UNIVERSIDAD DE SANTIAGO DE CHILE
FACULTAD DE INGENIERÍA
Departamento de Ingeniería Informática

Implementación de método de *Deep Learning* para predicción de movimiento en interfaces cerebro-máquina

Guillermo Fernando Campos Muñoz

Profesor guía: Leonel Medina Daza

Trabajo de titulación presentado en conformidad a los requisitos para obtener el titulo de Ingeniero de Ejecución en Computación e Informática

Santiago – Chile
2020

RESUMEN

El presente informe tiene como objetivo el análisis de un nuevo método a utilizar para la transformación de señales neurológicas en actividad cerebral, se especifica la definición de un plan de trabajo y de los objetivos específicos del Trabajo de Titulación: Implementación de método de *Deep Learning* para predicción de movimiento corporal en interfaces cerebro máquina, desarrollado por el estudiante de Ingeniería de Ejecución en Computación e Informática Guillermo Campos Muñoz y guiado por el Dr. Leonel Medina Daza.

Para el desarrollo de este nuevo método se requiere de una base de datos, por lo tanto, lo primero es una descripción de esta, *Nonhuman Primate Reaching with Multichannel Sensorimotor Cortex Electrophysiology*, la cual esta publicada para su libre uso. Esta fue generada tomando mediciones de dos primates, captando señales directamente desde la corteza cerebral.

Se explica cada uno de los objetos que componen la base de datos, haciendo énfasis en que las mediciones de la misma son *spikes*, uno de los tipos de señales que sirven de entrada para una interfaz cerebro-máquina. También se generan algunos gráficos de los movimientos realizados por los primates para así tener un mejor entendimiento de la misma base de datos.

Se muestra todo el desarrollo realizado para la confección del algoritmo con el cual se puede realizar una predicción de movimiento, para el cual se hizo la implementación de un modelo basado en una red *LSTM* de la biblioteca *Keras*.

En los resultados obtenidos, se pueden visualizar las mediciones neuronales generadas por los datos de la base de datos antes mencionada, se analiza la predicción generada contra la curva esperada, se calculan los errores generados y los tiempos en los que incurre la ejecución del algoritmo.

Palabras Claves: biónica; *BMI*; *Deep learning*; inteligencia artificial; interfaces; *machine learning*; neurociencia; redes neuronales.

Dedicado a todos mis seres queridos y amados

AGRADECIMIENTOS

Agradezco a todos quienes me apoyaron en este largo proceso en la universidad, a quienes me entendieron en los momentos difíciles, a quienes me dieron palabras de aliento cuando me veía lleno de trabajos.

Se los debo agradecer ya que esto no sería posible solo gracias a mí, tengo una familia y seres queridos que me ayudaron a sobrepasar esta tarea.

Este trabajo fue financiado parcialmente por ANID - Fondecyt 11190822 y ANID - Iniciativa Científica Milenio, código NCN19_161

TABLA DE CONTENIDO

ÍNDICE DE TABLAS

ÍNDICE DE ILUSTRACIONES

CAPÍTULO 1. INTRODUCCIÓN

1.1 ANTECEDENTES Y MOTIVACIÓN

El prescindir de una extremidad puede generar situaciones de discapacidad para cualquier ser humano, dado que normalmente los ambientes no están preparados para recibir, por ejemplo, a personas en silla de ruedas, con muletas, etc. Son minoría los lugares que cuentan con rampas y/o ascensores para facilitar su acceso, y estos son vagos ejemplos en comparación de la gran diversidad de dificultades físicas que existen. Otra consecuencia de que lo anterior persista en el tiempo es que algunas de las situaciones que invalidan físicamente a un humano, también lo podrían imposibilitar para desempeñar cargos laborales.

Probablemente la situación de discapacidad lleve a una persona al desempleo, y esto a su vez se traduce en una perdida económica importante, esto no solo afecta a la persona, sino a su núcleo familiar, más aún pensando que puede ser jefe/a de hogar.

Actualmente se tienen varios avances tecnológicos que hacen pensar en una solución que mitigue la problemática anteriormente descrita. Específicamente los artefactos biónicos y el tratamiento de las señales cerebrales hacen pensar que se puede mejorar el funcionamiento de las prótesis biónicas para que estas cada vez se acerquen más a lo que es el movimiento natural de una extremidad.

Las interfaces cerebro-máquina (ICM), son un sistema de comunicación que proveen dicho proceso para que a través de señales cerebrales se realice un movimiento en un artefacto biónico.

Como se observa en la figura 1.1, para que funcione una interfaz cerebro-máquina es necesaria la adquisición de una señal, lo que para este documento esta cubierto utilizando la base de datos *Nonhuman Primate Reaching with Multichannel Sensorimotor Cortex Electrophysiology*, que entrega las mediciones digitalizadas en forma de *spikes*. Luego se tiene una etapa de procesamiento de señal, en el caso de este trabajo, la señal digitalizada de entrada es un *spike*, y luego del procesamiento que realiza el modelo *LSTM* se obtienen las características de la entrada, con lo cual solo quedaría un paso intermedio de traducción para realizar el movimiento en cualquier tipo de aplicación.

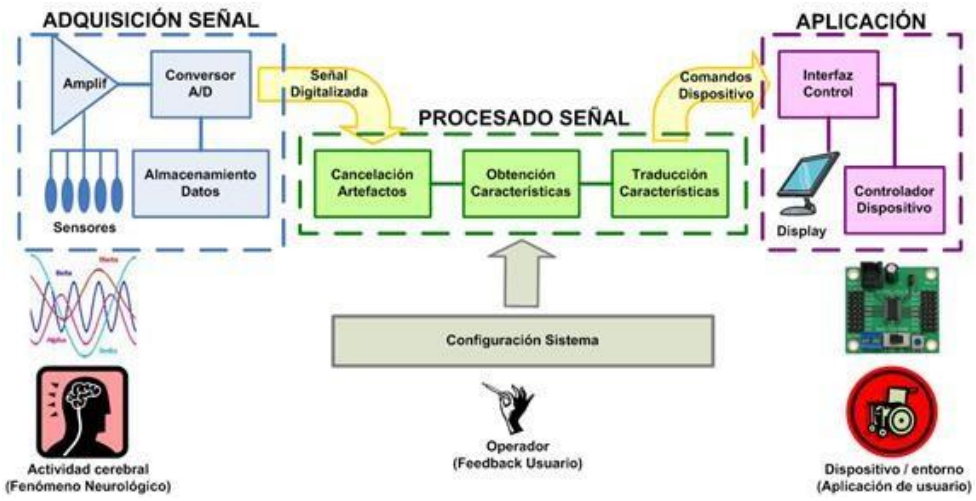

Figura 1.1: Esquema funcionamiento de una interfaz cerebro-máquina
Fuente: (Marte, 2019)

Si se mejora el método que se utiliza en la codificación de señal en la ICM, como consecuencia se mejora la calidad del movimiento realizado, siendo más parecido a lo que realiza el órgano humano que se reemplaza.

Si se logra dar solución a la problemática anterior y se mejora la transformación de la lectura de señales, mejoraría el funcionamiento de los artefactos biónicos. Así se puede ayudar a las personas que poseen algún tipo de discapacidad física, y de esta manera, probablemente pueda reinsertarse en el mundo laboral.

1.2 DESCRIPCIÓN DEL PROBLEMA

Se busca mejorar el proceso de predicción en interfaces cerebro-máquina, y poder controlar dispositivos biónicos de manera eficaz y eficiente. En base a esto se puede generar la siguiente pregunta:

¿Qué método utilizar para que, a partir de un *dataset* de señales cerebrales, estas se transformen en predicciones de movimiento, para así, generar el mismo en un artefacto biónico que ayude a recuperar movilidad a personas en situación de discapacidad física?

La duda anterior hace plantear la siguiente hipótesis *"se puede generar un método basado en Deep Learning que sea más efectivo en la predicción de movimiento, que los métodos utilizados actualmente"*.

1.3 SOLUCIÓN PROPUESTA

1.3.1 Características de la solución

Como solución se propone la modelación e implementación de un nuevo método basado en *Deep Learning*, el cual optimice la interpretación de señales cerebrales, y de esta manera obtener una mejor lectura de la actividad cerebral adecuada.

¿Cómo se puede mejorar la interpretación de señales cerebrales? Al mejorar los algoritmos que permiten predecir alguna métrica de movimiento, como posición, velocidad y aceleración. En este documento el enfoque se trabaja en función de la velocidad, la cual es obtenida a través de la derivada discreta de las mediciones de posición de la punta del dedo del primate en los ejes X e Y.

¿Cómo se puede evaluar una mejora en la predicción de movimiento? Se tienen métricas que permiten identificar qué curva (posición, velocidad, aceleración) arroja mejores resultados con respecto a otras. Para este trabajo la comparación será realizada en base a la raíz del error cuadrático medio y el índice de correlación.

El método de este trabajo se basa en una red neuronal profunda (*Deep Learning*) y todos los detalles de su implementación son vistos en el capítulo de Metodología. Ya que la solución utiliza un método de aprendizaje profundo, se debe definir la estructura general necesaria para su utilización. Se compone de una entrada, la implementación de un modelo de *Deep Learning*, y finalmente una salida. Para la entrada todas las pruebas se realizan en base a un conjunto de entrada de señales neurológicas descritas en el articulo *Nonhuman Primate Reaching with Multichannel Sensorimotor Cortex Electrophysiology* (O'Doherty et al., 2017). Luego de la obtención de datos, algunos de estos deben ser procesados para obtener, por ejemplo, los *spikes* por intervalos de tiempo o las velocidades, además se requiere un procesamiento para llegar al cuerpo (*shape*) que necesita el modelo como entrada. Para el modelo, se realiza su implementación utilizando una biblioteca provista por *Keras*, así que se crea una capa de abstracción de la lógica que usa el aprendizaje profundo para su funcionamiento. Solo se hace necesario la definición de la estructura del modelo y los hiperparámetros que definen su comportamiento. Para la salida, se puede requerir también un trabajo de cuerpo de los vectores de datos, para obtener la forma (x, 1) y de este modo poder graficar con respecto a otro vector, por ejemplo, el tiempo.

1.3.2 Propósito de la solución

La finalidad de este trabajo es ayudar a mejorar la predicción de movimiento a partir de señales cerebrales. Esto puede ser útil para los investigadores del campo de la neurociencia, para que lo consideren en sus estudios, y así poder conseguir resultados más certeros y precisos, finalmente de esta forma poder avanzar de manera efectiva en este campo.

Los principales beneficiados serían las personas con situación de discapacidad física, ya que, haciendo una buena predicción de las señales, por ejemplo, se podrían utilizar estas predicciones para controlar una mano mecánica en una persona que tuvo una amputación de su extremidad. Esto le permitiría recuperar algún grado de independencia al ejecutar algunas tareas cotidianas.

1.4 OBJETIVOS Y ALCANCE DEL PROYECTO

1.4.1 Objetivo general

Implementar y medir la efectividad de un método de *Deep Learning* en la predicción de movimiento a partir de registros cerebrales y comparar con métodos convencionales en interfaces cerebro máquina.

1.4.2 Objetivos específicos

1. Entender el contenido de la base de datos a utilizar *Nonhuman Primate Reaching with Multichannel Sensorimotor Cortex Electrophysiology* (O'Doherty et al., 2017).
2. Definir métricas de comparación, que informen efectivamente que método es superior.
3. Investigar sobre los métodos actualmente utilizados para la predicción de movimiento en base a señales cerebrales descritas en *Nonhuman Primate Reaching with Multichannel Sensorimotor Cortex Electrophysiology* (O'Doherty et al., 2017).
4. Implementar un algoritmo basado en *Deep Learning* el cual reciba como entrada los *spikes rates* calculados desde los datos de entrada y entregue las predicciones de velocidad del movimiento.

5. Realizar un análisis comparativo de los métodos convencionales con el método *Deep Learning* implementado.

1.4.3 Alcances

La solución propone un nuevo método para interpretar la actividad cerebral a partir de un conjunto de mediciones de señales neurológicas.

El alcance de la solución es generar un nuevo método el cual interprete señales cerebrales a partir de un *dataset* existente, se desea llegar a esta interpretación de actividad cerebral de manera de poder mejorar los parámetros de predicción, de los cuales se habla en detalle más adelante en este documento.

No se espera un 100% de efectividad, ya que se trata de artefactos que no forman parte naturalmente del cuerpo humano, pero, sí se espera mejorar los resultados que hasta el momento se han obtenido. Con esto se habla de obtener un *RMSE* menor y un índice de correlación mayor a los que exponen los métodos más relevantes de la actualidad.

Para calcular el error cuadrático medio (MSE), como se muestra en la figura 1.1 se obtiene la media de las diferencias entre las predicciones y los datos esperados, elevados al cuadrado.

$$MSE = \frac{\sum_{i=1}^{n} (prediction_i - real_i)^2}{n} \tag{1.1}$$

Luego se obtiene la raíz del error cuadrático medio (RMSE), como se muestra en la figura 1.2 solo se calcula la raíz cuadrática del dato anterior (*MSE*).

$$RMSE = \sqrt{mse} \tag{1.2}$$

Las limitaciones están relacionadas a la disponibilidad de *hardware* a utilizar, dado que se necesitan elementos potentes con los que aún no se cuenta, y debido a su alto costo no es algo que se pueda solucionar rápidamente, se deben analizar alternativas tales como el uso de servidores de buenas características, con esto se podrían manejar los tiempos de arriendo.

La opción que se utiliza para mitigar esta limitación es el uso de *Google Colaboratory*, el cual provee incluso el uso de *GPUs*, las características que proveen son quizás similares a las de una *laptor gamer* de gama media en la actualidad. Con esto la ejecución completa del código

implementado demora alrededor de tres horas, esto considerando el entrenamiento completo de un solo modelo.

Los métodos actualmente utilizados deben ser convencionales (no basados en *Deep Learning*) ya que el que se propone en este proyecto es un método de *Deep Learning* con características que aún no se habían implementado con este tipo de mediciones (*spikes*). Este último se espera tenga resultados superiores con respecto a la correctitud de las predicciones.

Se deben plantear las métricas de comparación correspondientes para ambos métodos en base al mismo *dataset* de señales cerebrales y una vez obtenidos los resultados para cada uno de ellos, se va a poder seleccionar cual sería la mejor manera de seguir adelante en los estudios de esta área.

Como limitaciones del trabajo se debe indicar que el *dataset* de mediciones será el mismo descrito en el artículo *Nonhuman Primate Reaching with Multichannel Sensorimotor Cortex Electrophysiology* (O'Doherty et al., 2017), por lo tanto, no se debe realizar una etapa de adquisición de señal para la interfaz cerebro-máquina.

En el desarrollo de este trabajo no se utilizaron las predicciones para controlar algún dispositivo, tampoco se realiza el desarrollo de una plataforma para que sea integrable a otros módulos y no se considera que las predicciones deban ser en tiempo real (todo el desarrollo es en *offline*).

1.5 ORGANIZACIÓN DEL DOCUMENTO

Este trabajo se encuentra divido en 4 capítulos que se describen a continuación:

- Capítulo 1 "*Introducción*": inicialmente se describe a grandes rasgos el trabajo, como un resumen de este, haciendo ver lo que motiva este trabajo, definiendo lo que es la problemática para resolver, para posteriormente poder identificar los objetivos, alcances y limitaciones del trabajo.
- Capítulo 2 "*Marco teórico*": se hace una introducción en los conocimientos necesarios para poder entender terminologías especificas que influyen en el desarrollo de las actividades planteadas para la correcta realización del proyecto. También en ese capítulo se informa acerca de otros proyectos que están buscando dar solución a esta misma problemática. Finalmente se presenta la propia propuesta de solución para el problema abordado.
- Capítulo 3 "*Metodología*": en este capítulo se explica en detalle todo el trabajo realizado para completar la implementación que se menciona, esto con la finalidad de que cualquier persona que lea este documento pueda replicar los resultados obtenidos en el mismo.

- Capítulo 4 ”*Resultados y Discusión*”: se determinan todas las características que posee la implementación realizada, para así poder comparar en base a las mismas métricas, con otras metodologías utilizadas sobre el mismo *set* de datos. Por otra parte, cada vez que se habla de los resultados, se realiza la correspondiente interpretación de estos, y se realiza el contraste con los métodos a comparar.
- Capítulo 5 ”*Conclusiones*”: en el capítulo final, se determinan conclusiones generales del proyecto, además de finalmente informar si el método efectivamente es mejor que los utilizados actualmente y, por otra parte, dejar planteados posibles trabajos posteriores que se podrían realizar para seguir construyendo conocimiento en esta línea de trabajo.

CAPÍTULO 2. MARCO TEÓRICO

En este capitulo se contextualiza con lo que está a la vanguardia en cuanto a las tecnologías y conocimientos que influyen en el desarrollo de este trabajo. Aquí se puede observar lo que ayuda en la construcción de la implementación y otros desarrollos que estarían siendo competencia directa de esta propuesta de trabajo.

2.1 CONCEPTOS GLOBALES

El ser humano a lo largo de su historia siempre ha tenido el pensamiento intrínseco de querer saber el por qué de las cosas, y el funcionamiento del cerebro no es la excepción.

2.1.1 Biónica

El objetivo de la biónica es el diseño de todo tipo de recambios artificiales para sustituir órganos dañados o recuperar funciones perdidas en el organismo humano (Guacho Rivera, 2018). Así esta ciencia se encarga de producir abstracciones artificiales de objetos naturales (Miralles & Giuliano, 2008). Según su definición, esta ciencia provee soluciones para ir en desmedro de alguna situación de discapacidad física, por ejemplo, el uso de prótesis tales como la que se observa en la figura 2.1.

Figura 2.1: Ejemplo mano biónica
Fuente: (Knoepfler, 2015)

También son destacables los exoesqueletos mecánicos, los cuales son robots acoplados a las extremidades del cuerpo humano enfocados en el incremento de su fuerza, velocidad y rendimiento principalmente (López et al., 2014).

2.1.2 Neurociencia

Las obras de la mecánica mencionadas anteriormente funcionan gracias a la lectura de señales eléctricas, tales como las que genera el cerebro humano. La ciencia que estudia dicho órgano es la neurociencia, en esta, se observa la forma en la que actúa el cerebro.

Hoy en día sigue siendo un misterio la forma en la que actúa el cerebro. Si bien se entienden ciertos aspectos de esto, como que es el órgano principal del sistema nervioso, que se compone de unos 86.000 millones de neuronas y 100 billones de sinapsis (Robert & Andrés, 2014), las señales que transmite son eléctricas, entre varios y diversos descubrimientos al respecto.

Aún no se puede elaborar un modelo que se ajuste con perfecta precisión a la forma en la cual actúa el cerebro, y esto se hace necesario para poder replicar en un artefacto mecánico lo que naturalmente hace alguna de las extremidades del ser humano.

La neurociencia comprende una amplia gama de interrogantes acerca de cómo se organiza el sistema nervioso y cómo funciona para generar la conducta (Purves et al., 2012).

Los neurocientíficos estudian el sistema nervioso en muchos niveles diferentes.

Examinan las moléculas, las células nerviosas, las redes neurales y la estructura del cerebro, de forma individual y en conjunto, y cómo estos componentes interactúan para realizar diferentes actividades (NIH, 2019).

En esta ciencia se estudia cómo se desarrolla y funciona un sistema nervioso típico, como así también los trastornos y las enfermedades que causan problemas al crecimiento o funcionamiento del sistema nervioso (NIH, 2019).

2.1.3 Interfaces cerebro-máquina (ICM)

En base a las áreas de conocimiento anteriormente expuestas se crearon las interfaces cerebro-máquina (ICM). Una ICM es un dispositivo que permite establecer una comunicación con el mundo externo a partir de la actividad eléctrica cerebral sin la ayuda de los nervios periféricos o de la actividad motora (Cossio & Gentiletti, 2008).

Para utilizar una ICM se utilizan mediciones de actividad neuronal que se decodifican para convertirlas en una señal de control (Makin et al., 2018). En el desarrollo de esta área se han hecho diversos estudios en base a tomas de muestras de señales leídas en el cerebro al momento de efectuar un movimiento y luego se intenta predecir dicho movimiento con un algoritmo.

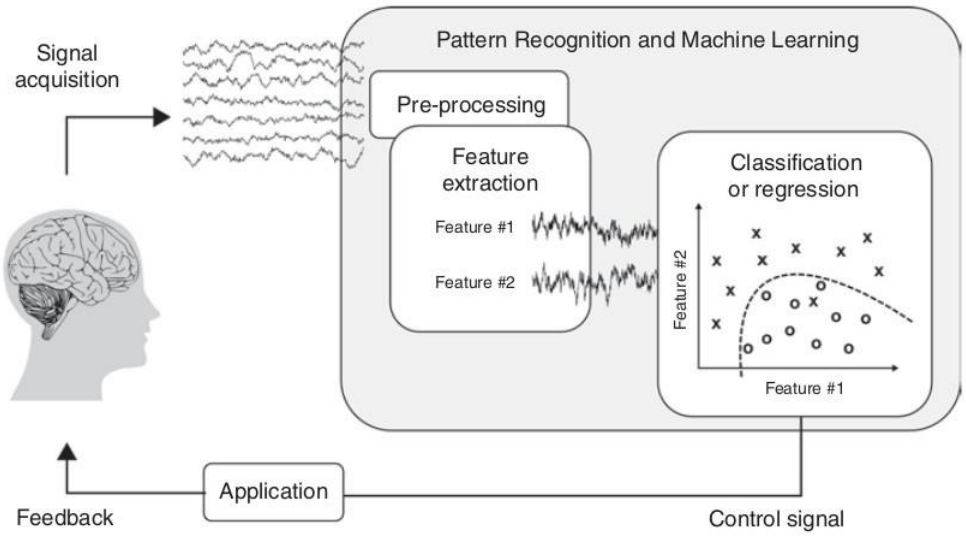

Figura 2.2: Ciclo de funcionamiento de una ICM
Fuente: (Marte, 2019)

Como se muestra en la figura 2.2 existe una adquisición de señal, la que después se procesa por medio de algún algoritmo. En el caso de este trabajo, el procesamiento lo lleva a cabo el modelo basado en aprendizaje profundo, luego el resultado se utiliza en algún tipo

de aplicación, la cual en sí misma, provee de retroalimentación a la persona, por lo tanto, es un proceso perfectible, y el resultado de este, siempre se puede mejorar en base al resultado anterior.

