Universidad de Costa Rica
Facultad de Ingeniería
Escuela de Ingeniería Eléctrica

TRABAJO FINAL DE GRADUACIÓN
LICENCIATURA

Implementación de un clasificador de señales de Potencial de Campo Local (LFP) utilizando redes neuronales

Por:

Jose Pablo Ávila López

Ciudad Universitaria Rodrigo Facio, Costa Rica

Julio del 2021

Implementación de un clasificador de señales de Potencial de Campo Local (LFP) utilizando redes neuronales

Por:
Jose Pablo Ávila López

Sometido a la Escuela de Ingeniería Eléctrica
de la Facultad de Ingeniería
de la Universidad de Costa Rica
como requisito parcial para optar por el grado de:
LICENCIADO EN INGENIERÍA ELÉCTRICA

Aprobado por el Tribunal:

Dr. rer. nat. Francisco Siles Canales
Representante de la Dirección,
Escuela de Ingeniería Eléctrica

Ing. Esteban Alonso Ortiz Cubero
Director, Comité Asesor

Ing. Luis Alfonso Castro Leiva MSc.
Miembro, Comité Asesor

Ing. Pablo Enrique Deliyore Vega MSc.
Miembro, Comité Asesor

Ing. Lochi Yu Lo, PhD.
Miembro del Tribunal

Índice general

Índice de figuras **vii**

Índice de cuadros **xi**

Nomenclatura **xiii**

1 Introducción **1**
1.1 Planteamiento del Problema 1
1.2 Antecedentes 2
1.3 Justificación 4
1.4 Hipótesis 5
1.5 Objetivos 6
1.6 Alcances 6
1.7 Metodología 7
1.8 Procedimiento de evaluación 8

2 Marco Teórico **9**
2.1 Actividad Cerebral 9
2.2 Banco de datos a utilizar 18
2.3 Inteligencia Artificial y Aprendizaje Automático 22
2.4 Redes Neuronales 25

3 Desarrollo **41**
3.1 Selección de las grabaciones del banco de datos 41
3.2 Preparación de los datos y extracción de características 47
3.3 Implementación de las redes neuronales 65

4 Análisis de Resultados **69**
4.1 Preparación de datos de entrada 69
4.2 Comparación de predicciones ante variaciones en los parámetros 87

5 Conclusiones y recomendaciones **107**
5.1 Conclusiones 107
5.2 Recomendaciones 107

Bibliografía **109**

Apéndices **111**

A Documentación del código **113**

Índice de figuras

2.1 Potencial de Acción. (Ramos, 2019) 10
2.2 Circuito principal del hipocampo. (Buzsaki, 2011) 12
2.3 Actividad neuronal contra ángulo de dirección a la que se dirige la cabeza. (Taube, 2009) 13
2.4 a) Modelo de circuito eléctrico de grabación extracelular. b) Atenuación de componentes de alta frecuencia en el medio extracelular. (Gibson et al., 2012) 16
2.5 Envolvente (verde) y promedio (azul) de la señal LFP del experimento de (Islam et al., 2017) señalando las 5 ventanas de tiempo (líneas punteadas) para extraer los 5 puntos en los distintos canales y mano del evento. 17
2.6 Ejemplo del tipo de sondas de adquisición de utilizadas. NeuroNexus (2020) 20
2.7 Un perceptrón con tres unidades de entrada (neuronas) y una neurona de salida (Vasilev et al., 2019). 25
2.8 Red neuronal con una capa escondida. (Vasilev et al., 2019) 26
2.9 Proceso de aprendizaje por descenso de gradiente (Vasilev et al., 2019). 27
2.10 Escenarios de convergencia con distintos valores de *learning rate* η comparado con el valor de η óptimo (Duda et al., 2012) 28
2.11 Descenso de gradientes por mini-lotes (Rojas, 2019) 29
2.12 Conexión entre capas de una Red Neuronal Convolucional (Nielsen, 2015). 31
2.13 Capa Convolucional. (Calvo, 2015) 32
2.14 Capa de Reducción. (Calvo, 2015) 33
2.15 Red Neuronal Convolucional. (Calvo, 2015) 34
2.16 Red Neuronal Recurrente. (Vasilev et al., 2019) 35
2.17 Celda de una red LSTM. (Vasilev et al., 2019) 37

3.1 Representación del Experimento 42
3.2 Gráfico de barras de cantidad de *Spikes* por tipo de neurona . . . 45
3.3 Primer segundo de señales de LFP por canal de la sesión 765 visualizadas con el software Neuroscope. 46
3.4 Primer segundo de señales de LFP por canal de la sesión 771 visualizadas con el software Neuroscope. 47

