje estos valores para generar mejores predicciones.

Cada imagen solo tiene 1 posibilidad entre las 10 posibles, por lo tanto, a cada clase le corresponderá una puntuación o *score* de que la entrada sea dicha clase. Convertiremos el score en probabilidades para cada clase que nos dirá la probabilidad de que esa clase sea la correcta. La forma para convertir el score en probabilidades es usando la función **Softmax**. [12]

5.1.1.2 Softmax

La función Softmax es capaz de convertir scores para clases en probabilidades, se denota de la siguiente forma:

$$S(y_i) = \frac{e^{y_i}}{\sum_j e^{y_j}} \qquad (5.1)$$

En la fórmula 5.1 vemos la función Softmax donde y_i se corresponde a la puntuación para la clase i.

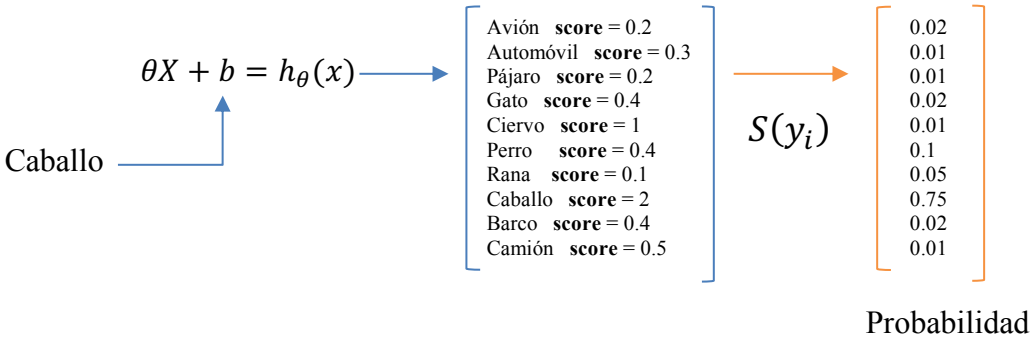

Figura - 14. Aplicación función Softmax.

Como podemos comprobar si sumamos todas las probabilidades nos daría 1. Scores también son llamados como *logits.*

Una vez realizado esto debemos medir lo bien que predice nuestro modelo, para realizar esto necesitamos una función conocida como **Cross-Entropy** que se encarga de medir el error cometido entre dos vectores, el de probabilidades generado por la función Softmax y el vector de labels que nos proporciona el dataset aplicando un proceso llamado One-Hot Encoding que se encarga de para clase asignarle un 1 en la posición del vector que se corresponda con cada clase. [12]

5.1.1.3 Cross-Entropy

La función mide el error de dichos vectores, S es el de probabilidades y L es el One-Hot Encoding:

$$D(S, L) = -\sum_i L_i \log(S_i) \qquad (5.2)$$

Nuestro problema de optimización reside en esta función ya que necesitamos minimizar D para clases correctas y maximizarla para clases incorrectas. Para medir el valor medio de esta distancia podemos calcular la media sobre todas las muestras de nuestro conjunto de datos de entrenamiento, a esto se le llama pérdida de entrenamiento o training loss. [12]

5.1.1.4 Loss

No es más que la media de la Cross-Entropy para todas las muestras de entrenamiento y es la siguiente:

$$\mathcal{L} = \frac{1}{N}\sum_i D(S(wx_i + b), L_i) \qquad (5.3)$$

Siendo N el número de muestras.

Queremos que el Loss sea pequeño para poder tener una precisión alta en nuestro modelo, es una función de pesos y bias por lo que minimizaremos dicha función. Para realizar dicha minimización podemos basarnos en la metaheurística y usar el algoritmo de gradiente descendente, de esta forma conseguiremos encontrar los valores de pesos y bias para minimizar nuestra función y generar mejores predicciones, pero usaremos una variante de esta denominada gradiente descendente estocástico. [12]

5.1.1.5 Redes neuronales

Una vez que hemos entrenado un clasificador logístico integraremos esta intuición en una red neuronal para conseguir mejores resultados.

Necesitamos una función no lineal para rectificar unidades lineales, la más conocida es RELU[16] (REctified Linear Units) conseguirán que sea > 0 si x es > 0 y si x < 0 será 0:

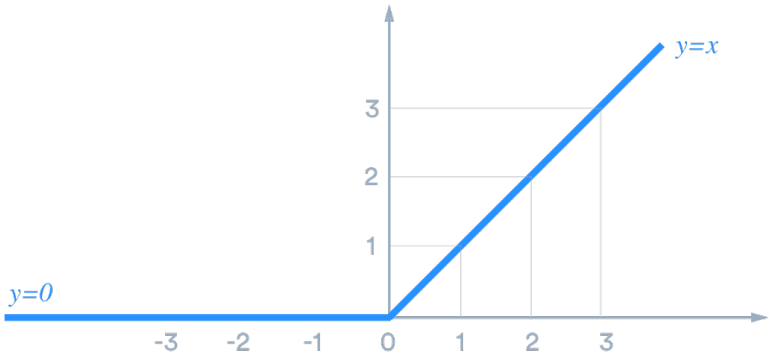

Figura - 15. Rectified Linear Units.