La invención de las ICM además de aumentar las capacidades físicas y psíquicas de los seres humanos (Peñaloza, 2019). Es de gran ayuda para aquellas personas que sufren situaciones de discapacidad físicas, por ejemplo, para el control y manejo de una silla de ruedas eléctrica. Además, estos sistemas prometen una mejor calidad de vida para estas personas en la medida que brindan un grado mayor de independencia para el sujeto (Cossio & Gentiletti, 2008).

Las posibilidades que crea este tipo de tecnología son infinitas, dado el hecho de que se puede lograr la comunicación con las máquinas con tan solo pensarlo. Empresas como *Neuralink* creada por Elon Musk, ya incursionan con las ICM. Su objetivo es implantar ”hilos” en el cerebro humano que permitan comunicar áreas concretas del mismo con el exterior. De esta forma, se pretende restaurar la capacidad de escuchar, hablar o moverse a personas que carecen de ella (Peñaloza, 2019).

2.1.4 Tipos de señales para una ICM

Para poder utilizar las ICM se necesita tener algún tipo de señal de entrada, un ejemplo de estas señales son los potenciales de campo local.

2.1.4.1 Electroencefalografía (EEG)

La electroencefalografía es una técnica de monitoreo electrofisiológico que permite registrar la actividad eléctrica del cerebro, que a su vez es producida por la actividad neuronal en el interior del cerebro (Gómez Figueroa, 2016).

Esta es una señal tomada de manera superficial como muestra la figura 2.3, por lo tanto, es muy poco invasiva, pero a la vez los datos no son tan certeros.

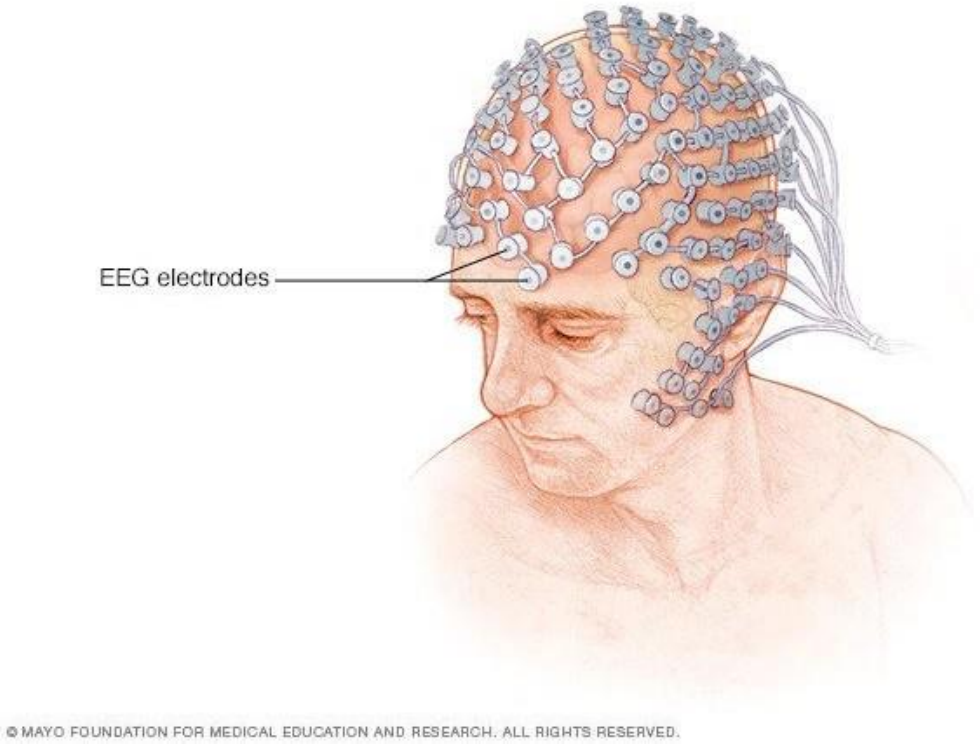

Figura 2.3: Adquisición de datos EEG
Fuente: (Pruthi, 2020)

2.1.4.2 Potencial de campo local

Los potenciales de campo local (*LFP*) son señales eléctricas transitorias generadas en los tejidos nerviosos y otros por la actividad eléctrica sumada y sincrónica de las células individuales (por ejemplo, neuronas) en ese tejido. Las *LFP* son señales ”extracelulares”, lo que significa que se generan por desequilibrios transitorios en las concentraciones de iones en los espacios fuera de las células, que resultan de la actividad eléctrica celular. Los *LFP* son ’locales’ porque son registrados por un electrodo colocado cerca de las células generadoras (Aso, 2021).

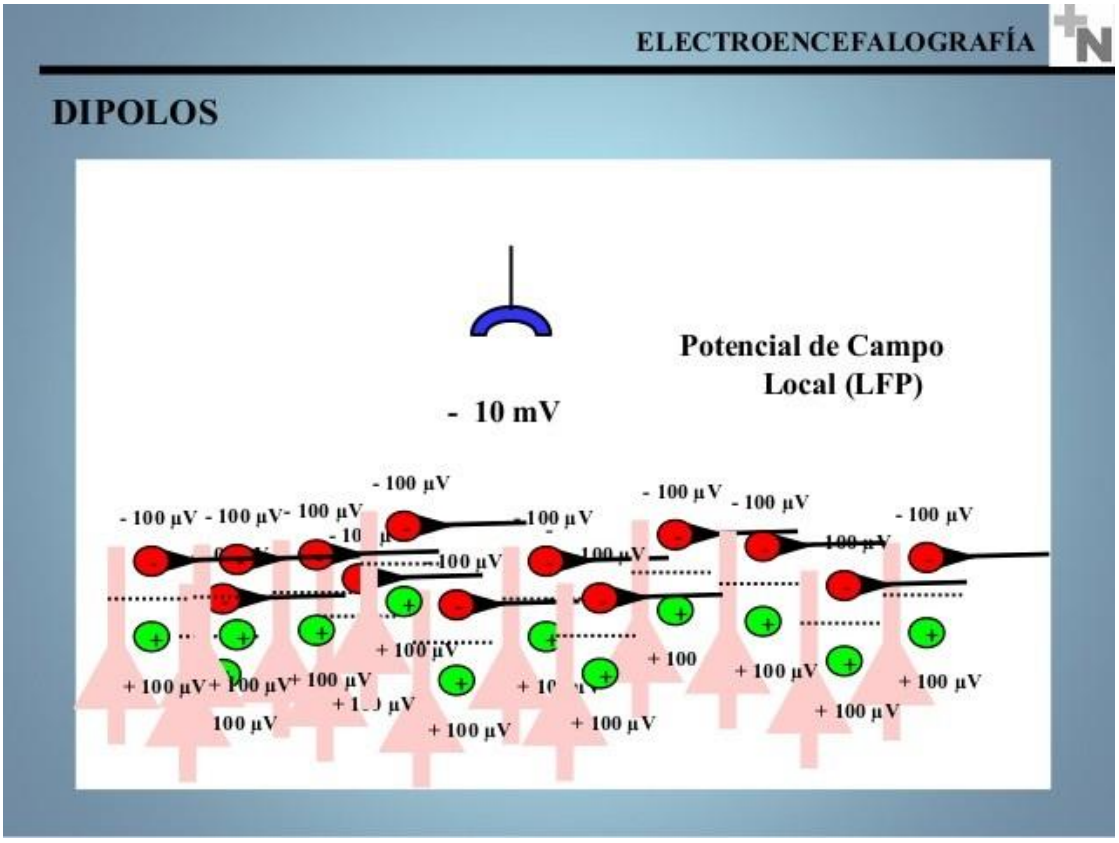

Figura 2.4: Ilustración Potencial de Campo Local
Fuente: (Pino, 2012)

2.1.4.3 Spikes

Por otra parte, se tienen los *spikes*, que al contrario de las señales *EEG* y *LFP* son invasivas ya que se adquieren por medio de chips implantados directamente en la corteza cerebral. Lo que permite una toma de mediciones más específicas y así los datos sean más certeros

En la base de datos utilizada, cada registro de *spike* tiene un registro correspondiente en la tabla de frecuencias (wf en la base de datos), por lo tanto, si un *spike* no es lo suficientemente fuerte para considerarlo en los cálculos, mejor se puede quitar ya que puede terminar siendo un registro *outlier*.

2.1.5 Inteligencia Artificial

Hoy en día los métodos que se utilizan para la predicción de movimiento no son 100% efectivos. En el documento *Superior arm-movement decoding from cortex with a new, unsupervised- learning algorithm* (Makin et al., 2018), se exponen comparaciones en torno a diferentes algoritmos actualmente utilizados, y cómo estos cambian en sus variables a predecir (posición, velocidad y aceleración) según varían ciertos parámetros de entrada como los intervalos de tiempo entre mediciones, el número de neuronas, etc. Por lo tanto, se deduce que no existe actualmente un método lo suficientemente robusto en cada una de sus aristas para poder soportar diferentes ambientes de pruebas.

La nueva propuesta de método basa su funcionamiento en el aprendizaje profundo. Pero antes de indagar en el *DL* se debe analizar el apartado de la ciencia que lo engloba, esto es conocido como la inteligencia artificial.

La Inteligencia Artificial (IA o en inglés *AI*) es la combinación de algoritmos planteados con el propósito de crear máquinas que presenten las mismas capacidades que el ser humano (Goodnight, 2021).

Las actuales aplicaciones de la inteligencia artificial son muy variadas, por ejemplo, se pueden crear *chatbots*, sistemas de detección facial, hasta diagnósticos médicos (Goodnight, 2021). Básicamente las limitaciones de lo que se puede llegar a hacer con esta tecnología solo está en la imaginación del ser humano que la utilice.

2.1.6 Redes Neuronales

Las redes neuronales artificiales son un modelo inspirado en el funcionamiento del cerebro humano. Esta formado por un conjunto de nodos conocidos como neuronas artificiales que están conectadas y transmiten señales entre sí. Estas señales se transmiten desde la entrada hasta generar una salida (Julián, 2014).

En la figura 2.5, se puede apreciar un esquema en grandes rasgos de lo que compone una red neuronal, donde se puede observar que se tienen capas de entrada, ocultas y de salida. Aquí cabe destacar que podrían existir más capas que hacen aún más complejo el funcionamiento de la red, y el número de nodos de cada capa normalmente es un hiperparámetro para configurar la red (Julián, 2014).

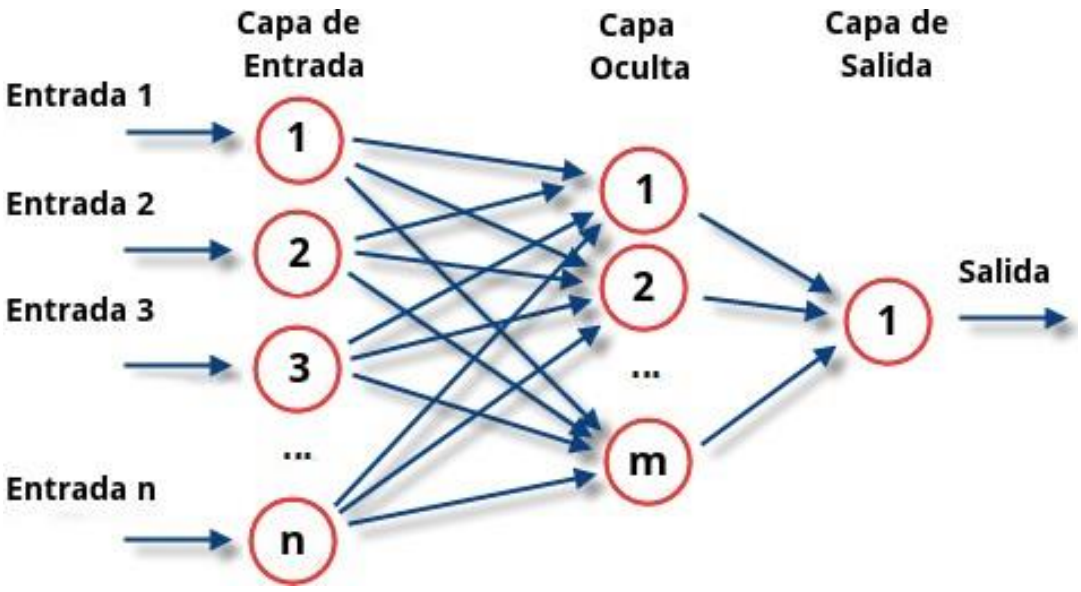

Figura 2.5: Esquema de una red neuronal
Fuente: (Julián, 2014)

El objetivo principal de este modelo es aprender vía actualización de los pesos de conexión entre neuronas, utilizando el error predictivo. Puede llegar a realizar tareas complejas que no podrían ser realizadas mediante la clásica programación basada en reglas. De esta forma se pueden automatizar funciones que en un principio solo podrían ser realizadas por personas (Julián, 2014).

2.1.7 Red Neuronal Profunda

Una red neuronal profunda (*DNN Deep Neural Network*) es una red neuronal artificial (*ANN*) con varias capas ocultas entre las capas de entrada y salida. Al igual que en las *ANN* poco profundas, las redes *deep learning* pueden modelar relaciones no lineales complejas (Montagud, 2019).

En la figura 2.6 se muestra la composición de una red neuronal profunda conformada por diferentes capas, entre las cuales existen conexiones entre cada par de nodos que las componen, y finalmente llegan a 1 salida.

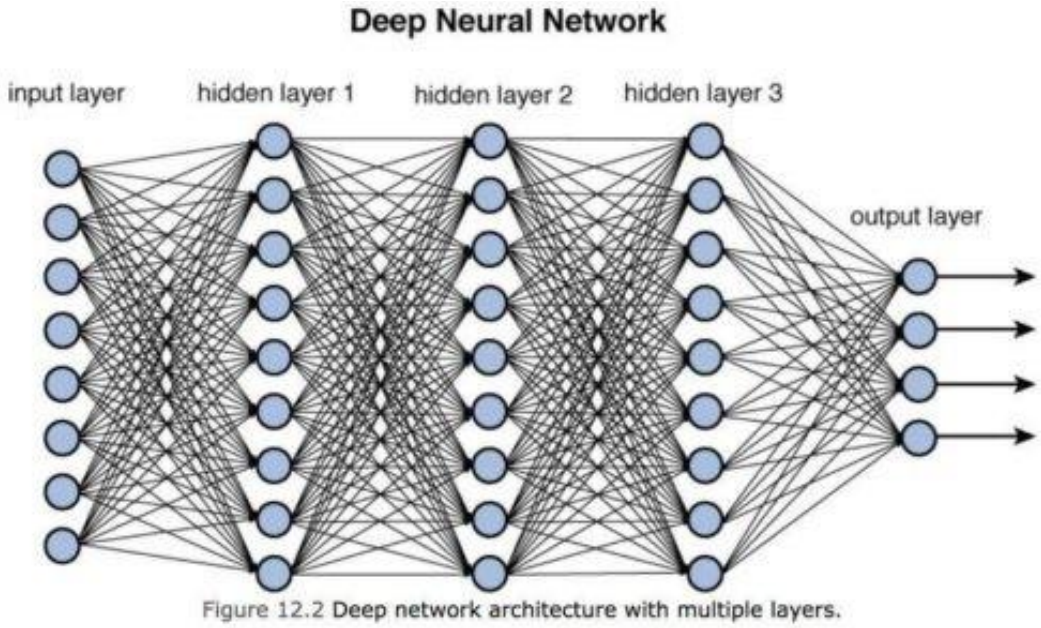

Figura 2.6: Esquema red neuronal profunda
Fuente: (Montagud, 2019)

Cada uno de los puntos mencionados anteriormente se pueden definir para la conveniencia de cada trabajo, esto es, el número de capas, el número de nodos de cada capa, el cuerpo de la salida, etc (Montagud, 2019).

Al pensar en métodos potencialmente efectivos se destacan los basados en *Deep Learning*, para los cuales ya se ha destacado su efectividad en otros experimentos. De lo anterior, se deduce que podrían ser igualmente efectivos en ICM.

Es evidente que ha habido un aumento de la utilización de métodos de *Deep Learning* en artículos publicados sobre estudios de medicina (Faust et al., 2018). Por ejemplo, Corti un asistente de voz que usan los servicios de emergencia en Dinamarca o RadIO un código de fuente abierta para la detección del cáncer (Melgar, 2018).

Además de su prominente popularidad, también es interesante saber que se utilizan en variadas aplicaciones que vemos diariamente como *Siri*, *Cortana*, buscadores de imágenes, predicciones de movimiento de bolsas de valores, etc. (Sharma, 2020).

Junto a todo lo anterior, este tipo de redes son capaces de aprender características complejas y dinámicas de datos, y esta descripción justamente se adapta a lo que son las características de los registros cerebrales.

El aprendizaje profundo tiene como objetivo automatizar y ayudar a la toma de decisiones dentro del ámbito de los negocios a través de la inteligencia artificial. Además, utiliza algoritmos avanzados para emular el aprendizaje humano, por lo que no son necesarias reglas

previas (Sharma, 2020).

Cada uno de los parámetros que utiliza una red neuronal pueden ser calculados por fórmulas matemáticas, para tener una aproximación, del número que sería correcto.

La siguiente fórmula sirve para determinar aproximadamente cual es el número correcto de nodos en una red neuronal, donde N_i es el número de neuronas de entrada, N_0 el número de neuronas de salida N_s el número de muestras en los datos de entrenamiento y α representa un factor de escala que suele estar entre 2 y 10 (Eckhardt, 2018).

$$N_h = \frac{N_s}{\alpha * (N_i + N_0)} \tag{2.1}$$

2.1.8 Red Neuronal Recurrente

Las redes neuronales recurrentes, o *Recurrent Neural Networks* en inglés, son una clase de redes para analizar datos de series temporales permitiendo tratar la dimensión de ”tiempo”, lo que no se ve en otros tipos de redes neuronales (Torres, 2021).

En cada instante de tiempo (también llamado *timestep*), cada neurona recurrente recibe la entrada de la capa anterior, así como su propia salida del instante de tiempo anterior para generar su salida (Torres, 2021).

Este proceso se puede observar en la figura 2.7 donde se ve el paso de datos a través del tiempo, el cual transcurre de izquierda a derecha, también se observa como cada una de las etapas recibe la información anterior y también su propia información para una salida más exacta dado el proceso de depuración que tiene.

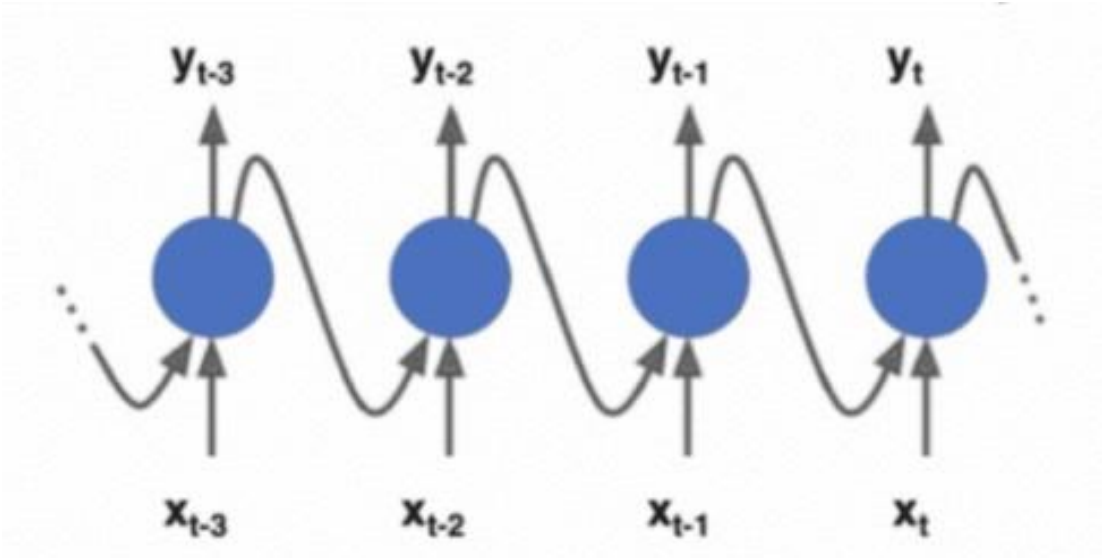

Figura 2.7: Esquema red neuronal recurrente
Fuente: (Torres, 2021)

La característica principal de las redes recurrentes es que la información puede persistir introduciendo bucles en el diagrama de la red, por lo que, básicamente, pueden ”recordar” estados previos y utilizar esta información para decidir cuál será el siguiente. Esta característica las hace muy adecuadas para manejar series cronológicas (Garzón, 2018).

2.1.9 Long Short Term Memory

Las *LSTM* del ingles *Long Short Term Memory* son un tipo especial de redes recurrentes, ya que este tipo de red permite un mejor trabajo con datos de tipo temporales y se piensa que el comportamiento de los *spikes* puede estar estrechamente relacionado con un factor de tiempo. Para esto se realiza un proceso en el trabajo, en el cual se calculan los *rate spikes*, esto es un índice de *spikes* obtenidos en un intervalo definido según una ventana temporal.

Mientras las redes recurrentes estándar pueden modelar dependencias a corto plazo (es decir, relaciones cercanas en la serie cronológica), las *LSTM* pueden aprender dependencias largas, por lo que se podría decir que tienen una ”memoria” a más largo plazo (Garzón, 2018).