3.5 Diagrama de cajas de los datos de LFP de la sesión 771 a su frecuencia original del archivo .eeg. 49
3.6 Primeros 100 segundos de los datos de los canales 0, 1 (en amarillo), 96, 97 y 98 (en azul). 50
3.7 Diagrama de cajas de los datos de LFP de la sesión 771 a 39.06Hz, frecuencia de adquisición de la posición de los LEDs en el sujeto. . 54
3.8 Ángulos de la sesión 771 a 39.06Hz. 58
3.9 Interpolación Lineal de Ángulos de la sesión 771 con rotaciones con cruce por 0°. 58
3.10 Interpolación Lineal de Ángulos de la sesión 771. 60
3.11 Interpolación por ángulo más corto de Ángulos de la sesión 771. . 61

4.1 Gráfica de las señales de LFP de los canales 0 y 97 de la sesión 771 a su frecuencia original (1250Hz). 69
4.2 Gráfica de las señales de LFP de los canales 0 y 97 de la sesión 771 a su frecuencia original (1250Hz). 70
4.3 Gráfica de las señales de LFP a su frecuencia original (1250Hz) durante los primeros 10 segundos del canal 70 de la sesión 771. . . 71
4.4 Gráfica de las señales de LFP a su frecuencia original (1250Hz) durante los primeros 10 segundos del canal 70 de la sesión 765. . . 71
4.5 Diagrama de cajas de los datos de LFP de la sesión 765 a su frecuencia original del archivo .eeg. 72
4.6 Gráfica de los ángulos de la sesión 765 a su frecuencia original (39.06Hz). 73
4.7 Gráfica de los ángulos de la sesión 771 a su frecuencia original (39.06Hz). 73
4.8 Gráfica de los ángulos a su frecuencia original (39.06Hz) en una ventana de 20 segundos al minuto 6 de la sesión 765. 74
4.9 Gráfica de los ángulos a su frecuencia original (39.06Hz) en una ventana de 20 segundos al minuto 6 de la sesión 771. 74
4.10 Señal de LFP de los canales 0 y 97 diezmada a 39.06Hz 76
4.11 Gráfica de las señales de LFP a la frecuencia de muestreo de los ángulos (39.06Hz) durante los primeros 10 segundos del canal 70. . 77
4.12 Gráfica de las señales de LFP del canal 70 y los ángulos a la frecuencia de muestreo de 39.06Hz en una ventana de $\sim 5min$. 78
4.13 Ángulos de la sesión 765 limpios a la frecuencia de muestreo de las señales de LFP (1250Hz) durante $\sim 3min$. Utilizando las interpolaciones: lineal (arriba izquierda), cuadrática (arriba derecha), valor más cercano (abajo izquierda) y ángulo más cercano (abajo derecha). 79

4.14 Ángulos de la sesión 771 limpios a la frecuencia de muestreo de las señales de LFP (1250Hz) durante $\sim 3min$ Utilizando las interpolaciones: lineal (arriba izquierda), cuadrática (arriba derecha), valor más cercano (abajo izquierda) y ángulo más cercano (abajo derecha). 80
4.15 Gráfica de las señales de LFP del canal 70 y los ángulos a la frecuencia de muestreo de 1250Hz en una ventana de $\sim 5min$. 82
4.16 Gráfica de amplitud de los LFPs contra el ángulo de dirección de la cabeza en la sesión 765 a 1250Hz. Ángulo de preferencia: 182° . 85
4.17 Gráfica de amplitud de los LFPs contra el ángulo de dirección de la cabeza en la sesión 771 a 1250Hz. Ángulo de preferencia: 266° . 86
4.18 Gráfica de amplitud de los LFPs contra el ángulo de dirección de la cabeza en la sesión 765 a 39,06Hz. Ángulo de preferencia: 154° . 87
4.19 Gráfica de amplitud de los LFPs contra el ángulo de dirección de la cabeza en la sesión 771 a 39,06Hz. Ángulo de preferencia: 266° . 87
4.20 Gráfica de 5 lotes continuos del conjunto de prueba de la sesión 771 con la predicción de ángulos (naranja) contra ángulos esperados (verde) y LFPs (azul). La duración es de 5 lotes de datos de 32 ventanas de 100ms: 16 segundos. Modelo: LSTM 90
4.21 Gráfica de 10 lotes continuos del conjunto de prueba de la sesión 771 con la predicción de ángulos (naranja) contra ángulos esperados (verde) y LFPs (azul). Duración de 32 segundos. Modelo: LSTM . 91
4.22 Gráfica de 20 lotes continuos del conjunto de prueba de la sesión 771 con la predicción de ángulos (naranja) contra ángulos esperados (verde) y LFPs (azul). Duración de 64 segundos. Modelo: LSTM . 92
4.23 Gráfica de 50 lotes continuos del conjunto de prueba de la sesión 771 con la predicción de ángulos (naranja) contra ángulos esperados (verde) y LFPs (azul). Duración de 160 segundos. Modelo: LSTM 92
4.24 Gráficas de la sesión 765, canal 40 con la predicción de ángulos (naranja) contra ángulos esperados (verde) y LFPs (azul). Modelo: CNN 93
4.25 Gráficas de la sesión 771 con la predicción de ángulos (naranja) contra ángulos esperados (verde) y LFPs (azul). Modelo: MLP . . 94
4.26 Gráficas de la sesión 771, canal 70 con la predicción de ángulos (naranja) contra ángulos esperados (verde) y LFPs (azul). Modelo: LSTM 97
4.27 Gráficas de la sesión 765, canal 40 con la predicción de ángulos (naranja) contra ángulos esperados (verde) y LFPs (azul). Modelo: LSTM 98
4.28 Gráfica de todo el conjunto de prueba de la sesión 765 con la predicción de ángulos (naranja) contra ángulos esperados (verde) y LFPs (azul). Modelo: LSTM 99