Lo que haremos será construir una nueva función no lineal, en lugar de tener solo una matriz insertaremos RELU en medio y tendremos dos matrices, una irá desde las entradas hasta RELU y otra desde RELU hasta el clasificador:

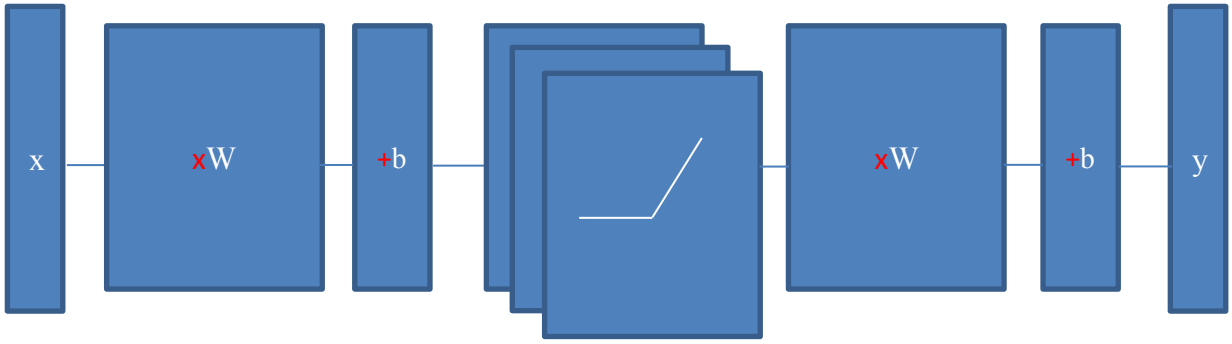

Figura - 16. Modelo de 2 capas.

[16] RELU, *Rectified Linear Units*, Función de activación.

A continuación, necesitamos ajustar los parámetros de nuestro modelo, para eso usamos los algoritmos diseñados llamados Forward-propagation y Back-propagation. Ambos recorren las redes neuronales en sentidos inversos realizando derivadas parciales.

Para cada batch de datos en nuestro conjunto de datos de entrenamiento ejecutaremos el Forward-propagation y luego el Back-propagation y esto nos dará el gradiente descendente para cada uno de nuestros pesos o weigths del modelo. [12]

5.1.1.5.1 Covnets (Redes neuronales convolucionales)

Son redes neuronales que comparten sus parámetros sobre el espacio. La idea general es que formarán una pirámide, empezaremos en la base con una imagen grande (32x32x3), aplicaremos convoluciones que irán reduciendo las dimensiones progresivamente en las imágenes mientras que la profundidad irá aumentando. En la cima, podemos poner nuestro clasificador.

Los conceptos fundamentales son los siguientes:

- **Profundidad**: En nuestro caso, al ser las imágenes RGB, será de 3.
- **Kernel**: Será la ventana o filtro con la que iremos mapeando la imagen.
- **Pasos**: Es la cantidad de pasos o pixeles que moveremos el Kernel mientras mapeamos la imagen. Un paso de 1 equivaldría a un tamaño de la misma imagen, sin embargo, un paso de 2 equivaldría a aproximadamente la mitad de la imagen.

El modelo seguido tiene como entrada un batch de 32x32x3 que representa cada imagen, a continuación, se le aplicará un *pooling,* en concreto maxpooling, que lo que hace es reducir las dimensiones de las entradas recogiendo valores importantes como la media, máximos, mínimos…

También aplicaremos en varias capas *dropout* que se encarga de regularizar el modelo desactivando algunas neuronas en función de una probabilidad. Esta técnica ayuda a que las unidades no se acoplen entre ellas.

A continuación, se muestra el esquema de la red neuronal convolucional realizada:

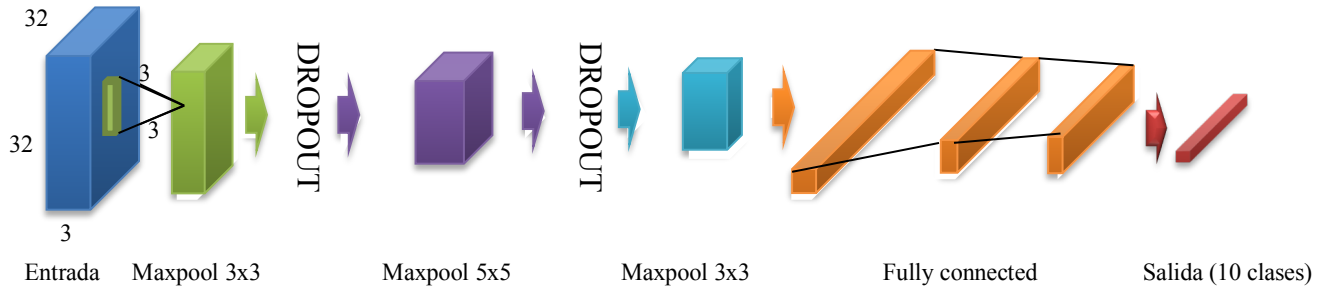

Figura - 17. CNN de reconocimiento de imágenes.

5.1.2 Resultados experimentales

Una vez implementada la red neuronal en la Raspberry pi, usaremos Pi Camera para hacer una foto a cualquier elemento de las clases que ofrece CIFAR-10. Una vez realizada la foto se convertirá a 32x32x3 y este elemento será nuestro input de la red neuronal. La red neuronal se encargará de identificar qué elemento es y de mostrarlo por la pantalla.

En el caso del clasificador logístico, la entrada no es la imagen con un shape de 32x32x3, necesitamos hacerle un reshape y dejar todos los datos en un vector fila de tamaño 3072 = 32 x 32 x 3. Una vez hecho esto, escalaremos los datos para que entrenar al clasificador en menos tiempo y guardaremos el modelo para implementarlo en el dispositivo.

A continuación, se muestra una tabla en la cuál se puede ver claramente como la red convolucional es más lenta que el clasificador logístico. Sin embargo, su precisión es mayor aunque no sea excesiva. Esto se debe a que cuando hemos entrenado nuestros modelos hemos usado el dataset con imágenes limpias (sin fondos aleatorios) y cuando hacemos una foto con la Pi Camera no se asimila por completo a los datos que le pasamos en entrenamiento, si le pasamos datos del dataset enfocado al test se puede comprobar que el desempeño en cuánto a precisión mejora en un 10%-20%. Se ha usado el algoritmo de gradiente descendente y 30 *epochs* en el entrenamiento de ambos modelos.

Tabla 2. Desempeño de Raspberry Pi con reconocimiento de imágenes

Algoritmo	**Tiempo de respuesta (segundos)**	**Precisión(Pi camera)**	**Precisión(Test Set)**
Red Convolucional (CNN)	0.4	63%	74%
Clasificador Logístico	0.005	50%	62%

La forma en la que se carga el modelo, se capturan las imágenes y se le pasa tanto a la red neuronal como al clasificador logístico se puede encontrar en el **Anexo: Código de ejemplos**.

El entrenamiento de dichos modelos se ha realizado en un ordenador, para el tipo de red convolucional descrito y el clasificador logístico se realizó con un tiempo de 2 horas, razón por la cuál no se han aplicado más capas.

5.2 Datos financieros en streaming

5.2.1 Metodología

La metodología seguida para el análisis de datos financieros se basa en el estudio de las series temporales y de las características principales a analizar. Además, se usarán redes neuronales recurrentes para el análisis en tiempo real.

5.2.2 Resultados experimentales

Se estudiará como actúa la Raspberry pi con 5 diferentes modelos cada uno de ellos con diferentes capas (desde 1 capa hasta 5 capas) y se medirá el error cuadrático medio de la predicción (MSE)[17] en tiempo real y el tiempo que tarda en dar dicha predicción. Además, se comparará con un modelo ARMA[18]. Para entrenar la red neuronal se ha usado el histórico del Bitcoin desde 2014 con entradas de datos de 20 en 20, además, normalizaremos los precios con MinMax para facilitar el entrenamiento a la red neuronal.

5.2.2.1 Redes neuronales multicapa

Se ha usado la API `alpha-vantage` para poder recibir en streaming el precio del Bitcoin (BTC)[19] cada 5 minutos, cada vez que llega un nuevo dato se prepara un conjunto de 20 puntos que será la entrada a la red neuronal, se normalizará y se realizará una predicción que se contrastará con el nuevo dato a los 5 minutos.

En la siguiente tabla podemos identificar las diferentes medidas tomadas desde la Raspberry pi al probar los distintos modelos:

Tabla 3. Desempeño Raspberry Pi datos financieros en streaming.