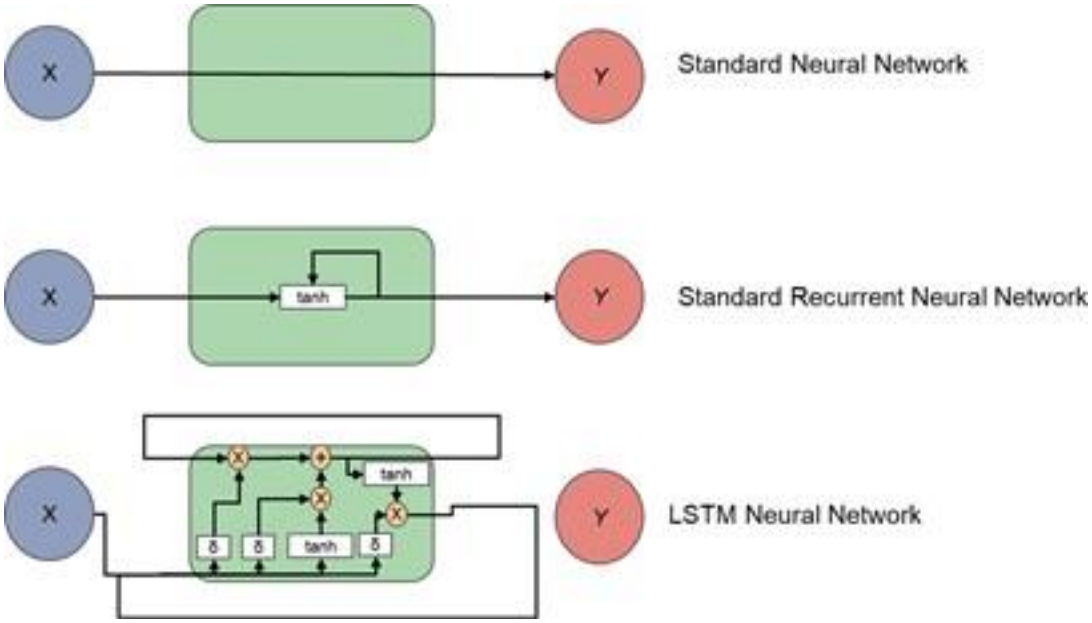

Figura 2.8: Comparativa LSTM y otras redes neuronales
Fuente: (Garzón, 2018)

En la figura 2.8 se observa la simplicidad que puede tener una red neuronal estándar, luego el cambio a una red neuronal recurrente, la cual utiliza su propia salida como retroalimentación para mejorar su efectividad, esto provee a la red de una cierta persistencia en los datos (”memoria”), además de la incorporación de una función de tangente hiperbólica.

Por otra parte, la red *LSTM* incluye todo lo de una red neuronal recurrente (ya que es una RNR), pero además posee diferentes capas lo que a diferencia de una RNR tradicional, le permite un aprendizaje a más largo plazo.

2.2 ESTADO DEL ARTE

2.2.1 Regresión Lineal

Como se muestra en la figura 2.9 si bien su representación es sencilla, sirve para obtener una buena relación entre datos, pero esta relación solo puede ser como una recta, como lo indica su fórmula.

$$Y = \beta_0 + \beta_1 X$$

Figura 2.9: Regresión Lineal Simple
Fuente: (Molina, 2020)

Si se quiere entrar en detalle, se deben utilizar métodos más especializados para así determinar un modelo con mayor exactitud.

La regresión lineal es fácil de utilizar, varios lenguajes de programación tienen bibliotecas que la implementan, por lo tanto, es muy sencillo su uso y bastante eficiente.

La regresión lineal se debería utilizar cuando se necesite un modelo óptimo para patrones de demanda con tendencia (creciente o decreciente), es decir, patrones que presenten una relación de linealidad (López, 2019).

Una regresión lineal utiliza el método de los mínimos cuadrados para predecir, simplemente entrega la recta que describe de mejor manera al conjunto de datos que recibe.

El método de los mínimos cuadrados se utiliza para calcular la recta de regresión lineal que minimiza los residuos, esto es, la diferencia entre los valores reales y los estimados por la recta (Molina, 2020).

Un ejemplo de interfaz cerebro máquina donde se utiliza la regresión lineal como función es (Olivares Carrillo et al., 2017), y como este trabajo, cualquiera podría utilizar este método, aunque dependiendo de la naturaleza del problema, probablemente no se obtienen los mejores resultados, ya que los datos pueden no comportarse de forma lineal como espera este método.

2.2.2 Filtro de *Kalman*

Un método que destaca fuertemente en la actualidad es el filtro de *Kalman*, este se compone de un conjunto de ecuaciones matemáticas que proveen una solución recursiva eficiente del método de mínimos cuadrados (Solera-Ramírez, 2003).

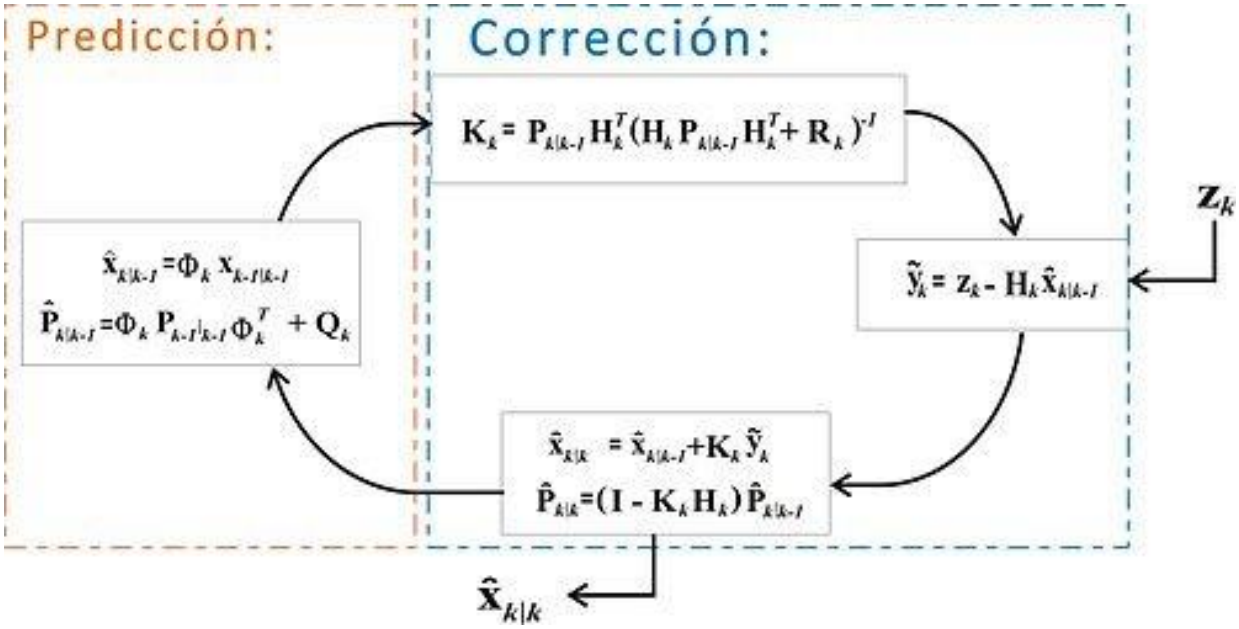

Figura 2.10: Esquema del filtro de *Kalman*
Fuente: (Solera-Ramírez, 2003)

Se busca no utilizar métodos como este que no se basan en el aprendizaje profundo, aunque como se observa en la figura 2.10, tiene una corrección lo cual se podría traducir en un aprendizaje que posee el algoritmo en su propia ejecución, aunque no llega a los niveles de aprendizaje del *Deep Learning*.

Básicamente el filtro de *Kalman* es un estimador de estados en un sistema dinámico, que toma en cuenta el ruido de las variables involucradas para generar una solución. De esta manera, realiza una estimación óptima de las variables utilizadas, en base a una señal y un modelo que explica esa señal, realiza el calculo de la predicción de la próxima posición. En la figura 2.10 se puede ver que la parte de predicción esta compuesta por dos fórmulas, la superior es para proyectar el siguiente estado y la inferior es para proyectar la covarianza del error. Cuando se entra en la zona de corrección, se comienza calculando la ganancia de *Kalman*, luego se actualiza la estimación y se actualiza la covarianza del error. La salida de este proceso sirve de entrada nuevamente para el proceso de predicción, para que observe que tan efectiva fue la predicción anterior, y de esa forma ajustar la próxima. Esta solución permite calcular un estimador lineal y óptimo del estado de un proceso en cada momento del tiempo con base en la información disponible en el momento t-1, y actualizar, con la información adicional disponible en el momento t, dichas estimaciones (Solera-Ramírez, 2003). Este filtro es el principal algoritmo para estimar sistemas dinámicos especificados en la forma de estado-espacio (*State-space*) (Solera-Ramírez, 2003).

Actualmente según se muestra en (Makin et al., 2018), el filtro de *Kalman* es el método

que obtiene los mejores resultados entre los métodos no basados en redes neuronales profundas en la actualidad. Por lo tanto, es de vital importancia contrastar los resultados obtenidos en el método planteado en este trabajo y el filtro de *Kalman*.

También se observa su uso en el documento (Ahmadi et al., 2019) en el cual también obtiene los mejores resultados fuera de los métodos que se basan en aprendizaje profundo. Probablemente sus resultados superlativos en el espectro de soluciones que no utilizan métodos de *Deep Learning* sea su matemática compleja que simula un aprendizaje en su ejecución. Sus resultados al predecir en promedio son, un *RMSE* de 61.15 ± 2.59 y un coeficiente de correlación de 0.69 ± 0.02. De lo anterior, se puede concluir que la raíz del error cuadrático medio obtenida, es demasiado alta quizás para lograr un movimiento tal como el que se busca reproducir. Por otra parte, el índice de correlación se puede ver un poco bajo, pero no es del todo malo, ya que se asegura que la curva no se esta sobre ajustando, pero a la vez, es otro indicador de que la predicción no se esta acercando demasiado a lo que se busca.

2.2.3 Enfoques de la Solución

En esta sección se presentan tres posibles enfoques para dar solución a esta problemática.

2.2.3.1 Mejorar parámetros del filtro de Kalman

Es una alternativa poder entender el método de *Kalman* en profundidad y poder mejorar de tal forma sus parámetros que se pueda llegar a una predicción mucho mejor de lo que se provee hasta el momento.

2.2.3.2 Crear modelo con lo que se conoce del cerebro

Aunque es posible, es un trabajo tremendamente complejo el poder definir un modelo matemático que defina el comportamiento de un cerebro en particular, y más aún adaptarlo para cada uno de los cerebros con los que se requiera trabajar.

2.2.3.3 Red neuronal recurrente

Para aumentar la efectividad del proceso, se tiene un amplio espectro de factores que influyen en este, pero se cree que la transformación más potente la produciría un cambio en el método a utilizar, es decir, una reestructuración total del modo en que se realiza la conversión de una señal neurológica en actividad cerebral.

Un método basado en una red *LSTM*, el cual en base a *spikes rates* calculados de las mediciones de un conjunto de datos existente, entregue como salida las velocidades del movimiento físico esperado.

2.2.4 Justificación del enfoque

La selección es red neuronal recurrente. Se utilizará un algoritmo orientado a la transformación de señales, dado que la finalidad es averiguar si un método *Deep Learning* puede ayudar a mejorar algunas de las métricas con que se mide el desempeño de una *ICM*.

Se espera que esta solución pueda mejorar la interpretación de las señales, si se lograse esto generaría un cambio notable en los resultados de esta área de investigación, y aquí cabe recalcar que lo que se desea lograr es la mejora en los tiempos de transformación de los datos y la ejecución de los movimientos, dado que estos dos factores, permitirían simular de mejor manera la naturalidad del movimiento humano.

Mejorar el filtro de *Kalman* parece que podría traer resultados poco mejores a los conseguidos hasta el momento, y según lo estudiado, el cambio a un método de aprendizaje profundo es mucho más prometedor.

La dificultad que significa formular un modelo cerebral sin utilizar un contexto como el filtro de *Kalman* o un método de aprendizaje profundo, es un trabajo muy complejo, quizás pudiendo llegar a resultados mejores que por cualquier otro camino, pero, también esta la posibilidad de que, invirtiendo mucho trabajo, no se superen los métodos actuales.

Como *bonus track* es interesante poder utilizar redes neuronales adversarias (RNA). Son un sistema compuesto por lo menos de dos redes neuronales, trabajando de manera conjunta para formar aprendizaje no supervisado.

El entrenamiento de una RNA requiere un tiempo de procesamiento mucho más prolongado (Soltani et al., 2019). Esto en comparación a la implementación ya realizada que demora aproximadamente tres horas su ejecución completa (dependiendo del conjunto de datos), lo que hace imaginar que un aprendizaje no supervisado podría demorar un par de días en su

entrenamiento.

2.2.5 Descripción de la tecnología implementada

Lo más complicado en la implementación es el manejo de la red neuronal, por lo tanto, esta sección hace especial énfasis en ese punto.

Para tener una noción básica de lo que es una red neuronal se puede hablar de una caja negra, a la cual se le entregan ciertos parámetros de entrada, y sin saber lo que pasa al interior, se obtiene una respuesta. Con esto se puede pensar que es como un método cualquiera, pero, lo que diferencia a una red neuronal de cualquier método es que poseen un tipo de aprendizaje.

De momento los aprendizajes pueden ser supervisados, parcialmente supervisado o no supervisado. De cualquier modo, esto quiere decir que la red neuronal a través de cada ejecución adquiere conocimiento del procedimiento que realiza, cambiando los resultados ejecución tras ejecución, y de esta manera cada vez se acerca más al modelo perfecto del objeto que desea modelar.

Formalmente, las redes neuronales son un modelo basado en el funcionamiento del cerebro humano. Se conforman por un conjunto de nodos los cuales cumplen el rol de una neurona, por lo tanto, su función es transmitir señales desde inicio a fin obteniendo una salida (Julián, 2014).

Cuando se cuenta con la red neuronal implementada, queda trabajar los datos de entrada, procesarlos con el modelo de aprendizaje profundo, para finalmente obtener conclusiones con las salidas obtenidas.

CAPÍTULO 3. METODOLOGÍA

3.1 CONTEXTO DEL DESARROLLO

3.1.1 Metodología de trabajo

La metodología para usar es el método científico, este consiste en cinco pasos, observación, hipótesis, experimentación, teoría y conclusiones (Gargantilla, 2019).

Primeramente, se realiza la observación, esta consiste en el momento en que un fenómeno se reconoce, el cual genera un impacto y supuestos, en el proyecto esto se ve reflejado al reconocer que existen puntos en el proceso de transformación de una señal neurológica en actividad cerebral los cuales pueden ser mejorados.

Luego de realizar la observación se postula una afirmación con respecto al evento donde se reconocen las falencias del sistema y se cree que existen puntos donde se pueden realizar mejoras, se plantea lo siguiente:

"se puede implementar un método basado en Deep Learning que sea más efectivo en la predicción de movimiento, que los métodos utilizados actualmente"

La efectividad se considera con los indicadores de raíz del error cuadrático medio y correlación.

En base a la hipótesis se establecen objetivos para cumplir con el alcance definido. Para este trabajo se contemplan la generación de métricas de comparación entre métodos, la selección de uno o más métodos altamente efectivos que se utilicen actualmente para la transformación de señales neurológicas en actividad cerebral, para comparar con el método propio implementado.

Con todo lo anterior se desarrolla una teoría, donde en base a las condiciones y al algoritmo planteado se debe llegar a los mismos resultados que se exponen en este proyecto.

Al finalizar el proceso completo se puede concluir efectivamente sobre la hipótesis planteada en un comienzo, donde se estipula la comparación del nuevo método desarrollado con los actualmente utilizados, en base a las métricas seleccionadas anteriormente. En este punto se tienen tres posibles respuestas, el nuevo método es en efecto mejor predictor que los actualmente utilizados, el nuevo método efectivamente es peor predictor que los actualmente utilizados, o finalmente puede que la respuesta no sea concluyente. Independiente de cual sea el resultado, toda la explicación del porqué de este, se ve en detalle en el análisis de resultados y en las conclusiones de este documento.

3.1.2 Herramientas de desarrollo

Las herramientas que se utilizan para el desarrollo de este proyecto son: *Python* por medio de *Google Colaboratory*. Se elige *Python* ya que es un lenguaje de programación que contiene amplia documentación con respecto al desarrollo de redes neuronales y bibliotecas tales como *Tensorflow* o *Keras*.

Para el control de versiones de utilizará *Git*, por medio del servicio *GitHub* y específicamente con la interfaz de uso *Git Kraken*.

Para generar documentación se utilizarán las herramientas proporcionadas por *Google*, sus documentos y plantillas, ya que, gracias a los beneficios de la cuenta institucional de la universidad de Santiago de Chile, se tiene almacenamiento ilimitado en *Google Drive* y teniendo una conexión a internet este permite el acceso a los documentos desde cualquier terminal.

Para el control de tareas, la herramienta a utilizar es *Trello*, ya que esta tiene una interfaz de usuario sencilla, también para acompañar la organización de tareas se generó una carta *Gantt* la cual permitió la planificación para el desarrollo de este trabajo.

3.1.3 Ambiente de desarrollo

El ambiente de desarrollo es en sistema operativo *macOS Big Sur* en su versión 11.2.2, el ordenador es un *MacBook Air* con procesador 1,8 *GHz Intel Core i5* de dos núcleos, memoria 8 *GB* 1600 *MHz DDR3* y gráficos *Intel HD Graphics* 6000 1536 *MB*.

La mayoría de las herramientas a utilizar son *online*, por lo tanto, la compatibilidad no es un problema. El lenguaje de programación *Python* es utilizado en su versión 3.8.5.

3.2 BASE DE DATOS

El archivo que contiene la base de datos tiene extensión .mat, lo que facilita su acceso desde el lenguaje de programación *matlab*, además, tiene un cifrado de tipo H5py que permite el almacenamiento de grandes cantidades de datos.

Para la confección del código se utilizó el lenguaje de programación *Python*, lo que complicó un tanto las cosas en cuanto al acceso a los datos de la base de datos, pero facilitó una tarea de mayor dificultad, que es el manejo de la red neuronal.

A continuación, se describen la base de datos y las funciones implementadas.

3.2.1 Descripción general de la base de datos

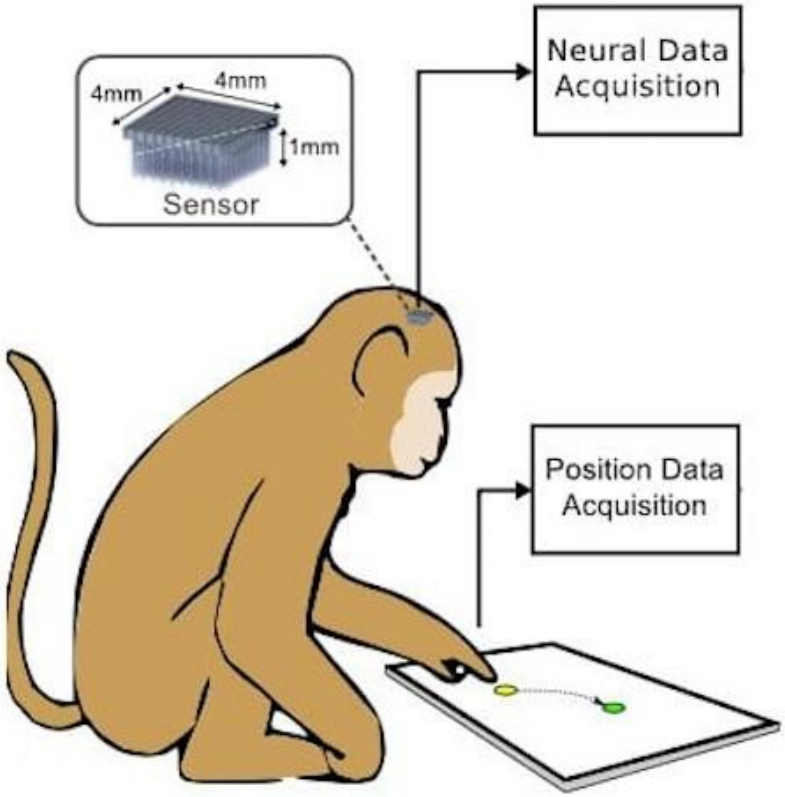

Figura 3.1: Diagrama desde adquisición de datos hasta red *LSTM*
Fuente: (Ahmadi et al., 2019).

El *dataset* de mediciones será el mismo que está descrito en el artículo *Nonhuman Primate Reaching with Multichannel Sensorimotor Cortex Electrophysiology* (O'Doherty et al., 2017). Cada uno de sus archivos posee la extensión *NWB* (*Neurodata Without Borders*) o *mat* dependiendo de la descarga que se desee. En la figura 3.2 se observan algunas de sus tablas, su contenido se explicará a continuación.