4.29 Gráficas de la sesión 771, canal 40 con la predicción de ángulos (naranja) contra ángulos esperados (verde) y LFPs (azul). Modelo: LSTM. Interpolación Lineal. 100
4.30 Gráficas de la sesión 771, canal 40 con la predicción de ángulos (naranja) contra ángulos esperados (verde) y LFPs (azul). Modelo: MLP. Diezmando las señales LFP. 101
4.31 Gráficas de la sesión 771, canal 40 con la predicción de ángulos (naranja) contra ángulos esperados (verde) y LFPs (azul). Modelo: CNN. Diezmando las señales LFP. 102
4.32 Gráficas de la sesión 771, canal 40 con la predicción de ángulos (naranja) contra ángulos esperados (verde) y LFPs (azul). Modelo: LSTM. Diezmando las señales LFP. 102
4.33 Gráficas Error Absoluto Medio contra Épocas. Utilizando datos diezmados a 39.06Hz del canal 40 de la sesión 771 con ventanas de 179ms durante 50 épocas. Datos de entrenamiento en rojo y validación en azul. 103
4.34 Gráficas de todo el conjunto de prueba con la predicción de ángulos (naranja) contra ángulos esperados (verde) y LFPs (azul). Modelo: LSTM 105
4.35 Gráficas Error Medio Absoluto contra Épocas al incrementar la ventana a 1s. Modelo: LSTM 106

Índice de cuadros

3.1 Información General del banco de datos 41
3.2 Información de los archivos de las sesiones. 42
3.3 Información de las células grabadas 43
3.4 Ubicación de las sondas en el cerebro. 44
3.5 Resumen de cantidad de *spikes* por tipo de célula en los bancos de datos. 45

4.1 Cantidad de muestras por cada subconjunto generado después de la limpieza de ángulos inválidos de la sesión 765. 83
4.2 Cantidad de muestras por cada subconjunto generado después de la limpieza de ángulos inválidos de la sesión 771. 84
4.3 Hiperparámetros para la generación del conjunto de datos de entrada para las redes neuronales. 88
4.4 Hiperparámetros para el entrenamiento de las redes neuronales. . . 88
4.5 Resumen de resultados de rendimiento de las 3 redes neuronales durante 10 épocas utilizando interpolación por el ángulo más corto de los ángulos (frecuencia de sincronización: 1250Hz). El tamaño de la ventana está dado en milisegundos [ms] y el tiempo total de la ejecución en [h:min]. 95

Nomenclatura

AD Núcleo Anterior-dorsal del tálamo del cerebro

AI Inteligencia Artificial (del inglés *Artificial Intelligence*)

AUC Área bajo la curva (del inglés *Area Under the Curve*)

BMI Interfaz Cerebro-Máquina (del inglés *Brain-Machine Interface*)

DBS Estimulación de Cerebro profundo (del inglés *Deep Brain Stimulation*)

EC Corteza Entorrinal *Entorhinal Cortex*)

$ECAP$ Conjunto de Potenciales de acción provocados (del inglés *Evoke Compound Action Potentials*)

EEG Electroencefalografía

EIE Escuela de Ingeniería Eléctrica de la Universidad de Costa Rica

EMI Interferencia Electromagnética (del inglés *Electromagnetic Interference*)

HD Células de dirección de la cabeza (del inglés *Head-Direction cells*)

$IEEE$ Instituto de Ingenieros Eléctricos y Electrónicos (del inglés *Institute of Electrical and Electronics Engineers*)