Capas	Tiempo de respuesta (segundos)	Error cuadrático medio (MSE)
1	0.2282	$1.4^{10^{-5}}$
2	0.3820	$4.74^{10^{-6}}$
3	0.4896	$4.6^{10^{-7}}$
4	0.6121	$4.9^{10^{-9}}$
5	0.7149	$6.66^{10^{-8}}$

Como se puede comprobar en la tabla, los tiempos de respuesta dados por la Raspberry son buenos sabiendo que el dato nuevo se obtiene cada 5 minutos, si vemos el desempeño del modelo también es bueno ya que los errores entre las predicciones y el valor real son del orden de 10^{-8} aproximadamente. Sin embargo, el decremento del error cuadrático es directamente proporcional al número de capas de nuestro modelo excepto con 5 capas donde el error es más grande que con 4 capas. Esto es debido al overfitting o sobreajuste, que no es más que un ajuste excesivo a los datos de entrenamiento y que perjudica a la hora de hacer predicciones con nuevos datos que no estén en el conjunto de entrenamiento.

[17] MSE, Mean Squared Error, Error cuadrático medio medido con el cuadrado de las diferencias entre el valor predicho y el valor real.
[18] ARMA, Modelo Autoregresivo de media móvil.
[19] BTC, Símbolo del Bitcoin.

A continuación, en las siguientes figuras se muestran gráficas de las predicciones (naranja) y de los valores reales (azul) para cada 20 puntos teniendo en cuenta que el último es siempre el último dato recibido a los 5 minutos:

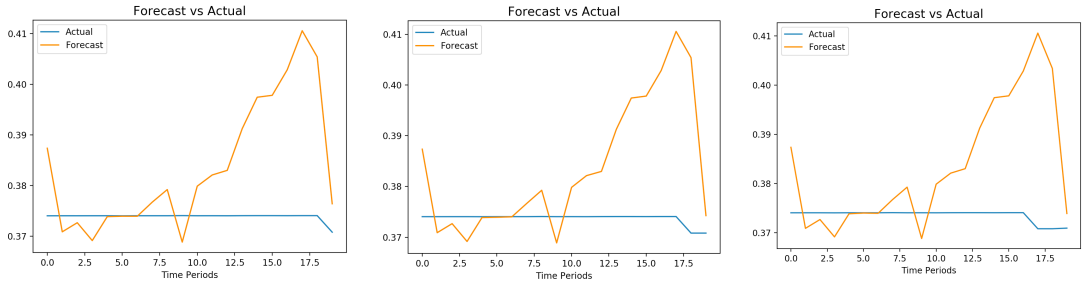

Figura - 18. Modelo de 2 capas

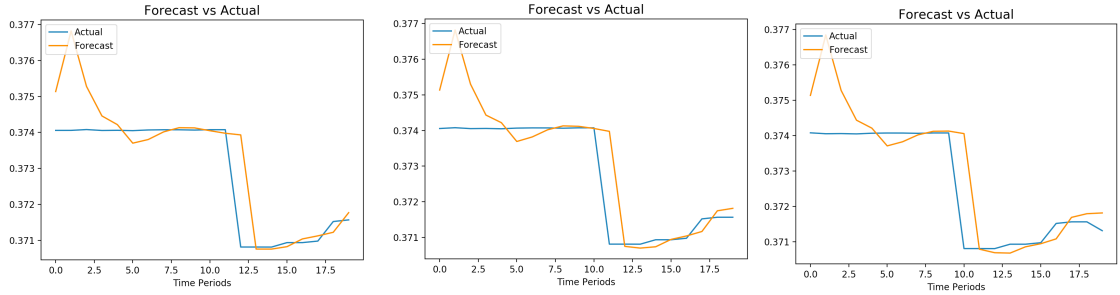

Figura -19. Modelo de 4 capas.

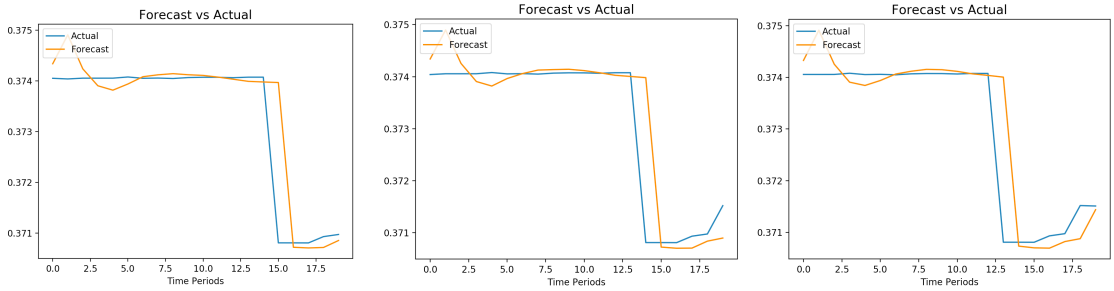

Figura - 20. Modelo de 5 capas.

Como se puede observar el desempeño de los distintos modelos va mejorando con respecto al número de capas.