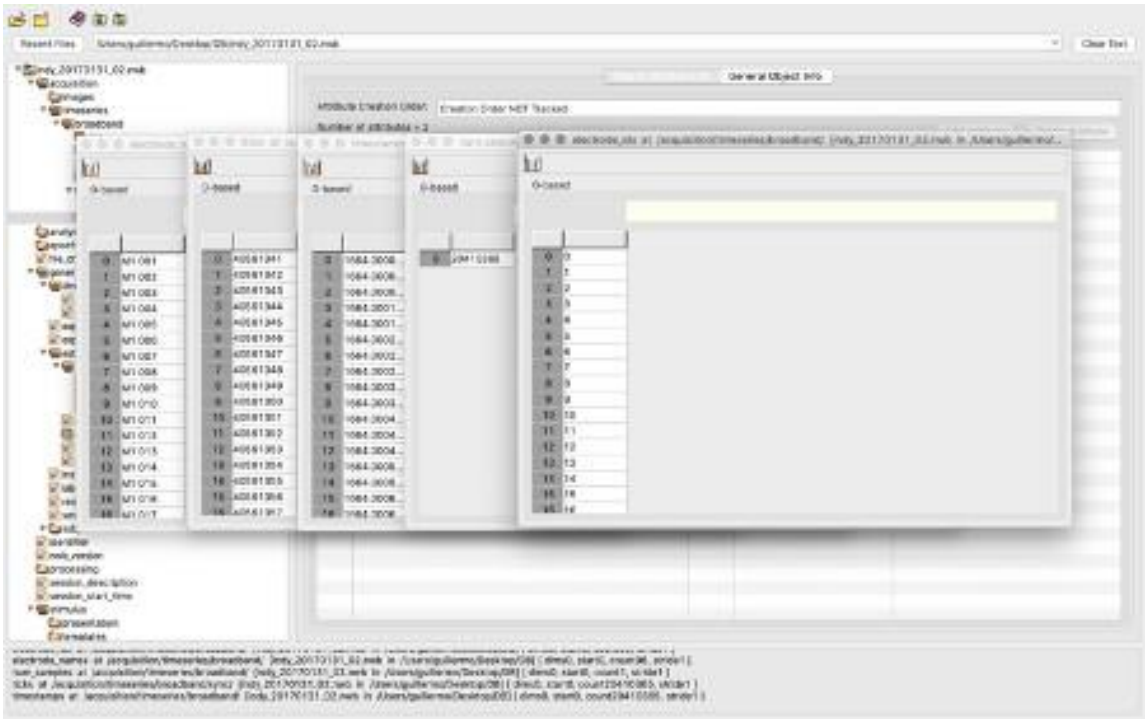

Figura 3.2: Tablas de archivo NWB visualizado con HDFView
Fuente: Elaboración propia, (2020).

El conjunto de datos se construyó en base a 37 sesiones con un mono llamado *Indy*, lo cual tuvo una duración de aproximadamente 10 meses, y otras 10 sesiones con un segundo mono llamado Loco, lo que abarco un período de tiempo de alrededor de un mes (O'Doherty et al., 2017).

El conjunto se datos esta compuesto por un total de aproximadamente 20.000 y 6.500 mediciones de los primates respectivamente (O'Doherty et al., 2017).

Cada medición consta de tiempos de cruces de umbral registrados extracelularmente, fragmentos de forma de onda, la posición (x, y) de la punta del dedo del primate y la posición (x, y) de los objetivos a alcanzar, los 2 últimos registrados a 250 Hz (250 veces por segundo) (O'Doherty et al., 2017).

Los datos fueron registrados sin espacios ni retraso previo al movimiento (O'Doherty et al., 2017).

La medición se llevó a cabo implantando un chip casi imperceptible en la corteza cerebral del primate, y lo interesante de esta base de datos en particular, es que las mediciones que provee son *spikes* un tipo de dato llamativo para poder realizar la predicción que se busca en este trabajo.

En detalle se desglosan los nombres de las variables (O'Doherty et al., 2017):

1. n: número de canales de grabación.

2. u: número de unidades clasificadas.
3. yk: número de muestras.
4. chan_names: matriz de celdas de cadenas de identificación de canal.
5. cursor_pos: posición del cursor en coordenadas cartesianas (x, y)(mm).
6. finger_pos: posición de la punta del dedo en coordenadas cartesianas (z, -x, -y)(cm).
7. target_pos: posición del objetivo en coordenadas cartesianas (x, y)(mm).
8. spikes: matriz de celdas de vectores de eventos de picos.
9. wf: matriz de celdas de ”fragmentos” de formas de onda de eventos de picos.

La base de datos también provee vídeos de *YouTube* donde se muestra como es la búsqueda del objetivo que realizan los primates, donde cada vez que alcanzan un objetivo, este se desactiva y se activa uno nuevo. Es posible ver esto en las figuras 3.3 y 3.4.

Figura 3.3: Ejemplo de objetivo 1
Fuente: (O’Doherty et al., 2017).

Figura 3.4: Ejemplo de objetivo 2
Fuente: (O'Doherty et al., 2017).

Las siguientes tablas permiten saber como es la composición de la base de datos. En la tabla 3.1 se muestran los datos más relevantes de la base de datos *indy_20170124_01*.

Datos importantes base de datos *indy_20170124_01*	
Dato	**Valor** (Aprox.)
Número de *trials*	59
Número total de *spikes*	593162
Promedio de *spikes* por *trial*	10053
Velocidad promedio por *trial*	0.036 [cm/s]
Tiempo promedio de *trial*	10 [s]

Tabla 3.1: Datos importantes base de datos *indy_20170124_01*
Fuente: Elaboración Propia, (2021)

En la tabla 3.2 se muestran los datos más relevantes de la base de datos *indy_20170127_03*.

Datos importantes base de datos *indy_20170127_03*	
Dato	**Valor** (Aprox.)
Número de *trials*	73.35
Número total de *spikes*	669322
Promedio de *spikes* por *trial*	9125
Velocidad promedio por *trial*	0.037 [cm/s]
Tiempo promedio de *trial*	10 [s]

Tabla 3.2: Datos importantes base de datos *indy_20170127_03*
Fuente: Elaboración Propia, (2021)

3.3 FUNCIONES IMPLEMENTADAS

3.3.1 Adquisición de datos

En este apartado se detallará cada una de las funciones implementada con fines de adquisición de datos.

La mayoría de estas funciones reciben por parámetro el archivo contenedor de la base de datos, al cual se accede a través de *Google Drive* y *H5py*.

3.3.1.1 Nombres de vectores

Para obtener los nombres de los vectores, la función recibe como argumento de entrada el archivo de la base de datos. En el desarrollo de la función primeramente se accede al archivo y se obtiene el objeto que lleva por nombre *chan_names*.

Luego se realiza un ciclo para trabajar cada dato de la variable *chan_names*, donde nuevamente se accede al objeto del archivo y se obtiene el nombre de vector según la posición que el ciclo indica. Como este dato viene en formato ASCII se debe realizar un procedimiento para transformarlo a formato alfanumérico y finalmente almacenarlo como *string* en el listado que retorna la función.

3.3.1.2 Posiciones del cursor

Para obtener las posiciones del cursor es similar a lo realizado anteriormente. Se recibe por parámetro de entrada el archivo contenedor de la base de datos, luego en base a este se obtiene el dato de nombre *cursor_pos*, posteriormente se accede a las dos dimensiones que tiene este dato, la primera dimensión se almacena en x y la segunda dimensión en y. Finalmente se vuelven a juntar estos datos en una lista que se devuelve al concluir la función.

3.3.1.3 Posiciones del objetivo

Se realiza exactamente el mismo procedimiento que implementa la obtención de las posiciones del cursor, pero en base al objeto del archivo *target_pos*.

3.3.1.4 Posiciones de la punta del dedo

El procedimiento es el mismo que el realizado para obtener las posiciones del cursor y del objetivo, pero en este caso se apunta al objeto *finger_pos*. Además, las posiciones obtenidas son tres, por lo tanto, estas son almacenadas en X, Y y Z, luego se unen en la lista que se es retornada cuando finaliza la ejecución de la función.

3.3.1.5 Obtener los tiempos

Se accede al objeto t en el archivo contenedor de la base de datos y este tiene una dimensión, así que se retorna ese conjunto de datos en un arreglo.

3.3.1.6 Obtener los spikes

Para la obtención de los *spikes* se requiere un poco más de trabajo. Igual que en los casos anteriores se recibe el archivo contenedor de la base de datos, sobre este archivo se

accede a su objeto *spikes*. Luego se realiza un primer ciclo, en base a *n_channels* variable global que contiene el número de canales, los cuales son 96 en esta base de datos.

Se procede a obtener los datos de cada *spike* recorriendo la matriz que los contiene, a la vez se realiza un *reshape* implícito de la variable, se pasa de un *shape* de (96, 5) a (5, 96), esto con fines de acceder primero a un canal y luego a las 96 neuronas que tiene cada canal.

En este trabajo se utiliza el primer canal el cual es nombrado *MUA*, contiene todas las mediciones sin clasificación, cada uno de los otros canales corresponden a clasificaciones de datos.

3.3.1.7 Obtener las frecuencias

Similar al trabajo de *spikes*, pero, en este caso se trabaja sobre el objeto *wf*. Para cada medición de *spike* se tienen 5 mediciones de frecuencia, por este motivo se realizó una nueva función que calcula el promedio y lo asigna en la variable que almacenaba la lista de los 5 registros.

3.3.1.8 Obtención general de datos

Esta función fue creada con la finalidad de unir las solicitudes de datos en una sola llamada, por lo tanto, recibe como parámetro el nombre de la base de datos requerida, la cual debe estar almacenada en el *file_path* que se designó en *Google Drive*. Aquí se obtienen la posición del dedo, los *spikes*, las frecuencias y los tiempos solo llamando a sus propias definiciones de función.

3.3.2 Trabajo con los datos

Esta sección es para mostrar todas las funciones que se crearon para realizar un análisis de los datos, por lo tanto, dependen de las funciones anteriores donde se obtuvieron los registros de la base de datos sin procesar.

3.3.2.1 Calcular promedio de frecuencias

Esta función tiene por finalidad calcular el promedio de las 5 mediciones de frecuencia que se tienen por *spike*, y así luego poder utilizar solo los *spikes* con una frecuencia promedio mayor a 0.5 como indica el documento de comparación (Ahmadi et al., 2019). El procedimiento se lleva a cabo de manera similar a la obtención de las frecuencias, recorriendo por número de canales y por las 5 mediciones que tiene cada frecuencia, luego se obtiene el promedio en base a los datos de la posición correspondiente y se retorna esta nueva lista como datos de frecuencia.

3.3.2.2 Obtención de MUA data

Se denomina *MUA data* al conjunto de datos que no fueron clasificados. Como se mencionaba anteriormente, los *spikes* pueden registrar hasta 5 mediciones diferentes, dependiendo de los nodos que captaron su activación, 4 de estos conjuntos son una clasificación realizada por los creadores de la base de datos, y el otro conjunto no fue clasificado, por lo tanto, cuenta con todos los registros captados.

En el procedimiento que se llevó a cabo para este calculo, se recorre todo el conjunto de *spikes* y en base a su correspondiente frecuencia inmediatamente se eliminan las mediciones con valor menor a 0.5, luego los registros que quedan son retornados.

3.3.2.3 Calcular spikes rate

Este es quizás el análisis de datos más complejo que se debe realizar.

Un *spike rate* es una métrica que permite determinar la actividad que tuvo un *spike* en una ventana de tiempo, como se muestra en la figura 3.1 se calcula con el conteo de los *spikes* que tuvo una neurona dentro de una ventana de tiempo y luego se divide por el tamaño de la misma ventana.

$$spikeRate(t) = \frac{\sum_{i=1}^{n} i}{banwidth} \tag{3.1}$$

En el procedimiento que se llevó a cabo para realizar este calculo se realizan dos iteraciones anidadas. La primera recorre todas las neuronas de *mua data* y la segunda, hace el

recorrido de toda la ventana de tiempo con traslape. Para cada iteración se calcula el *spike rate*, de modo que según el tamaño de ventana se realiza el conteo de *spikes* y luego se divide por el mismo tamaño de ventana. Este resultado es almacenado en una lista, y el proceso se repite para cada uno de los canales.

Al terminar los ciclos se realiza un procedimiento de *reshape* el cual permite ordenar estas mediciones con el fin de facilitar el trabajo posterior que requiere de estos datos.

3.3.2.4 Calcular las velocidades

$$velocityX = \frac{dx}{dt} \approx \frac{\delta x}{\delta t} \tag{3.2}$$

$$velocityY = \frac{dy}{dt} \approx \frac{\delta y}{\delta t} \tag{3.3}$$

$$velocity = \sqrt{velocityX^2 + velocityY^2} \tag{3.4}$$

En esta función primero se guarda el diferencial de velocidad (salto de tiempo) en la variable *dt*, ya que se busca realizar una derivada discreta. Luego se busca replicar lo que muestran las figuras 3.2, 3.3 y 3.4. Con el método de *numpy diff* se obtienen las diferencias de las posiciones, y al dividirlas en el paso de tiempo (*delta* de una derivada) se obtiene la derivada buscada de cada eje cartesiano. Finalmente se calcula la raíz cuadrada de la suma de los cuadrados de las velocidades.

3.3.2.5 Obtener spikes rate y velocidades

Se creó esta función para englobar varios de los procesos realizados anteriormente.

Por parámetros se reciben el nombre del archivo de la base de datos y un tiempo el cual es un número entero (solo para fines de pruebas), para así poder acotar el procesamiento.

Se realizó esta función para unificar la obtención de estos datos (*spikes rate* y velocidad) que se utilizan de entrada para el modelo.

Primeramente, se realiza la extracción básica de datos, y luego se utilizan las funciones explicadas anteriormente para ir trabajando los datos, se obtienen los promedios de las frecuencias y se realiza la adquisición de los datos del vector *MUA*. En base a lo anterior se calculan los *spikes rates* y la velocidad, para finalmente realizar un *split* a la velocidad para así poder ajustar su *shape* con fines del trabajo posterior.

3.3.2.6 Separación de datos para entrenamiento, pruebas y validación

Esta configuración fue tomada de ejemplo de (Ahmadi et al., 2019), donde se explica que todo el conjunto de datos obtenido se separa en 10 secciones independientes, y luego aleatoriamente se seleccionan 8 para entrenamiento, 1 para pruebas y 1 para validación.

Primero se define una lista con las 10 posiciones de cada uno de los subconjuntos de datos, luego 2 listas para los datos de X e Y, posteriormente se realiza una iteración con la finalidad de poder separar los 10 subconjuntos de datos.

Luego de finalizado el primer ciclo, se obtiene una nueva lista de posiciones en un orden aleatorio solo con 8 valores de la primera lista, posteriormente, se inicializan las variables contenedoras de los datos de X e Y con el valor que indica la primera posición de la lista, esto para luego llevar a cabo un ciclo con la función que provee *numpy concatenate*.

Después de finalizar el segundo ciclo, se obtienen las diferencias de ambas listas, quedando solo 2 valores, correspondientes el primero para validación y el segundo para pruebas.

3.3.2.7 Unir datos de bases de datos diferentes

Se unieron datos de dos conjuntos diferentes para llevar a cabo el proceso completo de la predicción, esto incluye el entrenamiento y el posterior uso del modelo para obtener la predicción.

Aunque fue un error, sirvió para notar que los mejores resultados se obtuvieron mezclando conjuntos de datos diferentes. Esto se debe a la heterogeneidad de la mezcla, lo cual hace que el sistema no se sobreajuste, y dado que todas las métricas hacen que reduzca el *RMSE*, se obtiene un buen predictor de un conjunto de datos más variado.

Se utilizó esta prueba para comparar y dejar un nuevo precedente, de igual forma para no perder la validación del modelo se realizaron las mismas pruebas de la literatura, para así poder realizar una comparación fehaciente.

3.3.3 Implementación del modelo

3.3.3.1 Entrenamiento

Al elegir el modelo correcto existen diversas configuraciones diferentes, dependiendo de cuantas capas se deseen agregar y de todos los hiperparámetros que se deben seleccionar.

Para facilitar lo anterior, se creó una función que permite en base a una configuración preasignada del modelo, que consiste en un método secuencial con 2 capas *LSTM*, una capa de *Dropout* y una última *Dense*, un optimizador de *Adam* y considerando una función de perdida basada en la raíz del error cuadrático medio.

Dropout es una capa de regularización, *Dense* es una capa completamente conectada con activación *ReLU* (*ReLU* es una función de activación que no es continua en 0). Si no hay una función de activación, todas las capas se pueden contraer en una sola capa (Iglesias, 2020).

El optimizador de *Adam* adapta el *ratio* de aprendizaje en función de cómo estén distribuidos los parámetros. Si los parámetros están muy dispersos el *ratio* de aprendizaje aumentará. Adam es una actualización del Optimizador RMSE (Calvo, 2018).

Teniendo la configuración anterior basada en el documento (Ahmadi et al., 2019), el modelo se puede ajustar utilizando los parámetros de entrada como hiperparámetros.

Esta función fue muy útil para probar el comportamiento de todos los hiperparámetros, ya que se pudieron realizar asignaciones de diferentes modelos fácilmente en una sola llamada a la función, tan solo variando el parámetro que se deseaba ajustar.

3.3.3.2 Pre procesamiento de datos

La preparación de los datos de entrada consiste en utilizar la función definida anteriormente para la separación de datos, y luego se realiza un *reshape* para poder ajustar los

datos al formato tridimensional que espera recibir el modelo.

3.3.3.3 Cálculo del error relativo

Se definió esta función para calcular el error relativo, se reciben por parámetro los valores reales y los valores de las predicciones, luego gracias al *broadcast* que permite *Python*, se aplican funciones matemáticas sobre los arreglos de entrada para emular la ecuación descrita en la figura 3.5. Se restan los valores esperados a los valores de predicción y el resultado se divide en los valores reales.

$$ERROR = \frac{prediction - real}{real} \tag{3.5}$$

3.3.4 Visualización de datos

Los *spikes* por rangos de tiempo se calculan para poder comparar posteriormente la actividad cerebral con lo que muestra un *rasterplot*.

Para esto primeramente se obtienen los datos de la base de datos, se calcula el promedio de las frecuencias, luego se obtiene solo el vector de *mua data*.

Para el procesamiento, primero se debe mencionar que por parámetros se reciben el tiempo mínimo y el tiempo máximo, ya que este es un proceso bastante costoso en cuanto a tiempo y recursos utilizados de la máquina, así que es bueno realizarlo en un período de tiempo más acotado para pruebas.

Solo se calculan los *spikes rates* en el tiempo definido según los parámetros de entrada, posteriormente se realiza un ciclo sobre los *spikes rates* entonces se calcula la suma de todos los *spikes* contabilizados y se divide por el número de canales, para tener un indicador promedio del número de *spikes* que se captan por canal. Luego se realiza un trabajo de tratamiento de los datos con fines de graficar.

Para preparar los datos del *rasterplot*, se realiza el procedimiento en un período de tiempo acotado aproximadamente a lo que sería un *trial*. La base de datos específicamente en su descripción dice que no fue dividida en *trials*, pero en el trabajo (Keshtkaran et al., 2021) se realizaron investigaciones para poder determinar una estimación de los tiempos que demora cada *trial*.

Para el procedimiento se realiza una iteración sobre los *spikes* y se va guardando toda la información de *mua data* que cumpla con los parámetros de entrada.

3.4 PRUEBAS REALIZADAS

3.4.1 Definición de métricas de comparación

Para las métricas de comparación se tomó como ejemplo el documento (Ahmadi et al., 2019), donde se compara en base a la raíz del error cuadrático medio y el índice de correlación.

Además, el error cuadrático medio es sensible al tamaño de los datos, no es lo mismo comparar datos entre 1 y 2, a comparar datos entre 1000 y 2000, aunque porcentualmente podrían tener el mismo error, el *rmse* del segundo ejemplo sería 1000 veces mayor al primero.

Se realizaron varias ejecuciones y las predicciones son dinámicas, pero siempre se obtiene un error cuadrático medio de aproximadamente 0.008.

La figura 3.5 muestra la comparativa entre los datos esperados y lo que fue la predicción del modelo, en aproximadamente el tiempo correspondiente a un *trial* donde se ven bastante similares ambas curvas.

Se realizó un calculo del índice de correlación de *Pearson*, donde se observa que es bastante alta la correlación de los datos reales y la predicción, siempre alrededor de un 99%, en cualquiera de las tres pruebas realizadas (*indy_20170124_01*, *indy_20170127_03* o un conjunto de ambos).

Se busca un método con error cuadrático medio más bajo e índice de correlación más alto. Ya que un *RMSE* bajo indica una baja diferencia entre los datos que se predicen y los datos reales. Por otra parte, un incide de correlación alto, indica un comportamiento proporcional de las curvas (se mueven de la misma forma).

Por medio del método de los mínimos cuadrados se puede obtener la recta con el menor *RMSE*, pero eso no necesariamente indica que la curva predicción se comporta de la misma forma que la curva real, por esa razón, se agrega el índice de correlación.

Estás métricas son evaluadores bastante certeros en cuanto a lo que se busca en este trabajo, ya que se espera obtener una curva que se ajuste de buena manera, pero tampoco termine sobre ajustada a un conjunto de datos.

3.4.2 Investigar métodos actuales utilizando la misma base de datos

Se utilizan las métricas dispuestas en el documento (Ahmadi et al., 2019), el cual provee resultados de diversos trabajos, especialmente los basados en el filtro de *Kalman*, el cual de momento provee los métodos más eficientes que utilizan la misma base de datos.

Comenzando este trabajo, no se conocía de algún otro trabajo que utilizara un método de *Deep learning* para la predicción de señales en cuanto a trabajo con *spikes*, pero, al iniciar la investigación profunda del tema, se encontró el documento (Ahmadi et al., 2019).

El documento (Ahmadi et al., 2019) utiliza una red *LSTM* para predecir, y se reconoce que el mejor vector para predecir es *MUA*, lo mismo que se especulaba al comienzo de este proyecto.

El documento mencionado, compara el método *LSTM* con métodos convencionales, y se reconoce que el método más efectivo (el que cumple mejor con las métricas de comparación), es el método *LSTM*. De todas formas, se realizó la implementación que se detalla en este documento y los resultados fueron diferentes.

3.4.3 Implementación de nuevo método *Deep Learning*

Anteriormente se mostraron todas las funciones implementadas para lograr la confección de lo que se necesitaba. Se pudo ver también como se probó el sistema y que este predice con un error bastante bajo, a la vez se acerca bastante a la curva que busca predecir.

A pesar de que el método propuesto en este trabajo ya fue implementado, su código no es *open source*, por lo tanto, la implementación ofrecida en este trabajo es totalmente particular, y se obtuvieron diferentes resultados en cuanto a aquello.