LFP Potenciales de campo locales (del inglés *Local Field Potential*)

$LSTM$ Redes de Memoria Larga de Corto Plazo (del inglés *Long Short-Term Memoryl*)

MAE Error Absoluto Medio (del inglés *Mean Absolute Error*)

$MAPE$ Error Porcentual Absoluto Medio (del inglés *Mean Absolute Percentage Error*)

ML Aprendizaje Automático (del inglés *Machine Learning*)

MSE Error Cuadrático Medio (del inglés *Mean Squared Error*)

NaN Valor el cual no es un número (del inglés *Not a Number*)

NN Red Neuronal (del inglés *Neural Network*)

$RMSE$ Raíz del Error Cuadrático Medio (del inglés *Root Mean Squared Error*)

ROC Característica Operativa del Receptor (del inglés *Receiver Operating Characteristic*)

RP Reconocimiento de Patrones

SCS Estimulación de médula espinal (del inglés *Spinal Cord Stimulation*)

UCR Universidad de Costa Rica

Resumen

Palabras Clave: Inteligencia Artificial, extracción de características, aprendizaje automático, LFP, cerebro, red neuronal, serie de tiempo, predicción, neuronas.

La extracción de información a partir de la actividad neuronal, es un tema que está siendo ampliamente estudiado con diversos propósitos cómo la creación de nuevos dispositivos y expandir el conocimiento sobre el funcionamiento del sistema nervioso y el cerebro en particular.
La presente investigación plantea el análisis de la actividad cerebral de ratas, medida utilizando la técnica de Potenciales de Campo Local con el fin de extraerle características que funcionen como entradas a algoritmos de aprendizaje automático para predecir el ángulo al cual se dirige la cabeza del sujeto a partir de dichas señales neuronales.
Los datos fueron tomados de las sesiones de grabación 765 y 771 realizadas en el laboratorio Buzaki Lab. En ellas se colocaron 99 electrodos en el hipocampo y tálamo de una rata que se ubicaba e incentivaba a desplazarse en tal espacio. Dichas regiones cerebrales contienen células de dirección de la cabeza que son relacionadas con el sentido de orientación.
El proyecto se divide en la exploración, limpieza y preparación del conjunto de datos a utilizar; la implementación de los algoritmos de predicción y finalmente se discuten los resultados.
Se escogen tres tipos de redes neuronales para realizar la predicción: clásica, convolucional y una *Long Short-Term Memory*. Todas las redes propuestas logran predecir ángulos entre 0 y 360 grados, con pocas excepciones. La LSTM fue la que produjo el menor error con respecto a los valores reales. Ninguna logró obtener predicciones confiables, ya que el Error Absoluto Medio más bajo en datos de prueba fue de 78°, obtenido con la LSTM.
Se logra demostrar que los datos obtenidos contienen información de células de dirección de la cabeza al observar un pico de actividad claro en sus respectivos ángulos de preferencia, lo cual indica que existe relación directa entre los LFPs y el ángulo al cuál se dirige el sujeto y es viable predecir los ángulos con cierta exactitud.
Finalmente se concluye que la mejor opción entre las redes neuronales para continuar la investigación y mejorar la predicción de los ángulos a partir de señales de LFPs es la LSTM utilizando datos diezmados a 39.06Hz.

1 Introducción

1.1 Planteamiento del Problema

El cerebro ha sido estudiado de manera funcional a través de los años utilizando técnicas como encefalogramas (EEG), que permiten observar las regiones del cerebro con más actividad eléctrica mientras el sujeto realiza una determinada acción. Al ser un método poco invasivo, es uno de los más utilizados para el estudio de este órgano del sistema nervioso. Sin embargo, presenta el inconveniente de que sus datos adquiridos, contienen actividad de distintas regiones del cerebro, así como otras señales no deseadas de órganos como el corazón, movimiento ocular, músculos, así como interferencia captada por los electrodos y el sistema de adquisición. Dichas señales no deseadas son conocidos como artefactos, y se han desarrollado técnicas para reconocer y filtrarlos para obtener los datos de la región de interés. (Sornmo y Laguna, 2005)

Por otro lado, si el objetivo es sensar y manipular señales de actividad neuronal en un área particular del sistema nervioso, existen otras técnicas de adquisición como *Local Field Potentials* (LFP), las cuales contienen muchos menos artefactos, con la desventaja de que la colocación de los electrodos es un procedimiento más invasivo para el sujeto.

Comprender con más detalle la información contenida en adquisiciones de LFP, permitiría proporcionar terapias de *Deep Brain Stimulation* (DBS) mejor localizadas, y que modifiquen el comportamiento de las neuronas estimuladas de manera óptima. Además permitiría simplificar y automatizar parte del proceso de configuración de los programas de estimulación para pacientes con enfermedades como Parkinson, distonía, dolor crónico, entre otros.