A continuación, un ejemplo de ejecución del programa:

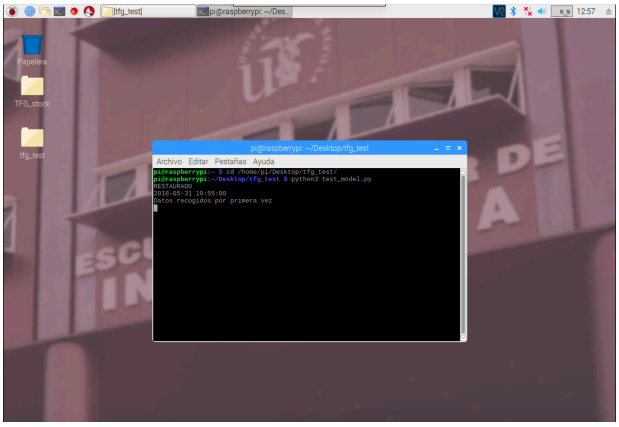

Figura - 21. Ejemplo de ejecución de programa(1/2)

En la figura 21 se contempla el inicio del programa en el que se recogen los datos por primera vez en este caso el dato de las 10:55 (Aunque el horario español sea 12:57 hay 2 horas de retraso debido al mercado seleccionado), a partir de aquí empezará a sondear hasta recoger el dato de las 11:00 (13:00 hora española) en el que hará la primera predicción.

El entrenamiento de la red neuronal recurrente se realizó en un ordenador y se desarrolló en 2 horas y media. Por eso, no se ha entrenado el modelo con más capas.

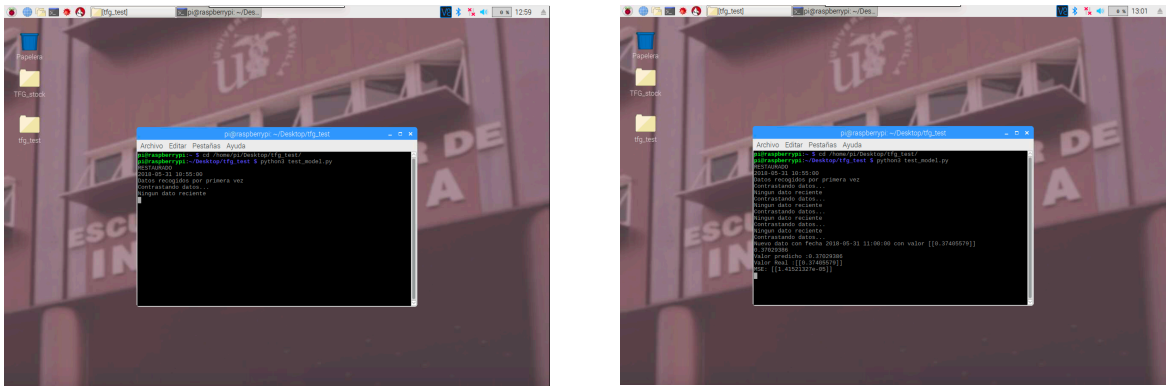

Figura - 22. Ejemplo de ejecución de programa (2/2)

Una vez realizado esto, seguirá sondeando hasta que le llegue el valor de las 11:05 (13:05 hora española) y hará una nueva predicción, y asi sucesivamente cada 5 minutos.

5.2.2.2 Modelo ARMA

La segunda forma de predicción se ha hecho usando un modelo ARMA (Autoregresivo de media movil) para comparar asi su desempeño y la precisión haciendo un uso más teórico de las series temporales.

Para ello usaremos las librerías `statsmodel` que nos proporciona clases para la creación del modelo y la librería `alpha_vantage` con el mismo fin que en el apartado anterior.

En este caso, no necesitamos entrenar el modelo con valores normalizados ya que el gasto computacional y la optimización del entrenamiento no son necesarias.

El error cuadrático medio es de 0.02 por lo que vemos que no es un mal modelo a la hora de predecir valores, ya que nos estamos equivocando en 0.02 unidades pero no ofrece el rendimiento que nos da una red neuronal.

Además, la Raspberry tarda 0.07 segundos en dar la predicción por lo que en tiempos mejora con creces a la red neuronal.

6 CONCLUSIONES Y LÍNEAS FUTURAS

"Machines will be able to do anything made up by humans because humans are machines".

- Marvin Minsky -

El uso de dispositivos que integran modelos de predicción y de reconocimiento nos lleva a un paradigma muy conocido como el de la Inteligencia Artificial. La forma en la que actuamos los humanos es estudiada para algún día poder ser simulada por máquinas (en este caso robots) los cuáles ya se están desarrollando y algunos ya probados como los de la empresa Boston Dynamics.

La posibilidad de usar este tipo de algoritmia en la Raspberry pi nos indica que muchos dispositivos con similar fuerza de cómputo serán capaces de integrar estos modelos ya sea en local o en la nube. En este caso se han estudiado las aplicaciones de reconocimiento de imágenes y de datos en streaming, pero se podrían utilizar otras como el procesamiento de lenguaje natural y el reconocimiento de voz.