Como se observa en el desarrollo de este trabajo, las funciones implementadas, fueron realizadas desde sus cimientos, solo se utilizaron bibliotecas para el manejo de gráficos y del modelo que utiliza la red *LSTM*.

En las figuras 3.5, 3.6 y 3.7, se observan 10 segundos de mediciones en cada gráfico, donde en amarillo se muestra la curva de predicción y en morado la curva de medición real. Cada una de estás predicciones fue realizada con un conjunto de datos diferente, y en todas se observa una curva casi idéntica a la curva real.

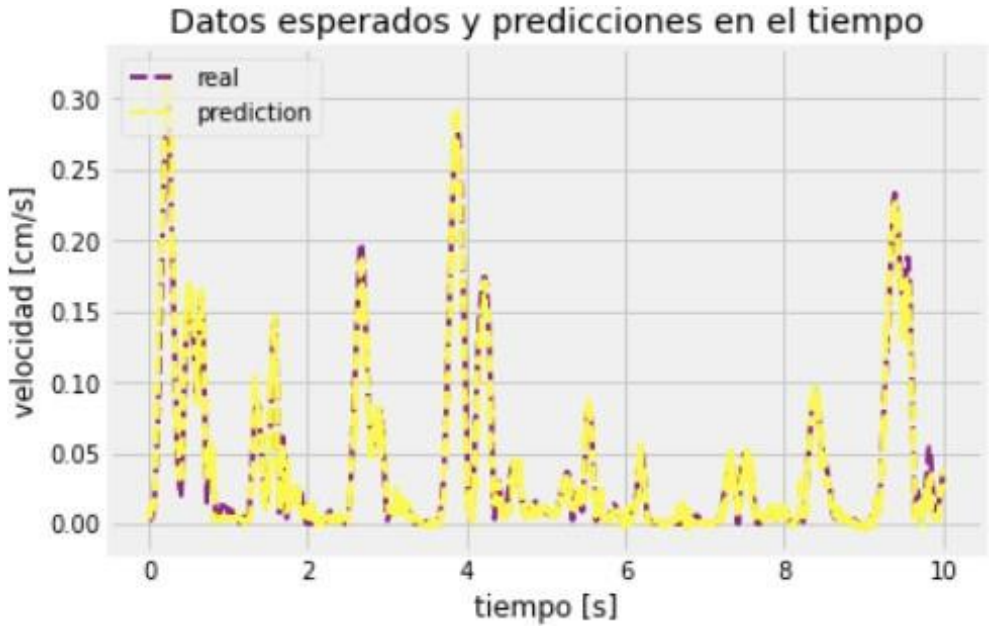

Figura 3.5: Gráfico valores reales contra predicciones
Fuente: Elaboración propia, (2020).

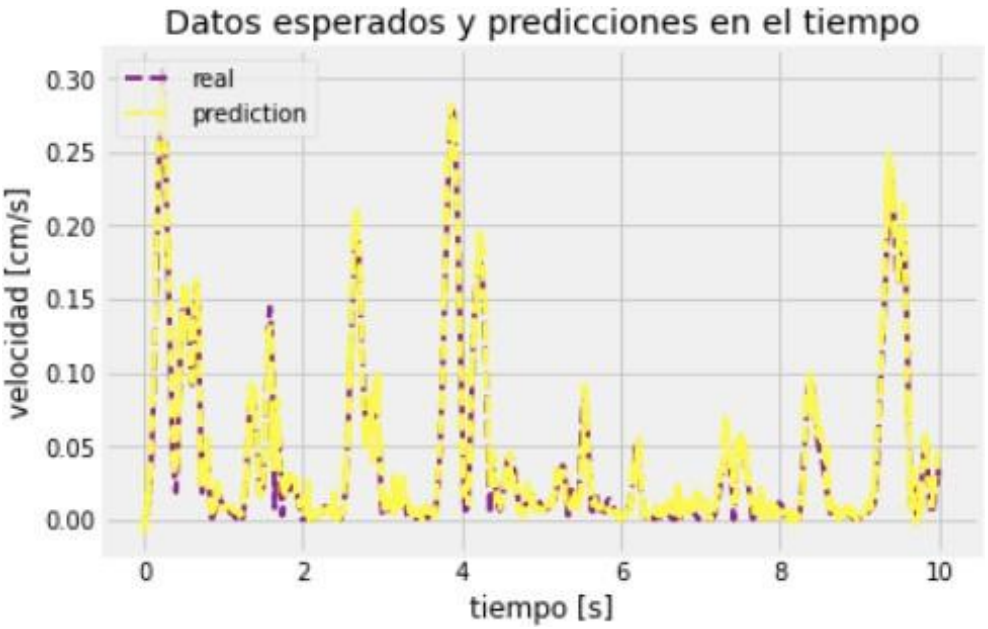

Figura 3.6: Gráfico valores reales contra predicciones
Fuente: Elaboración propia, (2020).

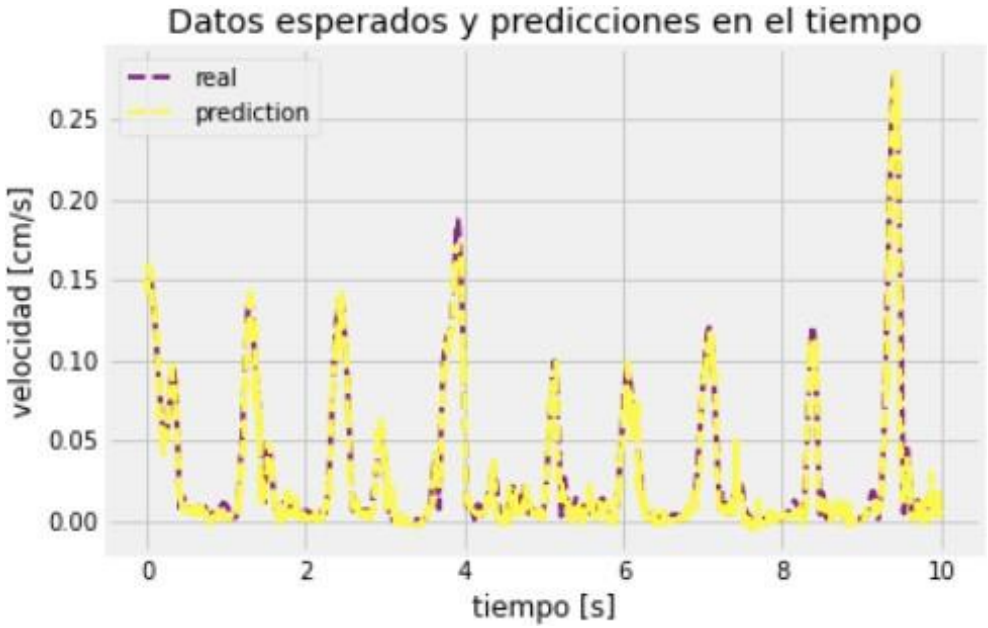

Figura 3.7: Gráfico valores reales contra predicciones
Fuente: Elaboración propia, (2020).

3.4.4 Ajuste de hiperparámetros

Para llegar a la configuración final de este programa, fuera de todo lo que es la adquisición y el trabajo de datos para utilizar posteriormente en el modelo, lo más complicado de determinar son los hiperparámetros de la red *LSTM*.

Para realizar este trabajo, primeramente, fue necesario establecer la estructura base de la red neuronal, esto consistió en una mezcla de datos obtenidos desde el documento (Ahmadi et al., 2019), datos recabados investigando en internet (*stack overflow*, entre otros), y luego realizando prueba y error, tratando de conservar una arquitectura minimalista, pero a la vez funcional.

Llegando a la configuración base de un modelo que trabaja en una función secuencial, con 2 capas *LSTM*, 1 *Dropout*, 1 *Dense*, con un optimizador de *Adam* y una función de perdida basada en el error cuadrático medio.

Ya con el modelo base, solo quedaba determinar cada uno de los hiperparámetros necesarios que se observan en la figura 3.8, para obtener los mejores resultados posibles.

Hyperparameter	Values
Number of units	$\{50, 75, 100, \cdots, 200\}$
Number of epochs	$\{2, 3, 4, \cdots, 8\}$
Batch size	$\{32, 64, 96, 128\}$
Dropout rate	$\{0, 0.1, 0.2, 0.3, 0.4\}$
Learning rate	$\{0.001, 0.0015, 0.002, \cdots, 0.003\}$

Figura 3.8: Hiperparámetros de LSTM
Fuente: (Ahmadi et al., 2019)

Para cumplir con esta última tarea, se implemento una nueva función, la cual permite entrenar el modelo con los parámetros de entrada de la función, y esta retorna el modelo entrenado, por lo tanto, se hizo mucho más sencillo tener una declaración y entrenamiento de un modelo en un solo llamado a una función.

Luego en base a las mismas investigaciones, y también realizando prueba y error, se determinó de qué forma de comportaban mejor el modelo según los parámetros utilizados. De lo que se puede concluir lo siguiente:

1. *Batch size*: mientras más pequeña mejor.
2. *hidden nodes*: mientras más grande mejor.
3. *return secuences*: para este caso solo puede estar en *true*.
4. *Units*: mientras más grande mejor.
5. *Dropout*: si se incrementa demasiado, incrementa la perdida.
6. *Epochs*: mientras más grande mejor.

Con ningún parámetro se debe exagerar, ya que se puede sobre ajustar la curva.

Finalmente, la tabla 3.3 muestra cada uno de los valores determinados para el modelo óptimo conseguido.

Hiperparámetros utilizados	
Hiperparámetro	**Valor**
Número de unidades	100
Número de epochs	10
Tamaño de ventana	100
Dropout rate	0.2
Learning rate	100

Tabla 3.3: Hiperparámetros utilizados
Fuente: Elaboración Propia, (2021)

3.4.5 Ejemplo de ejecución

Para llegar a los resultados del siguiente capítulo se pueden seguir los pasos que se describen ahora, primero se debe destacar que este desarrollo se realizó en *Google Colaboratory*, por lo tanto, el primer paso es conectar este entorno de desarrollo a *Google Drive*. Luego se importan las bibliotecas necesarias para el funcionamiento de cada una de las funciones: *matplotlib*, *numpy*, *keras*, *math*, *h5oy*, *random*, *sklearn*, etc. Posteriormente se ejecutan todas las declaraciones de funciones, para así luego utilizarlas tan solo con llamarlas.

Después se deben declarar un tiempo final y un diferencial de tiempo, esto fue implementado con fines de hacer más veloces las pruebas y probar como afecta el tamaño de ventana.

En este momento ya se puede hacer el llamado a la función *get_spikes_velocity*, la cual recibe solo dos parámetros, el tiempo que se definió anteriormente, y el nombre del archivo donde esta almacenada la base de datos. Al recepcionar el retorno de esta función se guardan simultáneamente los *spikes rates* y las velocidades.

Más adelante, se deben dividir los datos en entrenamiento, prueba y validación, con estos datos ya se puede entrenar el modelo, este es el punto que tarda más tiempo (pero encontrando el modelo correcto, ya puede ser exportado, para luego tan solo importarlo y volver a utilizarlo). Cuando el modelo esta preparado, ya se puede predecir utilizándolo.

Hasta este punto se logra conseguir la predicción, y con todo lo implementado, se pueden obtener los gráficos expuestos en este trabajo, igual que todo el análisis desarrollado.

CAPÍTULO 4. RESULTADOS Y DISCUSIÓN

A continuación, se muestran los resultados de este trabajo y se discute cada uno de ellos.

4.1 ENTENDER EL CONTENIDO DE LA BASE DE DATOS

El manejo de este tipo de bases de datos es complejo, ya que los datos están encriptados con *h5py*. Se debe acceder a objetos cifrados y utilizar bibliotecas que provee *Python* para llegar al dato necesario en cada caso. La visualización de la base de datos es más sencilla en el lenguaje de programación *Matlab*, sin embargo, el resto del desarrollo se llevó a cabo con bibliotecas como *Keras*, lo cual facilita las tareas con la red neuronal, un trabajo complejo de realizar.

Antes de acceder a los datos fue vital leer el documento que provee la base de datos, (O'Doherty et al., 2017), ya que en él se hace una descripción detallada sobre cada uno de los objetos contenedores de datos, como se realizaron las mediciones, etc. Posterior a leer los datos, se procede a realizar diversos gráficos, para entender el comportamiento de estos, y poder verificar el correcto acceso a los datos. Se descubre que con la base de datos se puede visualizar diferentes fenómenos relacionados a las mediciones que trae en cada uno de sus objetos.

En las figuras 4.1, 4.2, 4.3, 4.4 y 4.5 se ven ejemplos del movimiento de la punta del dedo según sus coordenadas cartesianas en los ejes X e Y.

Con estás curvas que están aproximadamente acotadas a la duración de un *trial*, se puede observar el movimiento del dedo del primate para mediciones del conjunto de datos *indy_20170124_01*.

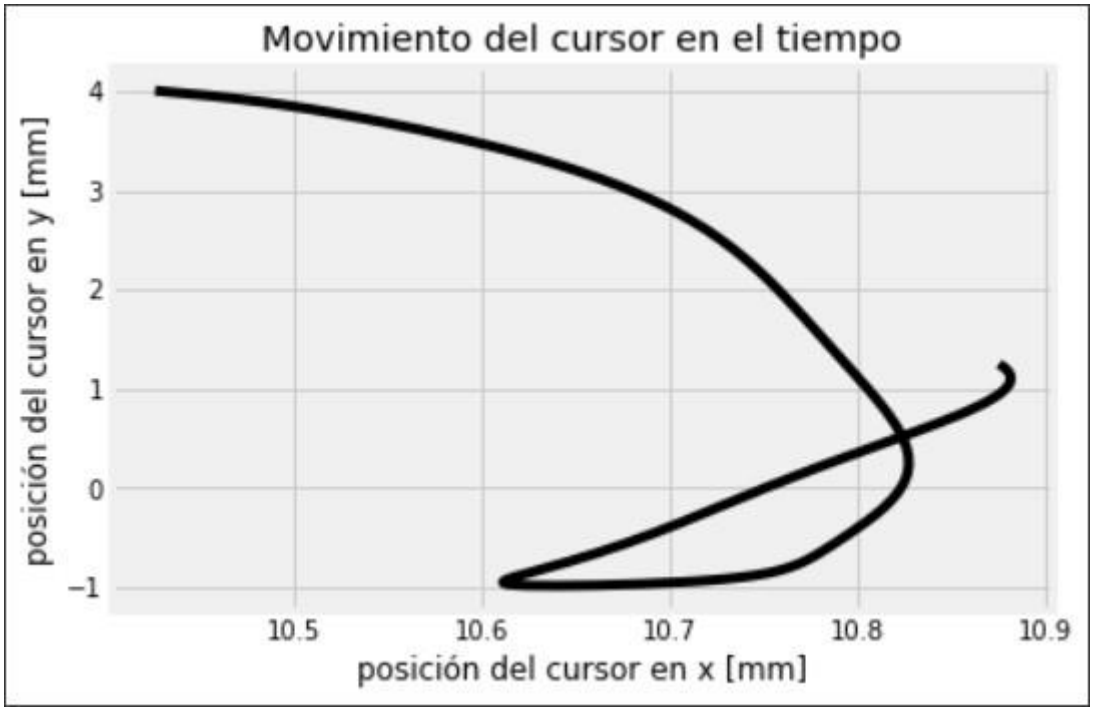

Figura 4.1: Posición X e Y en el tiempo *indy_20170124_01 trial* 1
Fuente: Elaboración propia, (2021).

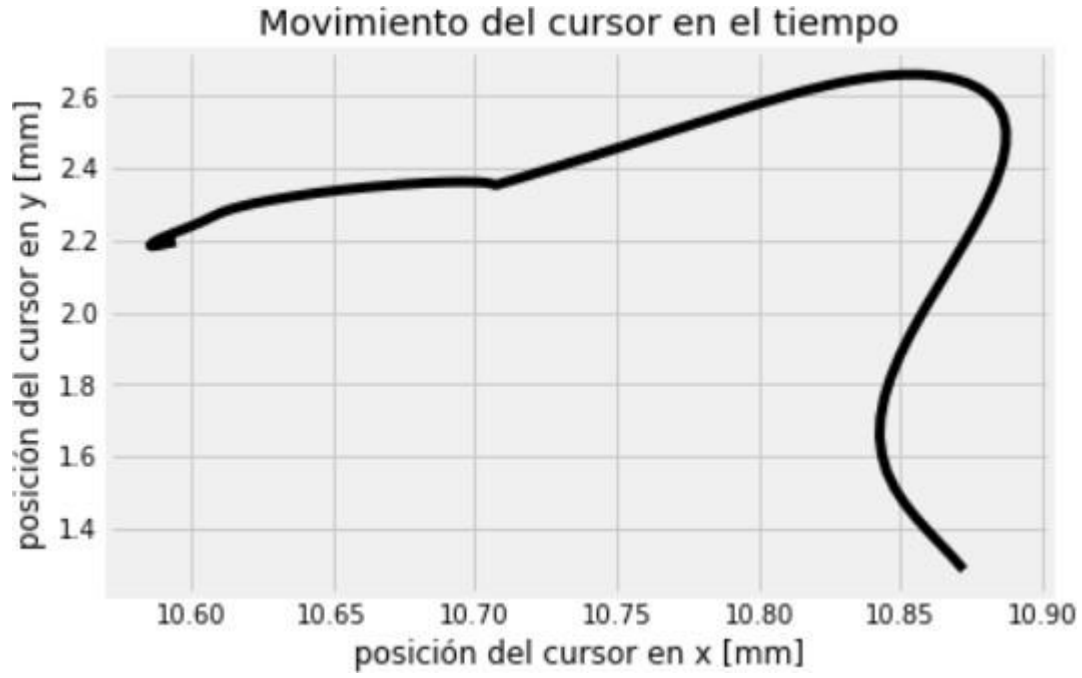

Figura 4.2: Posición X e Y en el tiempo *indy_20170124_01 trial* 2
Fuente: Elaboración propia, (2021).

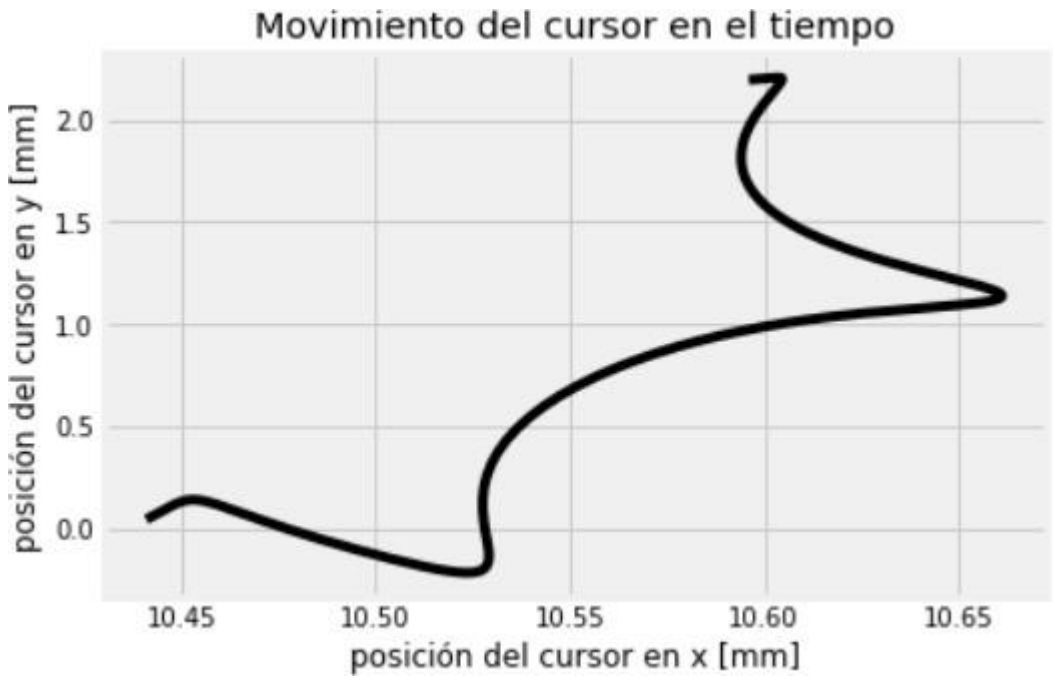

Figura 4.3: Posición X e Y en el tiempo *indy_20170124_01 trial* 3
Fuente: Elaboración propia, (2021).

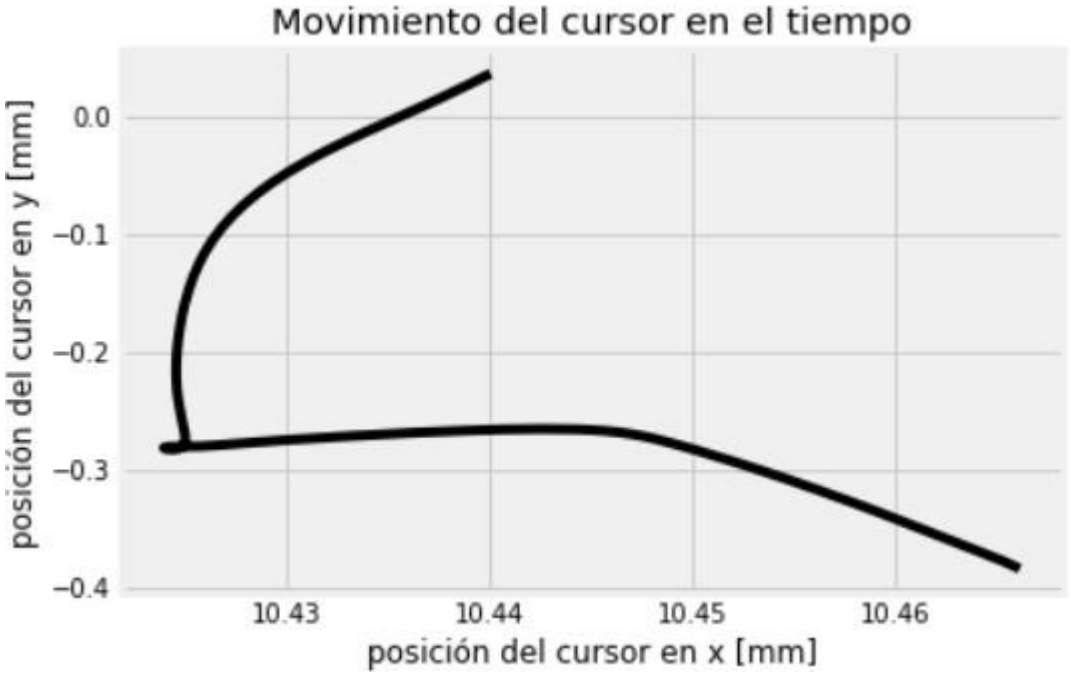

Figura 4.4: Posición X e Y en el tiempo *indy_20170124_01 trial* 4
Fuente: Elaboración propia, (2021).