Utilizando señales *Local Field Potentials* (LFP) se propone identificar la relación entre la actividad eléctrica de un conjunto de neuronas dedicadas al sentido de orientación, y la dirección a la cual se dirige la cabeza del sujeto. Las señales a utilizar son tomados de un banco de datos en la cual utilizan ratas para el experimento.

Se extraerá un conjunto de características de los datos los cuales se alimentarán como entrada para el entrenamiento de algoritmos de aprendizaje automático supervisado. Específicamente, se propondrán dos topologías distintas de redes neuronales (NN) con el fin de determinar si es posible obtener la información angular de la cabeza de la rata a partir de señales eléctricas provenientes de una sección específica del cerebro.

1.2 Antecedentes

La investigación sobre la actividad cerebral ha sido desarrollada utilizando principalmente señales de electroencefalogramas (EEG) ya que se considera poco invasiva. Lo anterior dado que los electrodos de adquisición se localizan sobre el cuero cabelludo. La colocación de los electrodos se hace de acuerdo con el Sistema Internacional 10/20 de adquisición de datos EEG, basada en los puntos craneométricos (Cavelli, 2017).

Gracias a este mecanismo de adquisición, se ha logrado recolectar gran cantidad de datos de actividad eléctrica del cerebro humano, que ha permitido la investigación de reconocimiento de patrones en estas señales, en áreas de robótica, prótesis, y demás. Por ejemplo, la detección de convulsiones epilépticas en señales EEG utilizando Redes Neuronales Recurrentes (RNN) por (Naderi y Mahdavi-Nasab, 2010). Por otro lado, se han realizado investigaciones sobre Interfaces cerebro-máquina (BMI) para desarrollar aplicaciones móviles de mensajería basados en deletreadores a partir de señales EEG (Jayabhavani et al., 2013).

La investigación de (He et al., 2016) propone un BMI basado en comandos de señales EEG para controlar distintos actuadores de una prótesis robótica de un brazo. Este experimento graba, filtra, extrae las características esenciales y clasifica las señales EEG, para posteriormente enviarlas a través de Bluetooth los comandos a *encoders* y giroscopios en el brazo robótico. También, fueron capaces de adquirir información de posición, velocidad y aceleración en tiempo real del movimiento de las articulaciones para hacer ajustes de las señales de control.

Sin embargo, técnicas de adquisición como EEG o NIRS, presentan el inconveniente de que al utilizar electrodos de mucha área, que además se colocan en la superficie de la cabeza, se obtienen señales con muchos artefactos provenientes de otros órganos como el latido del corazón, la respiración, el movimiento de los ojos. Artefactos que son relacionados al sistema de adquisición como interferencia electromagnética (EMI) captados por los electrodos. Pero principalmente, aún aislando y filtrando los artefactos, la adquisición de las señales cerebrales contendrán información de distintas partes del cerebro. De forma que permiten identificar cuál parte del cerebro presenta más actividad ante un determinado estímulo, pero no es posible obtener con detalle el comportamiento de esta sección particular del cerebro. (Sornmo y Laguna, 2005)

Es para resolver este problema que se comienza a investigar sobre adquisición a una menor escala, que permita una adquisición más detallada y de menor potencia como lo son las señales LFP e ECAP. Sin embargo, ha sido utilizada en menor medida ya que el procedimiento para la colocación de los electrodos es más invasiva que EEG. Y además, la tecnología de adquisición

fue concebida muy recientemente a través de estimuladores neuronales para DBS que no solo tienen la capacidad de entrega de terapia, sino también de sensado.

Entre las investigaciones más importantes sobre reconocimiento de patrones en señales de LFP se encuentra la de (Islam et al., 2017), quienes obtuvieron datos de personas con Distonía o Parkinson presionando voluntariamente un botón, con ambos dedos índices (izquierdo y derecho).

Ellos utilizaron tres topologías de redes neuronales distintas para reconocer a cuál dedo correspondía cada señal adquirida: *Feedforward Backpropagation Artificial Neural Network* (FBANN), *Radial Basis Function Neural Network* (RBFNN) y *Probabilistic Neural Network* (PNN), cuyos resultados se someten a una votación por mayoría y así se determina el resultado final de la clasificación. Finalmente obtuvieron un rendimiento de $0{,}671 \pm 0{,}14$ al clasificar los datos entre movimientos con el dedo izquierdo o derecho.

Una aproximación a interpretar datos LFP utilizando algoritmos de AI no supervisado es la de (Jackson y Hall, 2017) quienes escogieron los métodos de *biomimetic decoding* y *biofeedback learning* para obtener la información cinemática de la corteza motora en monos.