El uso de redes neuronales recurrentes (RNN) tiene varios casos de uso desde la generación de la descripción de texto a partir de una imagen, aplicaciones de mejoras del filtro de Kalman para técnicas de GPS y un gran potencial para procesamiento de video y detección de objetos. En este proyecto el entrenamiento se realizó en un ordenador y esto limitó el número de capas que se podrían haber desarrollado. Si se contase con un ordenador más potente o GPUs podríamos dotar al modelo de más capas.

Otra parte fundamental es la capacidad de toma de decisiones en tiempo real. En este caso hemos usado datos financieros pero el continuo tráfico que pasa por una red es un posible objeto de estudio. Además, otra posible aplicación sería la clasificación de transacciones bancarias que se realizan al día en sistemas integrados con Spark.

Actualmente existen CNN con muchas más capas de las desarrolladas en este proyecto por lo que es un objeto de investigación el desempeño de estos dispositivos con distintas redes neuronales y múltiples capas. Además, dotar de una inteligencia local capaz de entrenar modelos y aplicarlos sin necesidad de un dispositivo o capacidad de cómputo externa también entra en el campo de la investigación que actualmente se está mejorando.

En conclusión, durante la realización de este trabajo, se han llevado a la práctica numerosos aspectos teórico-prácticos de sistemas y modelos usados actualmente. Desde el punto de vista

personal, esto ha ayudado a ampliar mis conocimientos en un campo nuevo como el del aprendizaje automático.

7 REFERENCIAS

[1] Autor, «Applealianos,» 2017. [En línea]. Available: http://www.applelianos.com/2017/01/27/google-llevara-la-ia-la-raspberry-pi/.

[2] O. Autor, «Medium,» 22 12 2017. [En línea]. Available: https://medium.com/@Movetia/la-batalla-de-los-smartphones-con-machine-learning-15084cf58d9a.

[3] «El androide libre,» [En línea]. Available: https://elandroidelibre.elespanol.com/2017/05/google-goggles-aplicacion-sacrificada-google.html.

[4] Cleverdata, «Cleverdata,» [En línea]. Available: http://cleverdata.io/conceptos-basicos-machine-learning/.

[5] AndrewNG, «Coursera(Aprendizaje Automático),» [En línea]. Available: www.coursera.com.

[6] «Xataka,» [En línea]. Available: https://www.xataka.com/robotica-e-ia/deep-learning-que-es-y-por-que-va-a-ser-una-tecnologia-clave-en-el-futuro-de-la-inteligencia-artificial.

[7] «Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Perceptrón.

[8] «Xataka,» [En línea]. Available: https://www.xataka.com/ordenadores/raspberry-pi-3-model-b-analisis-mas-potencia-y-mejor-wifi-para-un-minipc-que-sigue-asombrando.

[9] «Raspbian,» [En línea]. Available: https://www.raspbian.org.

[10] «UbunLog,» [En línea]. Available: https://ubunlog.com/spyder-entorno-desarrollo-python/.

[11] «TensorFlow,» [En línea]. Available: https://www.tensorflow.org.

[12] «Deep Learning by Google,» [En línea]. Available: www.udacity.com.

[13] «Introduction to Time Series Analysis,» [En línea]. Available: www.datacamp.com.

[14] «Datacamp,» [En línea]. Available: https://www.datacamp.com/community/tutorials/lstm-python-stock-market.