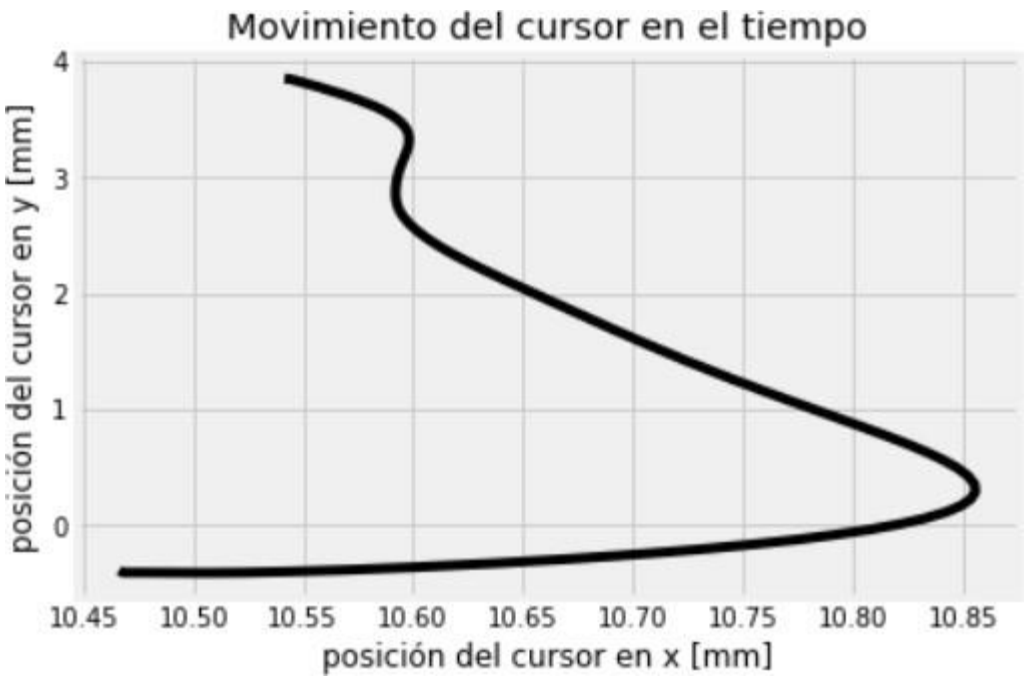

Figura 4.5: Posición X e Y en el tiempo *indy_20170124_01 trial* 5
Fuente: Elaboración propia, (2021).

En las figuras 4.6, 4.7, 4.8, 4.9 y 4.10 se ve la misma clase de registros, pero, en este caso obtenidos desde el conjunto de datos *indy_20170127_03*.

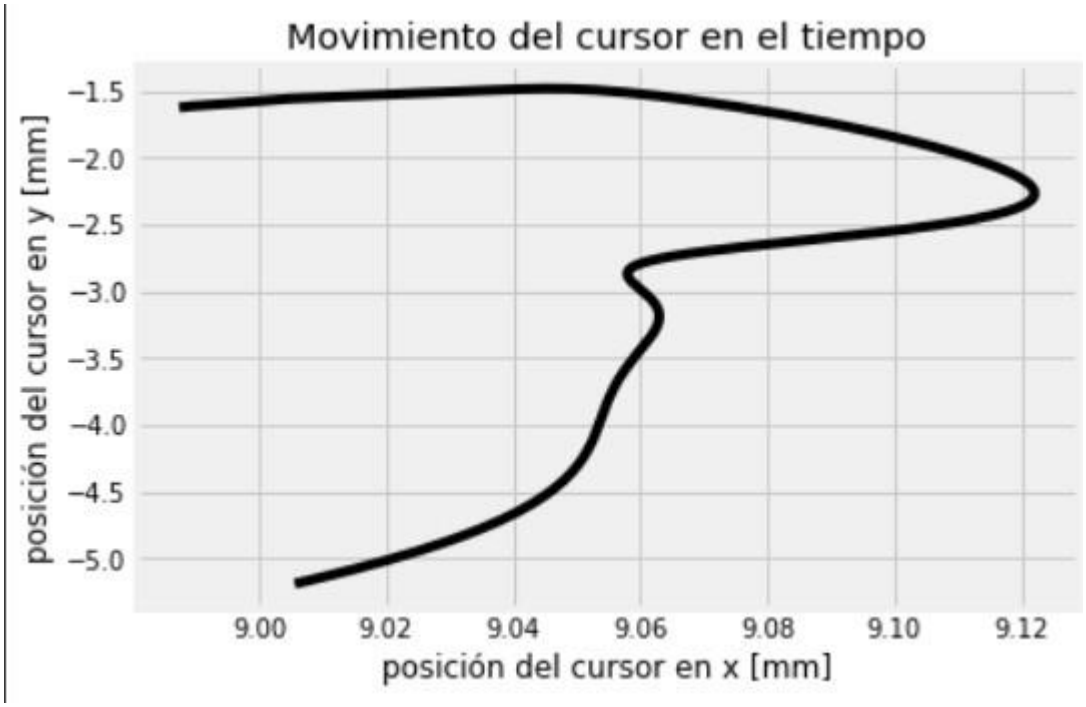

Figura 4.6: Posición X e Y en el tiempo *indy_20170127_03 trial* 1
Fuente: Elaboración propia, (2021).

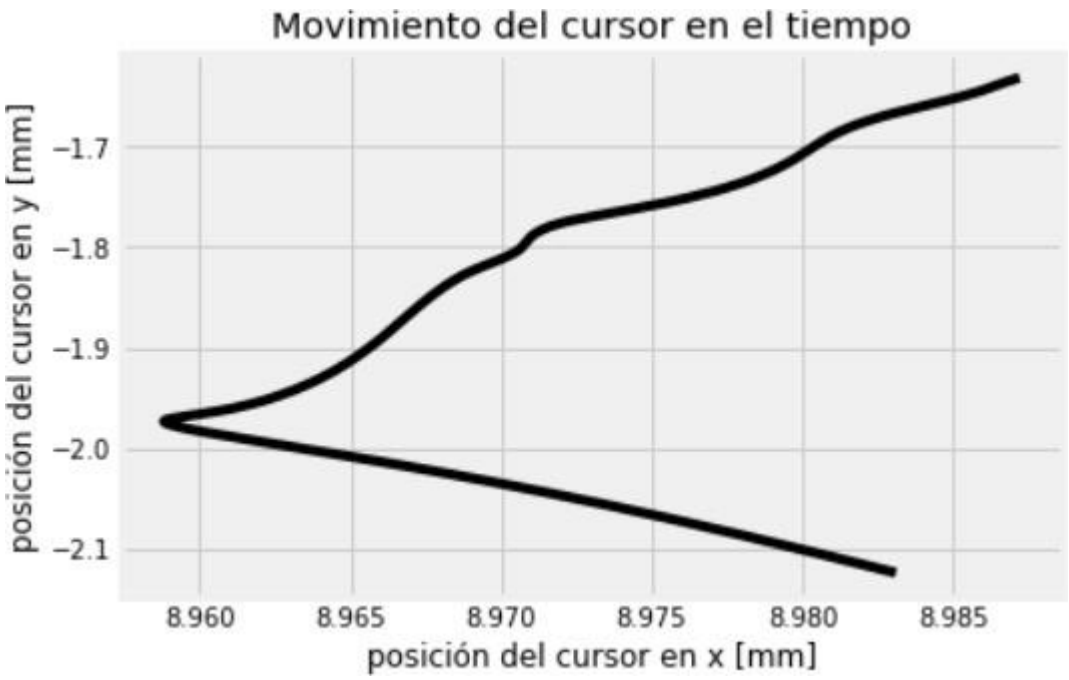

Figura 4.7: Posición X e Y en el tiempo *indy_20170127_03 trial* 2
Fuente: Elaboración propia, (2021).

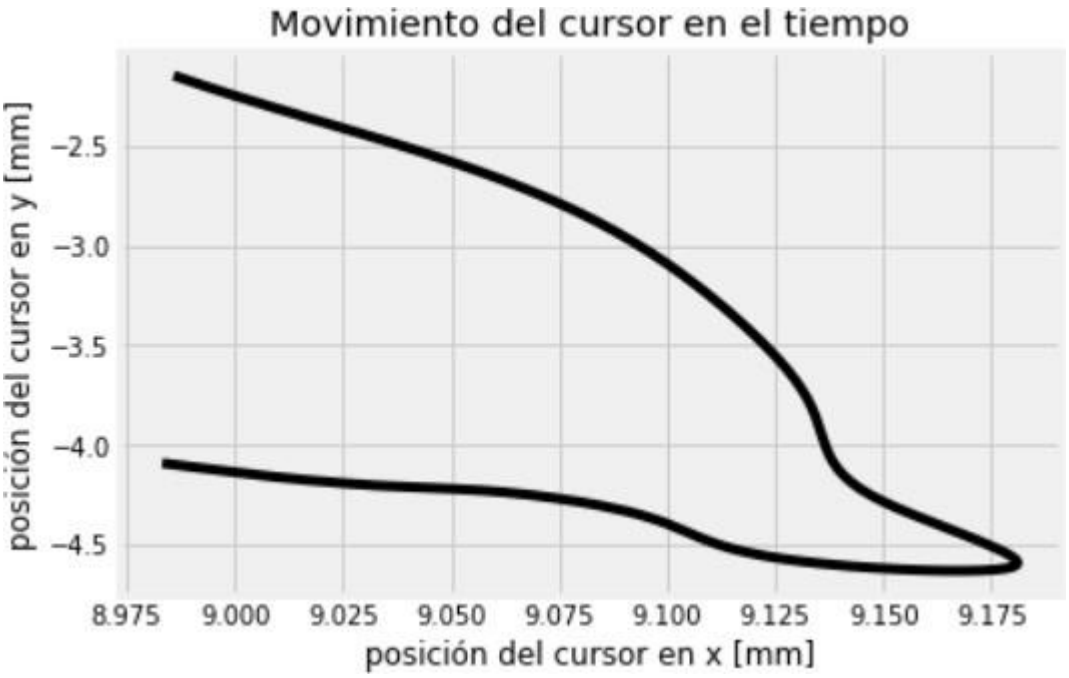

Figura 4.8: Posición X e Y en el tiempo *indy_20170127_03 trial* 3
Fuente: Elaboración propia, (2021).

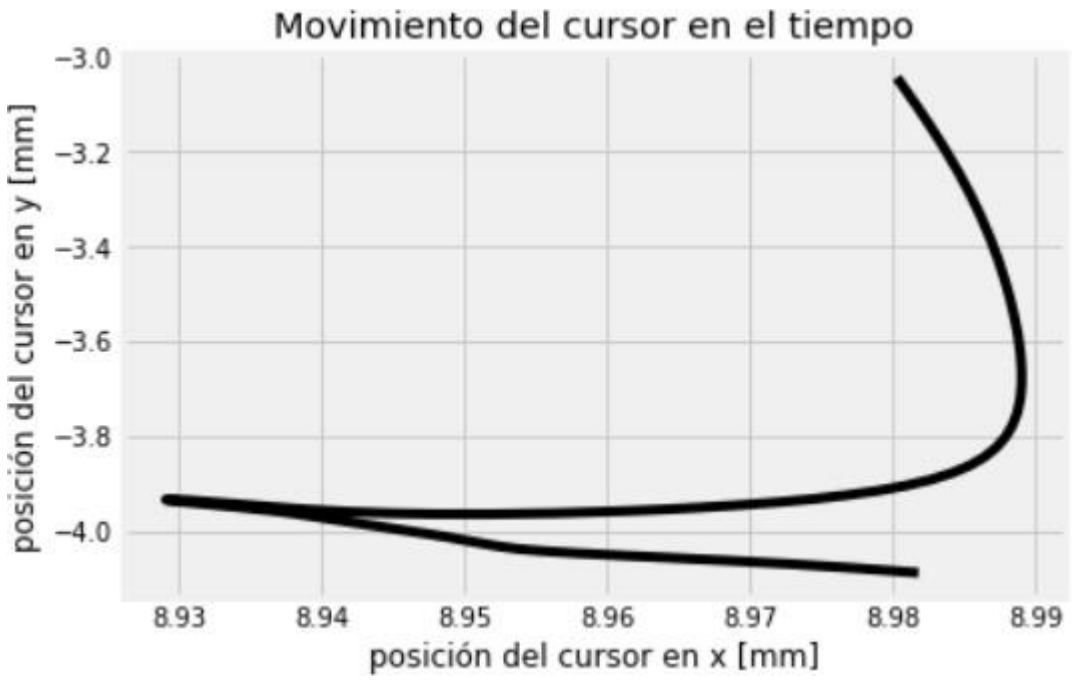

Figura 4.9: Posición X e Y en el tiempo *indy_20170127_03 trial* 4
Fuente: Elaboración propia, (2021).

Figura 4.10: Posición X e Y en el tiempo *indy_20170127_03 trial* 5
Fuente: Elaboración propia. (2021).

Gráficos como estos, permiten tener información visible que dice que los datos están

siendo accedidos y utilizados de manera correcta.

Se logra utilizar la base de datos como se requiere y se obtienen gráficos que según su análisis muestran un correcto funcionamiento.

4.2 ANÁLISIS COMPARATIVO DE MÉTODOS ACTUALES Y NUEVO MÉTODO PROPUESTO

Para realizar la comparativa entre los diferentes métodos se obtuvo información del documento (Ahmadi et al., 2019), donde se exponen valores de los algoritmos LFP-LTSM que se basa en LTSM y es impulsado por LFP, también con el decodificador KF impulsado por LFP y finalmente los métodos de LSTM en base a los vectores MUA y SUA, de los que se consideraron solo los resultados de MUA por ser mejores entre estos 2 últimos.

En la confección de este trabajo se hizo una propuesta propia de MUA-LSTM, la cual como se ha explicado anteriormente, consiste en que como entrada al modelo se ingresan los *spikes rate* y como salida se obtienen las velocidades. A la vez, se realizan varias pruebas y ajustes a los hiperparámetros del modelo, y se llega a los resultados que se muestran en las tablas 4.1 y 4.2.

Comparativa de error cuadrático medio		
Algoritmo	***indy_20170124_01***	***indy_20170127_03***
LFP-LSTM	36.90	43.60
LFP-KF	57.29	65.01
MUA-LSTM	22.08	27.84
MUA-LSTM (propia)	0.08	0.09

Tabla 4.1: Comparativa de error cuadrático medio
Fuente: Elaboración Propia, (2020)

Comparativa correlación valores esperados y reales		
Algoritmo	**Correlación** ***indy_20170124_01***	**Correlación** ***indy_20170127_03***
LFP-LSTM	0.83	0.82
LFP-KF	0.68	0.70
MUA-LSTM	0.86	0.89
MUA-LSTM (propia)	0.99	0.98

Tabla 4.2: Comparativa correlación valores esperados y reales
Fuente: Elaboración Propia, (2020)

La raíz del error cuadrático medio fue tan baja con respecto a los competidores por la implementación que se llevó a cabo, no se tiene acceso al código de la primera implementación de MUA-LSTM, lo que si se puede decir que para esta implementación se realizaron exhaustivas

pruebas para llegar a una configuración óptima contemplando solo los indicadores *RMSE* y correlación.

Dejar los tiempos de respuesta de lado, sirve de mucho para dar todo el énfasis a obtener buenos resultados y no sacrificar otros aspectos en cuando al tiempo de respuesta, ya que no se busca tener resultados para una aplicación en tiempo real.

Para buscar el ajuste entre resultados de *RMSE* o correlación, se decide intentar mejorar el indicador de *RMSE* primero, y en la medida que se pierde correlación, se busca nuevamente recuperar ese parámetro.

Se realizó una prueba adicional a la mencionada en el documento (Ahmadi et al., 2019), donde se unieron las dos bases de datos, y se utilizaron como un único conjunto de datos, y al realizar las pruebas se obtuvieron indicadores de alrededor de un 0.07 *RMSE* y un 0.99 de coeficiente de correlación. Esto nos dice que, con una mayor cantidad y variedad de datos para el entrenamiento de la red neuronal, la predicción puede ser aún mejor.

4.3 REAL CONTRA PREDICCIÓN

El *rasterplot* es un buen indicador para ver la actividad que tuvieron los *spikes* a lo largo del tiempo, para este gráfico solo se tomó el tiempo correspondiente a un *trial* aproximadamente 10 segundos según indica el documento (Keshtkaran et al., 2021).

Como se puede observar en las figuras 4.11 y 4.14, en las partes más oscuras del gráfico fue donde se tuvo mayor actividad cerebral, ya que hubo una mayor activación de neuronas.

En las figuras 4.12 y 4.15 es similar a lo que muestra el *rasterplot*, ya que también muestra actividad cerebral. Se realizó un recuento de los *spikes* contabilizados, según intervalos de tiempo, y fue escalado en un rango de 0 a 1, solo para saber donde hubo mayor actividad cerebral. Por lo anterior, el 1 se puede considerar como el punto de actividad más alta y el 0 la más baja o nula actividad.

Se puede observar que los segundos donde la actividad es más cercana a 1 (picos del gráfico), coincide con las partes más oscuras del *rasterplot*. Lo anterior dice que la actividad cerebral si coincide y se esta determinando correctamente.

Finalmente, en las figuras 4.13 y 4.16 se puede ver el comportamiento de la predicción con respecto a la curva real.

Puede ser que se obtenga un error cuadrático medio muy bajo, pero esto no asegura que se este prediciendo de manera correcta, inclusive esta fue una problemática que se abordó durante la solución del problema. Cuando se ajustaban los hiperpárametros del modelo, se llegó a una configuración con la cual se obtenía un *RMSE* extremadamente bajo, pero era causado por

un sobre ajuste de la curva, y al intentar utilizarla posteriormente para predecir, en verdad se tenía casi una recta que generaba el menor error posible (casi se estaba cayendo en el método de los mínimos cuadrados).

En contraste a lo anterior, se puede ver en estos gráficos que la curva predicción (amarilla) busca imitar el comportamiento de la curva real (morada), se ve que aún falta para que sean completamente iguales, pero esta aproximación es bastante buena en comparación a lo que se había logrado hasta el momento.

A continuación, se pueden ver las figuras descritas anteriormente, las primeras 4.11, 4.12 y 4.13 corresponden a la base de datos *indy_20170124_01*, las siguientes 4.14, 4.15 y 4.16 corresponden a la base de datos *indy_20170127_03*.

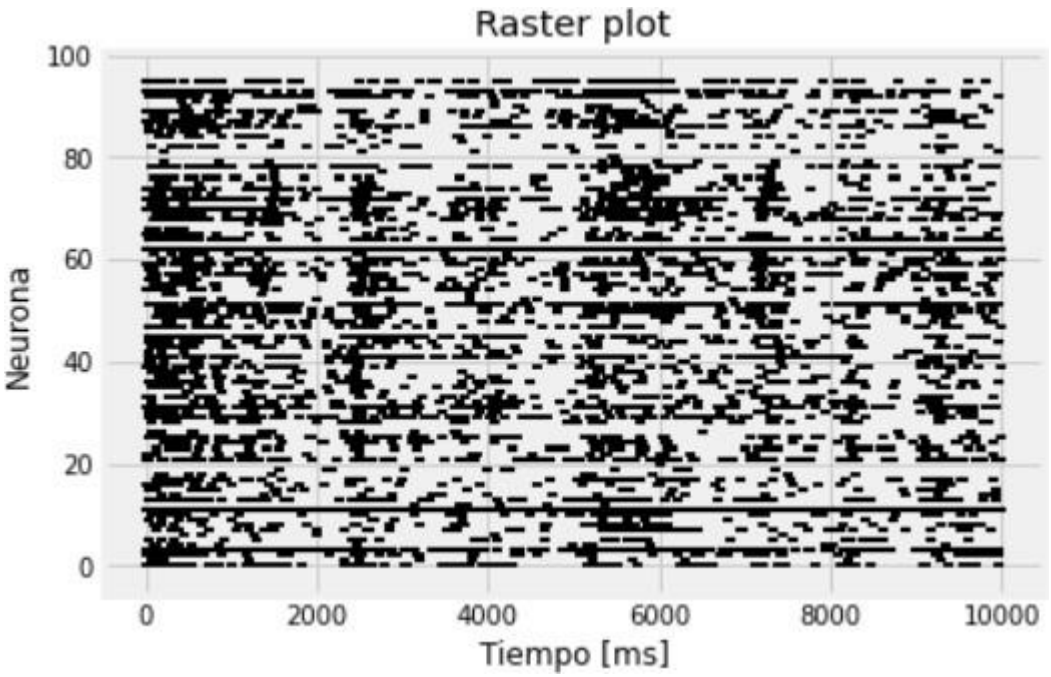

Figura 4.11: Actividad cerebral temporal *indy_20170124_01*
Fuente: Elaboración propia, (2021).