Por otro lado, (Mizuseki et al., 2014) recolectaron datos de LFP de distintas regiones del Hipocampo y la Corteza Entorrinal de 11 ratas mientras realizaban 1 de 8 comportamientos. Dicha investigación buscaba recolectar datos para investigaciones posteriores como vincular la actividad eléctrica del conjunto de neuronas de interés, con la dirección del movimiento de la cabeza del animal. Además de intentar probar la hipótesis de que la información de las células de dirección de la cabeza debía persistir a través de las distintas etapas del sueño.(Peyrache et al., 2015)

El banco de datos recolectado del artículo mencionado anteriormente es el que se escoge para llevar a cabo la presente investigación, ya que tiene varias de las características deseadas como la disponibilidad de datos etiquetados, la cual debía de tener la suficiente cantidad de ejemplos para entrenar una red neuronal. Además, tiene datos de LFP relacionados a movimiento y a su vez los datos del ángulo de dirección de la cabeza que se usarán como las etiquetas para el entrenamiento. Finalmente, dichos datos provienen del laboratorio *Buzáki Labs* que tiene vasta experiencia en temas de recolección de datos cerebrales, lo cual aumenta la confiabilidad del origen de los datos a utilizar.

Entre las investigaciones realizadas en la Escuela de Ingeniería Eléctrica de la UCR, se han tratado temas de imaginería motora y sistemas EEG. El profesor Dr. rer. nat. Francisco Siles Canales fundó el Laboratorio de Reconocimiento de Patrones y Sistemas Inteligentes (PRIS-Lab) quienes han realizado investigaciones dirigidas en interfaces cerebro-computadoras (BCI), utilización de algoritmos de aprendizaje automático en aplicaciones como clasificación au-

tomática de células cancerígenas, la mejora de señales degradadas con ruido en aplicaciones de habla y clasificación automática de jugadores de fútbol a partir de video digital (PRIS-Lab, 2019).

Algunos de los proyectos realizados por este laboratorio relacionados a señales cerebrales son:
La investigación de Eliécer Agüero (Aguero, 2013), quien realizó su tesis sobre un deletreador utilizando señales de EEG con electrodos activos en el cual determinó que el uso de electrodos pasivos da mejores resultados. Los estudiantes Juan José Delgado (Delgado, 2018) y Andrea Salazar (Salazar, 2018) realizaron sus trabajos de graduación de bachillerato sobre la imaginería motora. Trabajaron en la clasificación de los datos del primer y segundo experimento de (Shin et al., 2017) respectivamente, y utilizaron la densidad espectral de potencia como método de clasificación. J. Shin generó un conjunto de datos híbrida para *Brain-Computer Interface* (BCI), utilizando NIRS y EEG. Realizaron dos experimentos, el primero consistía en imaginería motora del brazo izquierdo contra el derecho. El segundo comparaba la actividad cerebral de un individuo realizando cálculos aritméticos contra cuando se está en descanso (Shin et al., 2017). Para sus experimentos, colaboraron 30 sujetos.

1.3 Justificación

Actualmente la terapia basada en impulsos eléctricos para contrarrestar señales indeseadas en el sistema nervioso como lo son el dolor crónico, distonía o la enfermedad de Parkinson, conocidas como SCS y DBS respectivamente, son programadas manualmente por el médico en varias sesiones con el paciente (Arle y Shils, 2011). El proceso de personalización de la terapia puede tardar varios meses después de implantado el neuromodulador, lo cual representa también un esfuerzo monetario importante por parte del usuario.

Uno de los factores que afecta la velocidad del proceso de programación adecuada de la terapia es la dificultad de encontrar la combinación correcta de amplitud, ancho de pulso y frecuencia que atenuará de manera óptima la señal indeseada. Es por esta razón que en las últimas décadas se ha comenzado a investigar sobre formas de medir el comportamiento de las neuronas que se desean estimular, primeramente sin terapia para comprender su comportamiento natural y conocer más a fondo la señales que provocan estas enfermedades. Y segundo, observar la respuesta de las células ante los impulsos dosificados (señales ECAP).

Con la intención de avanzar en el diseño de sistemas de neuromodulación de lazo cerrado, distintas empresas de dispositivos médicos están investigando y desarrollando la tecnología para sensar los estímulos naturales del siste-

ma nervioso (LFP), así como su respuesta ante impulsos eléctricos conocidos (ECAP). También se estudian algoritmos para clasificar, predecir y reconocer patrones en las señales anteriormente mencionadas para posteriormente ajustar los parámetros de la estimulación hasta alcanzar la terapia más adecuada para cada paciente.