ANEXO: CÓDIGO DE EJEMPLOS

Código test_model.py (Datos financieros en streaming)

```
import tensorflow as tf
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import time
from alpha_vantage.cryptocurrencies import CryptoCurrencies
from sklearn.preprocessing import MinMaxScaler



save_model_path = './5_layer/stock_prediction'

df = CryptoCurrencies(key='GSZK6WDG2I0YHQ3I', output_format='pandas')
data_historical_df, meta_data = df.get_digital_currency_daily(symbol='BTC',
market='CNY')
high_prices = data_historical_df.loc[:, '2a. high (CNY)'].as_matrix()
low_prices = data_historical_df.loc[:, '3a. low (CNY)'].as_matrix()
mid_prices = (high_prices+low_prices)/2.0
ts = np.array(mid_prices)

ts = ts.reshape(-1,1)

scaler = MinMaxScaler()
scaler.fit(ts[:])
ts = scaler.transform(ts[:])


num_periods = 20
f_horizon = 1



with tf.Session() as sess:
    # Load model
    saver = tf.train.import_meta_graph(save_model_path + '.meta')

    saver.restore(sess, save_model_path)



    loaded_graph = tf.get_default_graph()

    outputs = loaded_graph.get_tensor_by_name('Reshape_1:0')

    Placeholder = loaded_graph.get_tensor_by_name('Placeholder:0')


    print('RESTAURADO')

    #------------------------------------------------------------
```

```
bandera = 0
primera_vez = 1

while True:
    if bandera == 1:
        data_intra, _ = df.get_digital_currency_intraday('BTC', 'CNY')
        data_intra = data_intra[data_intra.index > start_time]
        print('Contrastando datos...')

    else:

        data_intra, _ = df.get_digital_currency_intraday('BTC', 'CNY')

    if bandera == 0:
        X_test = data_intra['1a. price (CNY)']
        datos = data_intra['1a. price (CNY)']
        X_test = X_test[(len(X_test)-21):-1]
        bandera = 1
        start_time = datos.index[-1]
        print(start_time)
        print('Datos recogidos por primera vez')
        X_test = np.array(X_test).reshape(-1, 1)
        X_test = scaler.transform(X_test)



    else:
        data_intra = data_intra['1a. price (CNY)']
        if data_intra.empty:
            print('Ningun dato reciente')
        else:

            if primera_vez == 0:
                print('Entra')
                X_test = Y_test
                print(X_test)
            if primera_vez:
                X_test = X_test.reshape(-1, 20, 1)
                print(X_test)

            feed_dict = {Placeholder: X_test}

            y_pred = sess.run(outputs, feed_dict)
            print('Nuevo dato')
            valor_nuevo = data_intra.values[0]
            valor_nuevo = scaler.transform(valor_nuevo)

            mi_lista = X_test.reshape(-1).tolist()
            mi_lista = mi_lista[1:]
            mi_lista.append(valor_nuevo)
            Y_test = np.array(mi_lista).reshape(-1, 20, 1)

            datos = datos.append(data_intra)
            start_time = datos.index[-1]
            print(start_time)
            print(np.ravel(y_pred)[-1])
            print(f'MSE: {(np.ravel(y_pred)[-1] - valor_nuevo)**2}')
```

```
bandera = 0
primera_vez = 1

while True:
    if bandera == 1:
        data_intra, _ = df.get_digital_currency_intraday('BTC', 'CNY')
        data_intra = data_intra[data_intra.index > start_time]
        print('Contrastando datos...')

    else:

        data_intra, _ = df.get_digital_currency_intraday('BTC', 'CNY')

    if bandera == 0:
        X_test = data_intra['1a. price (CNY)']
        datos = data_intra['1a. price (CNY)']
        X_test = X_test[(len(X_test)-21):-1]
        bandera = 1
        start_time = datos.index[-1]
        print(start_time)
        print('Datos recogidos por primera vez')
        X_test = np.array(X_test).reshape(-1, 1)
        X_test = scaler.transform(X_test)



    else:
        data_intra = data_intra['1a. price (CNY)']
        if data_intra.empty:
            print('Ningun dato reciente')
        else:

            if primera_vez == 0:
                print('Entra')
                X_test = Y_test
                print(X_test)
            if primera_vez:
                X_test = X_test.reshape(-1, 20, 1)
                print(X_test)

            feed_dict = {Placeholder: X_test}

            y_pred = sess.run(outputs, feed_dict)
            print('Nuevo dato')
            valor_nuevo = data_intra.values[0]
            valor_nuevo = scaler.transform(valor_nuevo)

            mi_lista = X_test.reshape(-1).tolist()
            mi_lista = mi_lista[1:]
            mi_lista.append(valor_nuevo)
            Y_test = np.array(mi_lista).reshape(-1, 20, 1)

            datos = datos.append(data_intra)
            start_time = datos.index[-1]
            print(start_time)
            print(np.ravel(y_pred)[-1])
            print(f'MSE: {(np.ravel(y_pred)[-1] - valor_nuevo)**2}')

            plt.title("Forecast vs Actual", fontsize=14)
            plt.plot(pd.Series(np.ravel(Y_test)), label="Actual")
            plt.plot(pd.Series(np.ravel(y_pred)), label="Forecast")
```

```
            plt.legend(loc="upper left")
            plt.xlabel("Time Periods")
            plt.show()
            time.sleep(1)
            plt.close()

            primera_vez = 0

    time.sleep(15)
```

Código test_arma.py (Datos financieros en streaming)

```
from alpha_vantage.cryptocurrencies import CryptoCurrencies
import pandas as pd
import time
from statsmodels.tsa.arima_model import ARMA

apikey = 'GSZK6WDG2I0YHQ3I'


class Recoge_datos():

    datos = pd.DataFrame()
    mensaje = 0
    datos_aux = []

    def __init__(self):
        pass

    def run(self):
        bandera = 0

        crypto = CryptoCurrencies(key=apikey, output_format='pandas')

        data_historical, _ = crypto.get_digital_currency_daily('BTC',
market='CNY')
        print(data_historical['4b. close (USD)'])



        while True:
            crypto_2 = CryptoCurrencies(key=apikey, output_format='pandas')
            if bandera == 1:
                data_intra, _ =
crypto_2.get_digital_currency_intraday('BTC', 'CNY')
                data_intra = data_intra[data_intra.index > start_time]
                data_intra = data_intra['1b. price (USD)']


            else:

                data_intra, _ =
crypto_2.get_digital_currency_intraday('BTC', 'CNY')
                data_intra = data_intra['1b. price (USD)']

            if bandera == 0:
```