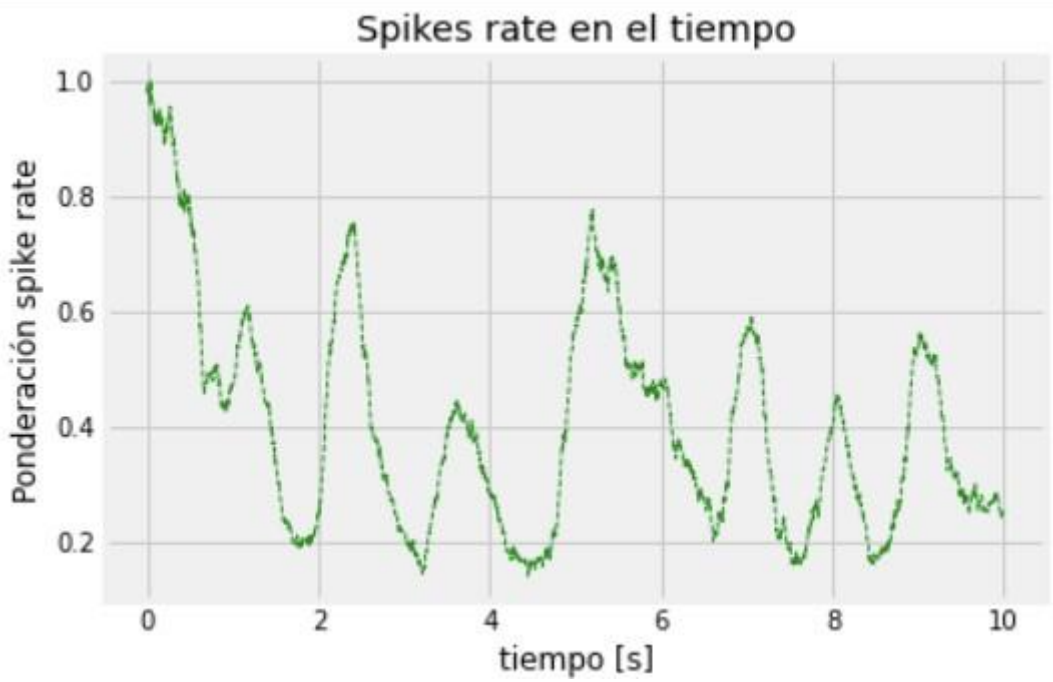

Figura 4.12: Gráfico actividad cerebral en ventana de tiempo de 4[ms] *indy_20170124_01*
Fuente: Elaboración propia, (2021).

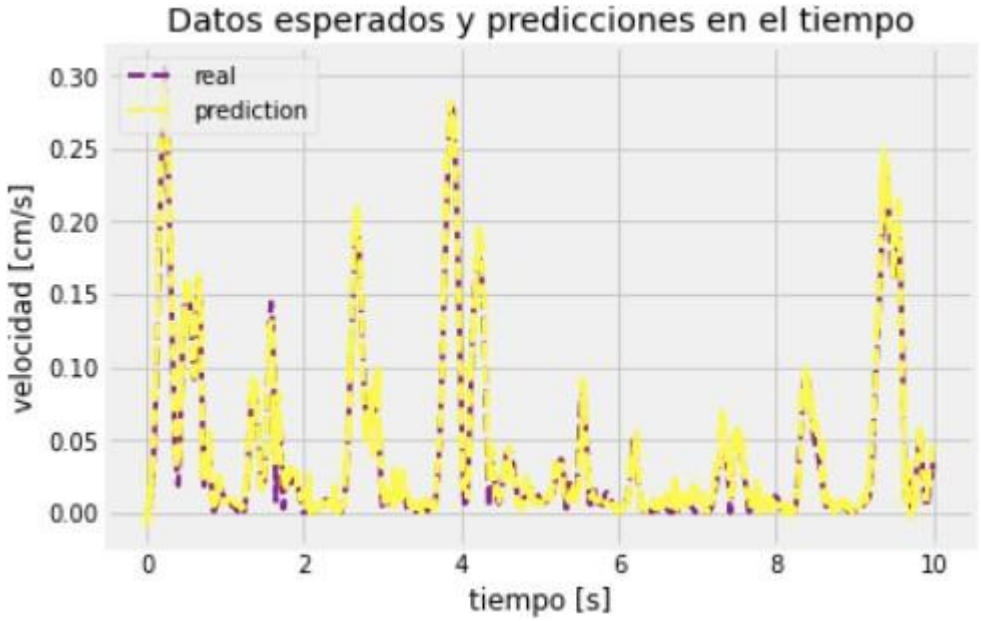

Figura 4.13: Gráfico datos reales contra predicción *indy_20170124_01*
Fuente: Elaboración propia, (2021).

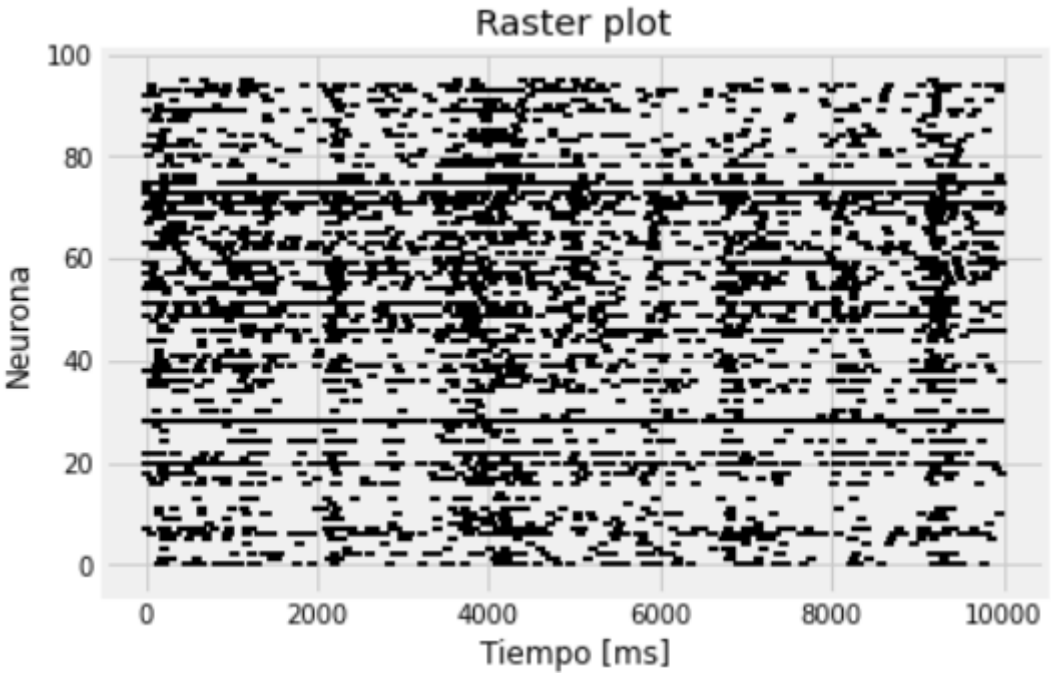

Figura 4.14: Actividad cerebral temporal *indy_20170127_03*
Fuente: Elaboración propia, (2021).

Figura 4.15: Gráfico actividad cerebral en ventana de tiempo de 4[ms] *indy_20170127_03*
Fuente: Elaboración propia, (2021).

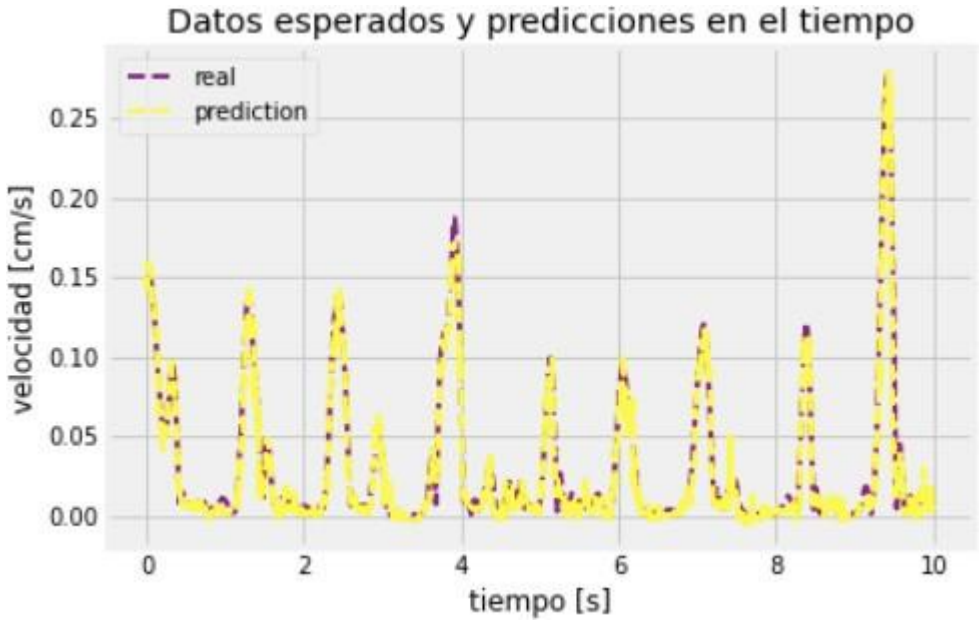

Figura 4.16: Gráfico datos reales contra predicción *indy_20170127_03*
Fuente: Elaboración propia, (2021).

De los gráficos anteriores también se puede observar una clara relación entre los picos de la figura de *spike rate* y la curva real o de predicción, en otras palabras, se observa una fuerte relación entre el movimiento generado y los picos de la actividad cerebral.

4.4 ERROR

Se realizaron gráficos tanto para el error relativo y absoluto. Cada uno de los gráficos se obtuvieron en promedio, por lo tanto, se dividió toda la muestra en *trials* de 10 [s] (el tamaño de esta ventana de tiempo no cambia las conclusiones) y se calculó el error en cada uno de estos casos, posteriormente se sumaron, para luego dividir por el número de *trials*. A continuación, se observan los errores obtenidos en la base de datos *indy_20170124_01*.

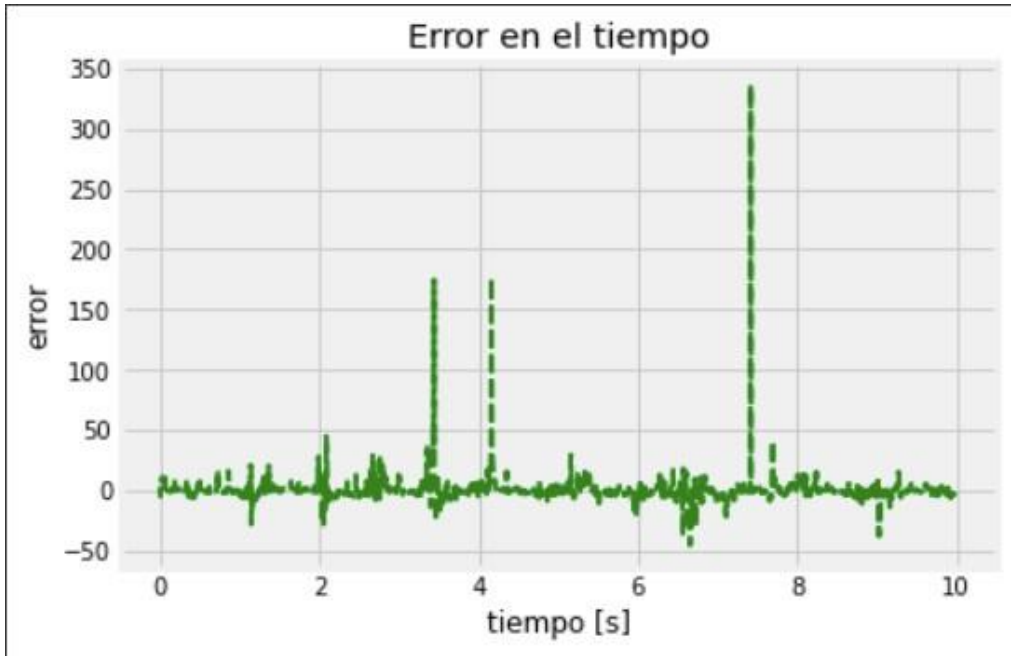

Figura 4.17: Gráfico del error relativo *indy_20170124_01*
Fuente: Elaboración propia, (2021).

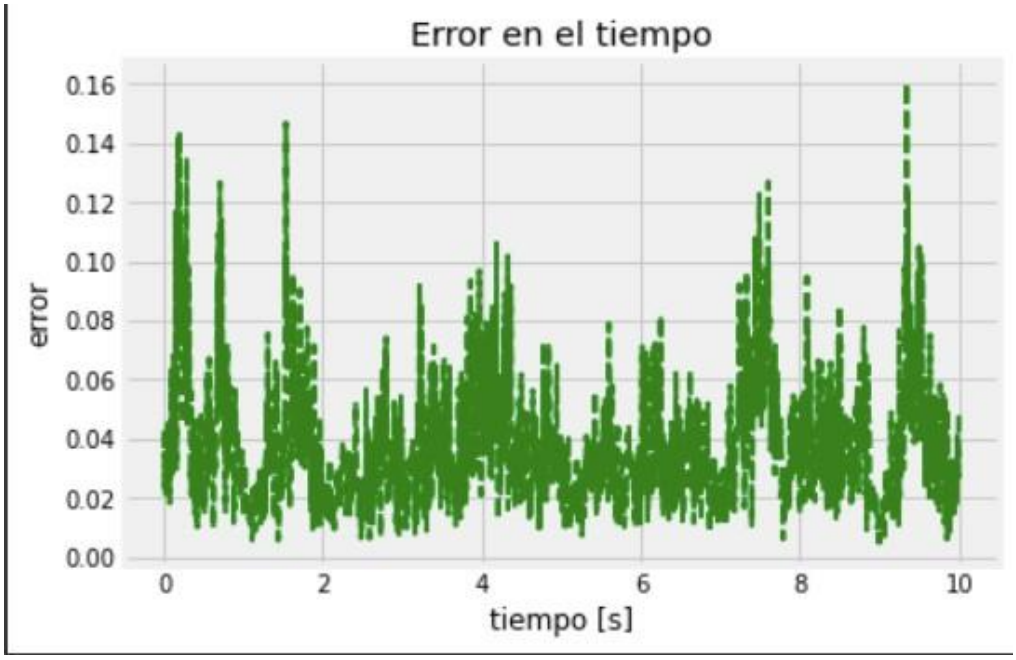

Figura 4.18: Gráfico del error absoluto *indy_20170124_01*
Fuente: Elaboración propia, (2021).

A continuación, se observan los errores obtenidos en la base de datos *indy_20170127_03*.

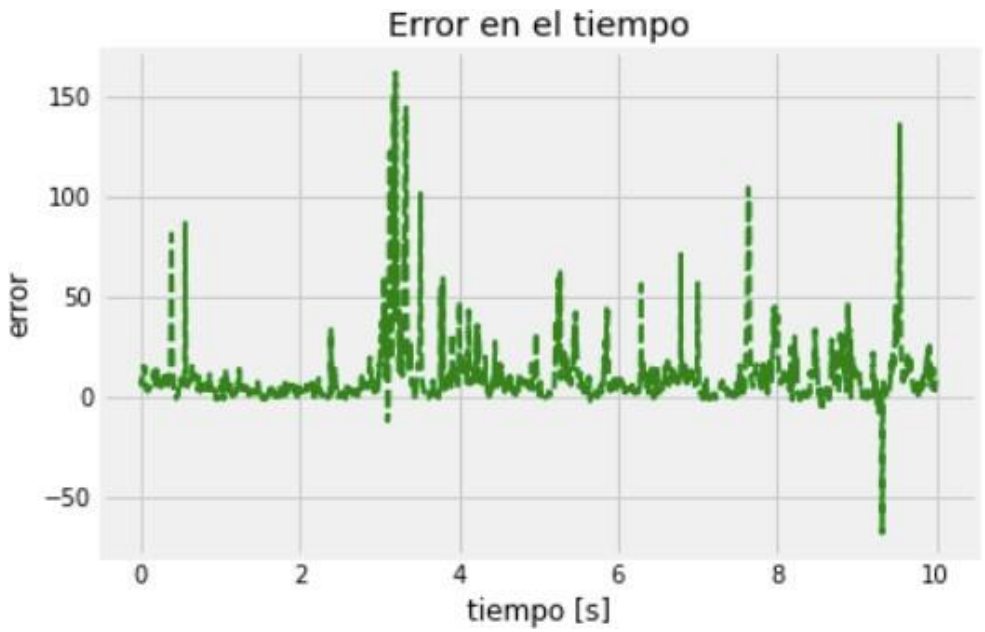

Figura 4.19: Gráfico del error relativo *indy_20170127_03*
Fuente: Elaboración propia, (2021).

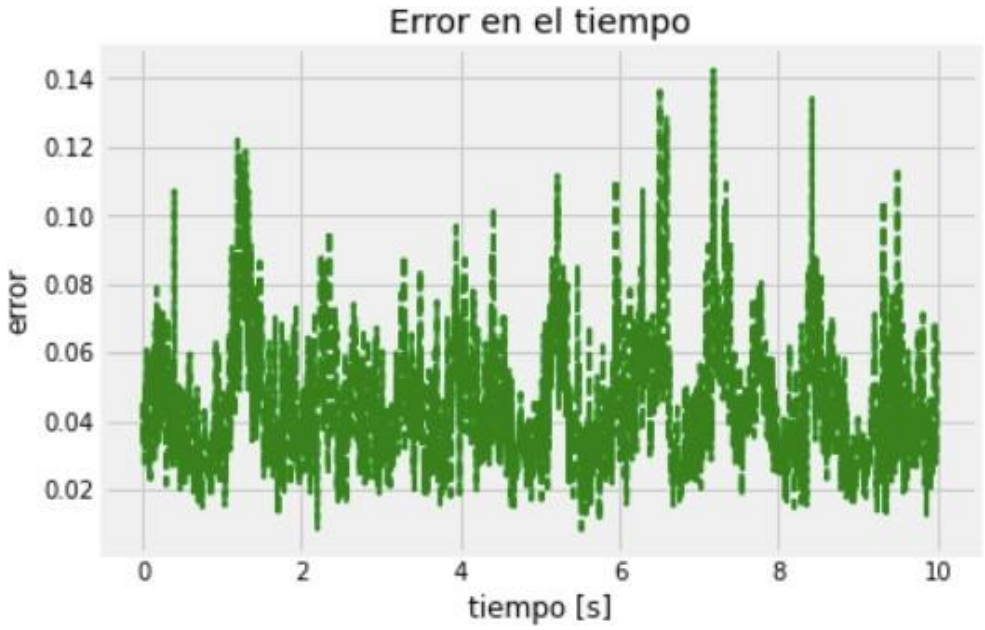

Figura 4.20: Gráfico del error absoluto *indy_20170127_03*
Fuente: Elaboración propia, (2021).

El error relativo es casi constante a lo largo de la predicción, en esos puntos donde se dispara se determinó que sucede dado que el valor de la predicción es mucho más pequeño que el valor real, y esa división produce un número muy grande incluso superando los 120 en el

punto de mayor conflicto.

Por otra parte, con el error absoluto no se observa mayor problema, se ve bajo y constante en todo momento.

4.5 TIEMPOS DE EJECUCIÓN

En la tabla 4.3 se pueden observar los tiempos de ejecución de cada una de las funcionalidades más representativas del trabajo. Estos tiempos dependen de la máquina utilizada para la ejecución (la máquina utilizada para este trabajo fue descrita en la sección ambiente de desarrollo).

Primeramente, se puede ver que los tiempos de importación de bibliotecas y la declaración general de funciones y variables globales son tiempos despreciables. Cuando se desea obtener los *spike rate* y las velocidades esto en promedio esto demora alrededor de 33 minutos, dependiendo de la cantidad de datos que tiene cada base de datos puede demorar más o menos. La preparación de los datos para el modelo, esto es, el ajuste de los vectores para la entrada del modelo también se ejecuta en un tiempo despreciable.

Luego, el entrenamiento del modelo es lo más costoso en cuanto a tiempos de ejecución, en promedio esta tarea dura 3 horas. Estos tiempos se pueden reducir o incrementar dependiendo de la configuración del modelo. Lo bueno, es que la biblioteca *keras*, permite importar y exportar los modelos que se crean en base a sus prestaciones. Como este trabajo se realizó conectado al entorno de *Google Drive*, un modelo puede ser exportado directamente a esta plataforma en un archivo con extensión *h5*. Luego la importación de este modelo solo tarda unos 3 segundos en promedio, lo que es un tiempo despreciable en cuando a lo que tarda el entrenamiento de este.

Finalmente predecir con el modelo puede demorar alrededor de 20 segundos.

Tiempos de ejecución	
Función	**Duración [s]**
Importación de bibliotecas	0
Declaración de funciones	0
Obtener *spikes rate* y velocidad	2000
Preparar datos para el modelo	0
Entrenar modelo	10800
Importar modelo	3
Realizar predicción	20

Tabla 4.3: Tiempos de ejecución
Fuente: Elaboración Propia, (2021)

Lo que se puede concluir en cuanto a todos estos tiempos de ejecución, es que aún

se alejan demasiado de lo que se requiere en la cotidianidad para realizar alguna tarea en base a una interfaz cerebro-máquina en tiempo real con una persona que utiliza un artefacto biónico.

Por otra parte, se han mejorado bastante los resultados que se tienen en este aspecto en cuando a la disminución del *RMSE* y el aumento en la correlación.

Esta implementación puede ser fácilmente acelerada utilizando componentes más potentes en cuanto a velocidad de procesamiento, por ejemplo, *GPU*s y cambiando la utilización de listas y vectores de *numpy* a *cupy*.

La biblioteca *cupy* utiliza directamente el procesamiento de una *GPU*, por lo tanto, es mucho más veloz de lo que logra una *CPU*.