Como fue mencionado anteriormente, ya se han realizado investigaciones importantes en el área de clasificación de señales LFP, por lo que el presente trabajo busca diferenciarse de los antecedentes del proyecto en los siguientes aspectos:

- Se definirá una técnica de extracción de características adecuada para potenciar la información contenida en señales de Potenciales de Campo Local (LFP).
- Se propondrá el uso de dos topologías de redes neuronales para la clasificación de señales de LFP.

1.4 Hipótesis

Se plantea que es posible predecir con un alto porcentaje de exactitud la dirección (ángulo) hacia la cual se dirige la cabeza de ratas, a partir de la actividad eléctrica de un conjunto de neuronas ubicado específicamente en el tálamo cerebral utilizando señales de *Local Field Potentials*. Dicha región del cerebro se encarga de la ubicación y orientación espacial del sujeto. El experimento se plantea basándose en el uso de técnicas de extracción de características y redes neuronales.

Lo anterior con el objetivo de determinar si es posible desarrollar mejoras en sistemas para tratamiento de enfermedades como Síndrome de Parkinson, Distonía, dolor crónico, entre otros; capaces de reconocer patrones de la enfermedad y proporcionar alternativas de configuración de la terapia para reducir el aporte del padecimiento al sistema nervioso.

1.5 Objetivos

1.5.1 Objetivo General

Implementar un clasificador automático de señales de Potencial de Campo Local (LFP) utilizando redes neuronales.

1.5.2 Objetivos Específicos

1. Definir una técnica de extracción de características adecuada para obtener el conjunto de datos de entrada para el algoritmo de clasificación automática.

2. Implementar dos topologías de Redes Neurales para la asociación automática de señales de LFP y dirección de la cabeza del sujeto.

3. Comparar los resultados obtenidos de ambas redes neuronales, basado en métricas de exactitud en la clasificación.

1.6 Alcances

El presente proyecto plantea evaluar la hipótesis de que, a partir de señales de actividad neuronal adquiridas de un conjunto específico de neuronas, es posible extraer información sensorial y/o racional del sujeto.

Debido a la dificultad de encontrar una base de datos de señales de LFP de sujetos humanos, ya que el procedimiento de adquisición es muy invasivo, se utilizará el banco de datos de (Mizuseki et al., 2013) que se compone de mediciones de *Local Field Potentials* en ratas.

De tal forma que, la presente investigación se limita a determinar si es posible obtener la dirección (en ángulos) a la cual se dirige la cabeza del animal, a partir de señales neuronales de LFP provenientes específicamente de un conjunto de neuronas en el tálamo del cerebro del sujeto.

Para esto, la investigación se desarrollará en dos etapas: Primeramente se preparará el banco de datos, para posteriormente definir una técnica que permita seleccionar el conjunto de características apropiado para el entrenamiento de las redes neuronales.

Seguidamente, se delimitará la solución del problema de clasificación a la implementación de algoritmos de inteligencia artificial de aprendizaje automático supervisado, específicamente, dos topologías de redes neuronales. Lo anterior, con el fin de comparar los resultados de ambas en términos de precisión.

1.7 Metodología

La metodología propuesta para llevar a cabo el presente proyecto se divide en cuatro etapas principales, las cuales se detallan a continuación:

- **Primera Etapa:** *Investigación de métodos, técnicas y algoritmos.*

  El proyecto a desarrollar consiste en la predicción de la dirección a la cual se dirigen los sujetos, a partir de clasificación de señales de LFP. Lo cuál requiere la selección de técnicas de limpieza y extrapolación de datos, extracción de *features*, y algoritmos de aprendizaje automático. Por lo que debe realizarse una investigación detallada y amplia del estado de la cuestión en estas áreas del conocimiento, así como una cautelosa deliberación para escoger las técnicas apropiadas en cada etapa.

  Se propone estudiar varias opciones de técnicas para cada etapa, para posteriormente, mediante pruebas utilizando los recursos disponibles, y tomando siempre en cuenta el tiempo y alcances fijados en el proyecto, seleccionar los algoritmos que mejor se ajusten a la necesidades e intereses del proyecto.

  Se utilizarán revistas, libros, artículos de investigación, paginas web y demás recursos bibliográficos para la recolección de información acerca del análisis de señales neuronales, especialmente *Local Field Potentials*, prepración y limpieza de datos, extracción de características, redes neuronales, aprendizaje automático supervisado y métricas de evaluación de los resultados que se obtengan, entre otros.

  Se revisará el estado de la cuestión sobre las topologías de redes neuronales más utilizadas para la clasificación de señales dependientes del tiempo, señales provenientes del cerebro como LFP, análisis en frecuencia y demás, con el fin de escoger al menos dos algoritmos para implementar.