Código test_image_recognition.py (Reconocimiento de imágenes con sklearn)

```
import datetime as dt
from sklearn import preprocessing
from sklearn.externals import joblib
from picamera import PiCamera
import time
import cv2
from sklearn.preprocessing import StandardScaler


# All the data files must go in here
def sklearn_preprocessing(X, y, method='scale'):
    print('Performing sklearn preprocessing via', method)

    if method == 'scale':
        return preprocessing.scale(X)
    elif method == 'standardscaler':
        return StandardScaler().fit_transform(X, y)


clases = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog',
'horse', 'ship', 'truck']

print('Started computation at', dt.datetime.now())

clf = joblib.load('modelo.pkl')

camera = PiCamera()
camera.resolution = (128, 128)

for i in range(0, 5):
    print(5 - i)
    time.sleep(1)

camera.capture('/home/pi/imagen.png')
imagen = cv2.imread('/home/pi/imagen.png')
resized = cv2.resize(imagen, dsize=(32, 32))
camera.close()

ima_feat = resized.reshape(1, -1)
print(ima_feat.shape)
ima_feat = ima_feat[0]
print(ima_feat)
input_features = sklearn_preprocessing(ima_feat, 0)
# print(input.shape)
starting_point = time.time()
print(clf.predict(input_features.reshape(1, -1)))
elapsed_time = time.time() - starting_point
print(elapsed_time)

print(clases)
[clf.predict(input_features.reshape(1, -1))[0]])
```

Código test_image_recognition.py (Reconocimiento de imágenes con CNN)

```
import tensorflow as tf
import random
import pickle
import matplotlib.pyplot as plt
import helper
# from skimage import io
from picamera import PiCamera
import time
from picamera.array import PiRGBArray
import cv2

import numpy as np

epochs = 10
batch_size = 1024
keep_probability = 0.5
save_model_path = './prueba2/image_classification'
n_samples = 4
top_n_predictions = 3
clases = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog',
'horse', 'ship', 'truck']


def normalize(x):
    """
    Normalize a list of sample image data in the range of 0 to 1
    : x: List of image data.  The image shape is (32, 32, 3)
    : return: Numpy array of normalize data
    """
    maximum = np.max(x)
    minimum = np.min(x)
    return (x - minimum) / (maximum - minimum)


def one_hot_encode(x):
    nx = np.max(x) + 1
    return np.eye(nx)[x]


try:
    if batch_size:
        pass
except NameError:
    batch_size = 64


def test_model():
    """
    Test the saved model against the test dataset
    """

    loaded_graph = tf.Graph()

    with tf.Session(graph=loaded_graph) as sess:
        # Load model
        loader = tf.train.import_meta_graph(save_model_path + '.meta')
        loader.restore(sess, save_model_path)
```

```
        # Get Tensors from loaded model
        loaded_x = loaded_graph.get_tensor_by_name('x:0')
        loaded_y = loaded_graph.get_tensor_by_name('y:0')
        loaded_keep_prob = loaded_graph.get_tensor_by_name('keep_prob:0')
        loaded_logits = loaded_graph.get_tensor_by_name('logits:0')
        loaded_acc = loaded_graph.get_tensor_by_name('accuracy:0')

        # ima_feat, ima_lab = pickle.load(open('prueba.p', mode='rb'))

        camera = PiCamera()
        camera.resolution = (128, 128)

        for i in range(0, 5):
            print(5 - i)
            time.sleep(1)

        camera.capture('/home/pi/imagen.png')
        imagen = cv2.imread('/home/pi/imagen.png')
        resized = cv2.resize(imagen, dsize=(32, 32))
        camera.close()

        # ima_feat = io.imread("/home/pi/imagen.png")
        ima_feat = resized.reshape(1, 32, 32, 3)
        ima_lab = np.array([0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0])
        ima_lab = ima_lab.reshape(1, 10)
        starting_point = time.time()
        pred_logits = sess.run(loaded_logits, feed_dict={loaded_x: ima_feat,
loaded_y: ima_lab, loaded_keep_prob: 1.0})
        elapsed_time = time.time() - starting_point

        print(elapsed_time)
        print(pred_logits)
        print(clases[pred_logits.argmax()])


test_model()
```