Realizando los cambios anteriormente descritos, el problema ”tiempo” sería mucho más irrelevante, aunque, probablemente se necesite otro tipo de trabajo para llegar a resultados realmente aceptables en este aspecto.

CAPÍTULO 5. CONCLUSIONES

5.1 DESARROLLO DEL SISTEMA

Se puede comenzar por la base de datos, fue un trabajo complicado el poder acceder a los datos, ya que no es como el uso de *SQL*, este tipo de datos *H5py* tiene algo similar a una capa de encriptación, la que requirió una investigación adicional para poder acceder a los datos desde *Python*.

Además, dado el tamaño de vectores que tenían algunos de estos datos, el acceder a ellos podía ser más complejo aún. En cuanto a lo mismo, no sé puede dejar de destacar que muchas veces se tuvo que realizar *reshape* de varios datos, ya que, venían invertidos a lo que se necesitaba y/o facilitaba el desarrollo de este trabajo.

Al comenzar este trabajo no sé tenía conocimiento de que se había implementado un método con las mismas características del que se ofrece en este documento, eso quizás fue un traspié, pero, permitió tener en consideración que ese era el mejor método utilizado actualmente. Por lo anterior se reformularon los objetivos personales, buscando por lo menos obtener los mismos resultados que se informan sobre ese método, y finalmente al ver dichos resultados, se pudo observar que estos fueron bastante mejores.

Se plantea una nueva mejora que es de irrefutable consideración, al unir los conjuntos de datos de más de una base de datos, el modelo es mejor predictor aún.

Fue un problema la utilización de la base de datos con *Python*, se optó por utilizar este lenguaje ya que por medio de bibliotecas como *Keras*, facilita el trabajo más complejo, ya que permite un manejo sencillo del modelo basado en redes neuronales.

Al recorrer y almacenar los conjuntos de datos, se tenían distinto número de mediciones, por ejemplo, en los datos de *spikes*, ya que puede ser que una neurona no se active durante toda la prueba y otra este constantemente en funcionamiento, por lo tanto, esas medidas finalmente tendrán diferente *size* y eso complejizó el trabajo.

En cuanto a la comparativa con otros métodos, pudo haber sido un problema el buscar información acerca de diversos métodos y ver cual sería el más eficiente, pensando en unificar métricas de comparación y sabiendo que estarían utilizando diversos conjuntos de datos (aunque fuesen de la misma base de datos). Aquí la solución fue gracias al documento (Ahmadi et al., 2019), ya que tenían establecidas las mediciones promedio de los métodos más eficientes.

5.2 OBJETIVOS ESPECÍFICOS

A lo largo de todo el documento se puede observar como se fueron cumpliendo cada uno de los objetivos, esto se puede ver en detalle en el capítulo de Resultados y Discusión, donde se realizó un desglose en base a cada uno de los objetivos de este trabajo. Haciendo un recuento el análisis de la base de datos, se pudo pensar que era una tarea sencilla, tomó más tiempo del considerado, pero se llevo a cabo finalmente de manera exitosa.

Sobre el método como tal, se pudo ver que es más eficiente que los otros métodos existentes, por lo tanto, ese objetivo se cumple de manera exitosa. Y para realizar esta comparación antes se debió cumplir la investigación de los otros métodos y generar la tabla comparativa.

Luego de todo esto se cumple el objetivo general del proyecto, y se deja planteado un interesante descubrimiento, mientras más diversos y mayor es el número de datos para el entrenamiento, el modelo se hace un mejor predictor.

5.3 HIPÓTESIS

"se puede implementar un método basado en Deep Learning que sea más efectivo en la predicción de movimiento, que los métodos utilizados actualmente"

Finalmente, la respuesta es sí, los resultados avalan esta premisa, se pudo generar un implementar basado en una red *LSTM* el cual provee resultados mejores en cuanto a predicción que cualquier método expuesto a la fecha, esto solo en cuanto a *RMSE* e índice de correlación, en cuanto a tiempos de ejecución no se realizó la comparación, pero, a lo largo de este documento se indicó que el factor tiempo no fue considerado, solo se buscó la mejor predicción posible y se logro obtener.

5.4 FUTURO DE LA INVESTIGACIÓN

Para el futuro se propone trabajar en base a redes neuronales adversarias, este tipo de implementaciones, tienen resultados muy buenos, y pueden ir mejorando en el tiempo. La red utilizada en este trabajo solo se entrena una vez, no es un proceso iterativo, y probablemente al realizar un entrenamiento muy largo se termine sobre ajustando el modelo a un conjunto de datos

en particular. Las redes adversarias son un poco más ”inteligentes” en este sentido, aprendiendo y descantando posibilidades muy rápidamente.

GLOSARIO

Amputación: Separación de un miembro, normalmente por un medio quirúrgico.

Dataset: Conjunto de datos que almacenan información en forma de par ordenado, normalmente tienen un alto número de datos.

Deep Learning: Tienen la capacidad de hacer que los sistemas basados en *AI* pueden no solo aprender conceptos, sino también pueden comprender contextos y entornos bastante complejos.

Inteligencia artificial: Se refiere al estudio, al desarrollo y a la aplicación de técnicas informáticas que les permiten a las computadoras imitar ciertas habilidades propias de la inteligencia humana.

Interfaz: Dispositivo capaz de transformar las señales generadas por un aparato en señales comprensibles por otro.

Machine Learning: Es la acción de establecer un conjunto de reglas que permitan a la maquina aprender a través de ellas.

Medición: Para este documento una medición es cada señal que cumpla con ciertos requerimientos para poder ser almacenada como válida.

Método: Modo ordenado y sistemático de proceder para llegar a un resultado o fin determinado.

Metodología: Parte de la lógica que estudia los métodos.

Métrica: Es un punto de clasificación el cual sirve para poder comparar.

Movimiento: Para estos propósitos un movimiento tiene un tiempo de iniciación t0 y uno de finalización tf que lo hacen ser un movimiento válido.

Neurociencia: Estudio del comportamiento del cerebro.

Prótesis: Sustitución de una parte del cuerpo, por un objeto que reproduce en cierto grado el comportamiento de dicha parte.

Redes adversarias: Dos redes las cuales trabajan de manera conjunta para entrenarse mutuamente y generar conocimiento con respecto a un área de investigación.

Registro: Para este documento será cada medición que se lleve a cabo.

Señal: Para estos propósitos será una honda cerebral.

Situación de discapacidad: En relación con una condición física, mental, intelectual, etc. y algún contexto, un ser vivo presenta problemas para desenvolverse de manera plena.

REFERENCIAS BIBLIOGRÁFICAS

Ahmadi, N., Constandinou, T. G., & Bouganis, C.-S. (2019). Decoding hand kinematics from local field potentials using long short-term memory (lstm) network. In *2019 9th International IEEE/EMBS Conference on Neural Engineering (NER)*, 415–419.

Aso, U. (2021). Electrofisiología: qué es y cómo se investiga en ella.
URL https://psicologiaymente.com/neurociencias/electrofisiologia

Calvo, D. (2018). Función de coste – redes neuronales.
URL https://www.diegocalvo.es/funcion-de-coste-redes-neuronales/

Cossio, E. G., & Gentiletti, G. G. (2008). Interfaz cerebro computadora (icc) basada en el potencial relacionado con eventos p300: análisis del efecto de la dimensión de la matriz de estimulación sobre su desempeño. *Revista Ingenieria Biomedica*, *2*(4), 26–33.

Eckhardt, K. (2018). Choosing the right hyperparameters for a simple lstm using keras.
URL https://towardsdatascience.com/choosing-the-right-hyperparameters-for-a-simple-lstm-using

Faust, O., Hagiwara, Y., Hong, T. J., Lih, O. S., & Acharya, U. R. (2018). Deep learning for healthcare applications based on physiological signals: A review. *Computer methods and programs in biomedicine*, *161*.

Gargantilla, P. (2019). ¿qué es el método científico? estos son sus cinco pasos.
URL https://www.abc.es/ciencia/abci-metodo-cientifico-estos-cinco-pasos-201902170129_noticia.html

Garzón, J. (2018). Cómo usar redes neuronales (LSTM) en la predicción de averías en las máquinas.
URL https://blog.gft.com/es/2018/11/06/como-usar-redes-neuronales-lstm-en-la-prediccion-de-

Gómez Figueroa, L. J. (2016). *Análisis de señales EEG para detección de eventos oculares, musculares y cognitivos* (Doctoral dissertation, Industriales).

Goodnight, J. (2021). Inteligencia artificial.
URL https://www.sas.com/es_cl/insights/analytics/what-is-artificial-intelligence.html

Guacho Rivera, D. D. (2018). *Implementación de un prototipo de prótesis con control muscular para mejora del movimiento y agarre de objetos, aplicada a personas con muñón o malformación en los dedos de la mano.*. Master's thesis, Escuela Superior Politécnica de Chimborazo.

Iglesias, J. (2020). Reconocimiento de escritura con keras: Perceptrón vs cnn.
URL https://www.paradigmadigital.com/dev/reconocimiento-escritura-keras-perceptron-vs-cnn/

Julián, G. (2014). Las redes neuronales: qué son y por qué están volviendo.
URL https://www.xataka.com/robotica-e-ia/las-redes-neuronales-que-son-y-por-que-estan-volvie

Keshtkaran, M. R., Sedler, A. R., Chowdhury, R. H., Tandon, R., Basrai, D., Nguyen, S. L., Sohn, H., Jazayeri, M., Miller, L. E., & Pandarinath, C. (2021). A large-scale neural network training framework for generalized estimation of single-trial population dynamics. *bioRxiv*.

Knoepfler, P. (2015). Will bioengineered body parts cost an arm and a leg?
URL https://ipscell.com/2015/06/bioengineeringcost/

López, R., Aguilar, H., Salazar, S., Lozano, R., & Torres, J. A. (2014). Modelado y control de un

exoesqueleto para la rehabilitación de extremidad inferior con dos grados de libertad. *Revista iberoamericana de automática e informática industrial*, *11*(3), 304–314.

López, B. S. (2019). Regresión lineal.
URL https://www.ingenieriaindustrialonline.com/pronostico-de-la-demanda/regresion-lineal/

Makin, J. G., O'Doherty, J. E., Cardoso, M. M., & Sabes, P. N. (2018). Superior arm-movement decoding from cortex with a new, unsupervised-learning algorithm. *Journal of neural engineering*, *15*(2), 026010.

Marte, H. (2019). Interfaz cerebro computador: Controlar cosas con la mente.
URL https://neuro-class.com/cerebro-computador-controlar-cosas-con-la-mente/

Melgar, J. (2018). 4 aplicaciones del deep learning en la vida cotidiana.
URL https://ilifebelt.com/aplicaciones-del-deep-learning/2018/02/

Miralles, M., & Giuliano, G. (2008). Biónica: eficacia versus eficiencia en la tecnología natural y artificial. *Scientiae Studia*, *6*(3), 359–369.

Molina, M. (2020). La distancia más corta. el método de los mínimos cuadrados.
URL https://anestesiar.org/2020/la-distancia-mas-corta-el-metodo-de-los-minimos-cuadrados/

Montagud, N. (2019). Redes neuronales profundas: qué son y cómo funcionan.
URL https://psicologiaymente.com/cultura/redes-neuronales-profundas

NIH (2019). Sobre de la neurociencia.
URL https://espanol.nichd.nih.gov/salud/temas/neuro/informacion

Olivares Carrillo, C. Y. (2017). Diseño y construcción de una interfaz cerebro computadora para el control de una silla de ruedas como ayuda a personas con discapacidad motriz (Master's thesis, Universidad del Norte).

O'Doherty, J. E., Cardoso, M., Makin, J., & Sabes, P. (2017). Nonhuman primate reaching with multichannel sensorimotor cortex electrophysiology. *Zenodo http://doi. erg/10.5281/zenodo*, *583331*.

Peñaloza, C. (2019). Crean interfaz cerebro-computadora para aumentar las capacidades fÍsicas de los humanos.
URL https://www.elhospital.com/temas/Crean-interfaz-cerebro-computadora-para-aumentar-las-cap131334

Pino, S. (2012). Bases neurofisiológicas.
URL https://www.slideshare.net/silgumpinzac/bases-neurofisiolgicas-del-eeg-parte-1

Pruthi, S. (2020). Eeg (electroencefalograma).
URL https://www.mayoclinic.org/es-es/tests-procedures/eeg/about/pac-20393875

Purves, D., Augustine, G. J., Fitzpatrick, D., Hall, W. C., LaMantia, A. S., McNamara, J. O., & White, L. (2014). Neuroscience, 2008. De Boeck, Sinauer, Sunderland, Mass, 15, 16. Robert, M., & Andrés, R. (2014). Estudio de la biogénesis y expresión de mirrorrnas en el transcriptoma de mamíferos.

Sharma, A. (2020). Differences between machine learning deep learning.
URL https://www.datacamp.com/community/tutorials/machine-deep-learning

Solera-Ramírez, Á. (2003). Filtro de Kalman. Banco Central de Costa Rica.

Soltani, M., Jain, S., & Sambasivan, A. (2019). Unsupervised demixing of structured signals from

their superposition using gans.

Torres, J. (2021). Redes neuronales recurrentes.
URL https://torres-ai.medium.com/redes-neuronales-recurrentes-13fb20dd8a6f

APÉNDICE A. CÓDIGO

A.1 ADQUISICIÓN DE DATOS

```
def get_chan_names(file):
  chan_names = file.get('chan_names')
  chan_names = file['chan_names'][:10]
  chan_names = chan_names[()][0]
  for i in range(chan_names.size):
    chan_name = file[chan_names[i]]
    chan_name_array = chan_name[()]
    chan_name_array = chan_name_array.reshape((1, 6))
    chan_name_list = chan_name_array.tolist()[0]
    chan_name_str = ''.join(chr(i) for i in chan_name_list)
    chan_names[i] = chan_name_str
  return chan_names
```

Figura A.1: Obtener nombre vectores
Fuente: Elaboración propia (2020).

```
def get_times(file):
  t = file.get('t')
  t = file['t'][:10]
  t = t[()][0]
  return t
```

Figura A.2: Obtener tiempos
Fuente: Elaboración propia (2020).

```
def get_cursor_pos(file):
  cursor_pos = file.get('cursor_pos')
  cursor_pos = file['cursor_pos'][:10]
  x = cursor_pos[()][0]
  y = cursor_pos[()][1]
  cursor_pos = []
  cursor_pos.append(x)
  cursor_pos.append(y)
  return cursor_pos
```

Figura A.3: Obtener posiciones cursor
Fuente: Elaboración propia (2020).

```
def get_target_pos(file):
  target_pos = file.get('target_pos')
  target_pos = file['target_pos'][:10]
  x = target_pos[()][0]
  y = target_pos[()][1]
  target_pos = []
  target_pos.append(x)
  target_pos.append(y)
  return target_pos
```

Figura A.4: Obtener posiciones objetivo
Fuente: Elaboración propia (2020).

```
def get_finger_pos(file):
    finger_pos = file.get('finger_pos')
    finger_pos = file['finger_pos'][:10]
    x = finger_pos[()][0]
    y = finger_pos[()][1]
    z = finger_pos[()][2]
    finger_pos = []
    finger_pos.append(x)
    finger_pos.append(y)
    finger_pos.append(z)
    return finger_pos
```

Figura A.5: Obtener posición dedo
Fuente: Elaboración propia (2020).

```
def get_spikes(file):
  spikes = file.get('spikes')
  spikes = file['spikes'][:10]
  spikes_list = []
  for i in range(n_channels):
    spike = []
    for j in range(5):
      spikes_n = spikes[()][j]
      spike_aux = file[spikes_n[i]]
      spike_aux = spike_aux[()]
      spike_aux = spike_aux[0]
      list_aux = spike_aux.tolist()
      spike.append(list_aux)
    spikes_list.append(spike)
  for spike in spikes_list:
    for idx, s in enumerate(spike):
      if not(isinstance(s, list)):
        spike[idx] = []
  return spikes_list
```

Figura A.6: Obtener *spikes*
Fuente: Elaboración propia (2020).

```
def get_wfs(file):
  wfs = file.get('wf')
  wfs = file['wf'][:10]
  wfs = wfs[()]
  wfs_list = []
  for i in range(n_channels):
    wf = []
    for j in range(5):
      wf_aux = file[wfs[j][i]]
      wf_aux = wf_aux[()]
      if (len(wf_aux) == 64):
        wf_aux = wf_aux.ravel()
        wf_aux = wf_aux.reshape(int(len(wf_aux)/64), 64, order='F')
      else:
        wf_aux = np.array([])
      wf.append(wf_aux)
    wfs_list.append(wf)
  return wfs_list
```

Figura A.7: Obtener frecuencias
Fuente: Elaboración propia (2020).

A.2 TRABAJO DE DATOS

```
def get_mua_data(spikes, wfs):
  mua_data = []
  pos = 0
  while pos < len(spikes):
    wf = np.array(wfs[pos][0])
    boolean_indexes = wf > 0.5
    neuron_data = np.array(spikes[pos][0], dtype="float32")
    mua_data.append(neuron_data[boolean_indexes])
    pos = pos+1
  return np.array(mua_data)
```

Figura A.8: Obtener datos del vector *MUA*
Fuente: Elaboración propia (2020).

```
def get_rate_spikes(mua_data, t):
  spikes_rate = []
  banwidth = 0.1
  for neuron in mua_data:
    neuron_rate = np.array([], dtype="float32")
    time = t[0]
    for time in t:
      final_time = time+banwidth
      positions = time < neuron
      data = neuron[positions]
      positions = data < final_time
      n_spikes = np.count_nonzero(positions == True)
      neuron_rate = np.append(neuron_rate, (n_spikes/banwidth))
    spikes_rate.append(neuron_rate)
  spikes_rate = np.array(spikes_rate)
  shape_x = spikes_rate.shape[1]
  shape_y = spikes_rate.shape[0]
  spikes_rate = spikes_rate.ravel()
  spikes_rate = spikes_rate.reshape(shape_x, shape_y, order='F')
  return spikes_rate
```

Figura A.9: Obtener *rate spikes*
Fuente: Elaboración propia (2020).

```
def get_velocity(x, y):
  dt = 0.004
  v_x = np.diff(x) / dt
  v_y = np.diff(y) / dt
  velocity = np.sqrt((v_x * v_x) + (v_y * v_y))
  velocity = np.append(velocity[0], velocity)
  return velocity
```

Figura A.10: Obtener velocidades
Fuente: Elaboración propia (2020).

```
def get_spikes_velocity(file_name, time):
  (x,y,spikes,t,wfs) = get_data(file_name)
  if time == 0:
    time = t[len(t)-1]
  wfs = wfs_average(wfs)
  mua_data = get_mua_data(spikes, wfs)
  spikes_rate = get_rate_spikes(mua_data, t)
  velocity = get_velocity(x, y)
  velocity = np.split(velocity, [spikes_rate.shape[0]])[0]
  return (spikes_rate, velocity)
```

Figura A.11: Obtener *rates spikes* y velocidad
Fuente: Elaboración propia (2020).

```
def load_neural_data():
  neural_data = []
  min = 1500
  max = 1501
  for pos in range(len(spikes)):
    data = spikes[pos][0]
    data = np.array(data)
    positions = data < max
    data = data[positions]
    positions = min < data
    neural_data.append(data[positions])
  neural_data = np.array(neural_data)
  return neural_data
```

Figura A.12: Calcular datos gráfico neural
Fuente: Elaboración propia (2020).

```
def load_spikes_per_time(min_time, max_time):
  (x, y, spikes, t, wfs) = get_data(file_name)
  wfs = wfs_average(wfs)
  mua_data = get_mua_data(spikes, wfs)
  positions = min_time < t
  spikes_rate = get_rate_spikes(mua_data, t[positions])
  spikes_per_time = []
  for sr in spikes_rate:
    spikes_per_time.append(np.sum(sr) / n_channels)
  spikes_per_time = np.array(spikes_per_time)
  spikes_per_time = spikes_per_time[1:]
  spikes_per_time = spikes_per_time.reshape(len(spikes_per_time), 1)
  scaler = MinMaxScaler(feature_range=(0, 1))
  spikes_per_time = scaler.fit_transform(spikes_per_time)
  return spikes_per_time
```

Figura A.13: Calcular *spikes* vs tiempo
Fuente: Elaboración propia (2020).

A.3 MODELO

```
spikes_rate = np.concatenate((spikes_rate_1, spikes_rate_2), axis=0)
velocity = np.concatenate((velocity_1, velocity_2), axis=0)
```

Figura A.14: Unir bases de datos
Fuente: Elaboración propia (2020).

```
(x_train, x_validation, x_test, y_train, y_validation, y_test) = split_data(spikes_rate, velocity)
x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], 1))
x_validation = np.reshape(x_validation, (x_validation.shape[0], x_validation.shape[1], 1))
```

Figura A.15: Preparar datos para modelo
Fuente: Elaboración propia (2020).

```
def fit_model(epochs, hidden_nodes, batch_size, units, units_2, dropout, x_train, y_train, x_validation, y_validation):
  model = Sequential()
  model.add(LSTM(units = units, return_sequences=True, input_shape=(x_train.shape[1], 1)))
  model.add(LSTM(units = units))
  model.add(Dropout(dropout))
  model.add(Dense(units = units_2))
  model.compile(optimizer='adam', loss='mean_squared_error', metrics=['acc'])
  model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs, validation_data=(x_validation, y_validation))
  return model
```

Figura A.16: Entrenar el modelo
Fuente: Elaboración propia (2020).

```
model = fit_model(3, 100, 100, 100, 1, 0.2, x_train, y_train, x_validation, y_validation)
x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], 1))
predictions = model.predict(x_test)
predictions = predictions.reshape(len(predictions))
```

Figura A.17: Obtener predicción
Fuente: Elaboración propia (2020).

```
mse=np.mean(((predictions - y_test) ** 2))
rmse=np.sqrt(mse)
```

Figura A.18: Calcular *RMSE*
Fuente: Elaboración propia (2020).