- **Segunda Etapa:** *Preparación de datos y extracción de características*

  Previo a la clasificación de señales, es necesario traducir los datos a un formato que facilite su importación, estandarizarlos (de ser requerido) y limpiarlos de aquellos que estén considerablemente fuera de la media (*outliers*), ruido y artefactos. Posteriormente, se escogerá una técnica de extracción de características esenciales de la señal para obtener un subconjunto balanceado de datos, que contengan información relevante y pueda utilizarse para predecir la etiqueta correctamente.

  La técnica de extracción de característica se evaluará al analizar el subconjunto de datos resultantes, basándose en métricas cuantitativas como correlación y P Valores, entre otros.

- **Tercera Etapa:** *Implementación de las redes neuronales*

  Una vez seleccionado el algoritmo de aprendizaje automático, se realizarán pruebas mediante variaciones en los parámetros de topología de las redes neuronales tales como: cantidad de nodos por capa, número de capas, conexión entre nodos y capas. Además de variaciones en los parámetros de entrenamiento de dicha red: iteraciones *epochs*, velocidad de aprendizaje *learning rate*, tamaño del lote *batch* de datos, número de *batches* y demás. Se utilizarán métricas que reflejen, de forma apropiada, la cantidad de aciertos o porcentaje de error de la predicción, para seleccionar la topología y configuración final de la red neuronal.

- **Cuarta Etapa:** *Análisis de resultados*

  Se compararán cuantitativamente los resultados de ambas topologías de redes neuronales ante variaciones en el conjunto de datos seleccionado para su entrenamiento, así como variaciones en la composición y proceso de entrenamiento de las redes, como se mencionó anteriormente.

1.8 Procedimiento de evaluación

Para evaluar este proyecto se propone el siguiente procedimiento:

- Los resultados correspondientes al primer objetivo se evaluarán con base en métricas o referencias bibliográficas que permitan determinar si la selección e implementación de la técnica de extracción de características utilizada proporciona finalmente valores adecuados para el entrenamiento de las redes neuronales.

- La evaluación del segundo objetivo se dará mediante la presentación del código de programación que implementa ambas topologías de redes neuronales, la cuales, deberán ser capaces de proporcionar una predicción del ángulo de dirección de la cabeza basándose en las características del banco de datos de LFP seleccionadas.

- El análisis y la comparación de los resultados obtenidos de la implementación de ambas topologías de NN será el entregable del tercer objetivo. Para esto, se establecerán métricas que reflejen apropiadamente la cantidad de aciertos o porcentaje de error en la predicción final del ángulo por la red neuronal, y así, seleccionar la red que presente mejores resultados en términos de rendimiento, exactitud o precisión.

2 Marco Teórico

2.1 Actividad Cerebral

2.1.1 Potencial de Acción

En el cerebro existe un equilibrio eléctrico entre el interior y exterior de las neuronas que es de aproximadamente -70 mV (Barnett et al., 2007). Si se produce un flujo de iones de sodio (Na^+) hacia el interior de la célula lo suficientemente grande se produce la **despolarización** de la neurona. El interior de esta termina más positivo en comparación con su exterior, obligándola a cerrar la compuerta de sodio de la membrana. Ya que se creó una diferencia de potencial entre el exterior e interior de la célula con la despolarización, átomos de Potasio (K^+) son liberados hacia el exterior de la célula por su respectiva compuerta. Sin embargo, este flujo de iones de potasio ocurre en forma de frenesí conocido como **hiperpolarización**, dejando salir más iones de lo que se necesitaban para anular la diferencia de potencial. Esto crea una diferencia de potencial de signo opuesto que cierra la compuerta de potasio, llamando este proceso **repolarización**. Para que la neurona sea capaz de recibir otra señal, es necesario dejar que los iones vuelvan a sus posiciones de reposo, lo cual es posible gracias a las **bombas de potasio y sodio**, quienes necesitan de energía para llevar acabo el proceso de sacar 3 iones de sodio y meter 2 de potasio a la célula. Razón por la cuál el cerebro es un órgano que consume mucha energía considerando la cantidad de interacciones neuronales que ocurren en él. Estas 3 etapas juntas se conocen como un **potencial de acción**. (Barnett et al., 2007)

Para que un potencial de acción se dé, es necesario que se logre disminuir la diferencia de potencial de -70mV a cerca de -55mV, entonces ocurrirá la despolarización dejando una diferencia de $\sim 40mV$ lo que conlleva a la repolarización y a la etapa de reposo para volver al equilibrio de -70mV, como se muestra en la figura 2.1. Las acciones de potencial se mueven en una sola dirección del axón creando una **corriente local** de iones (Barnett et al., 2007).

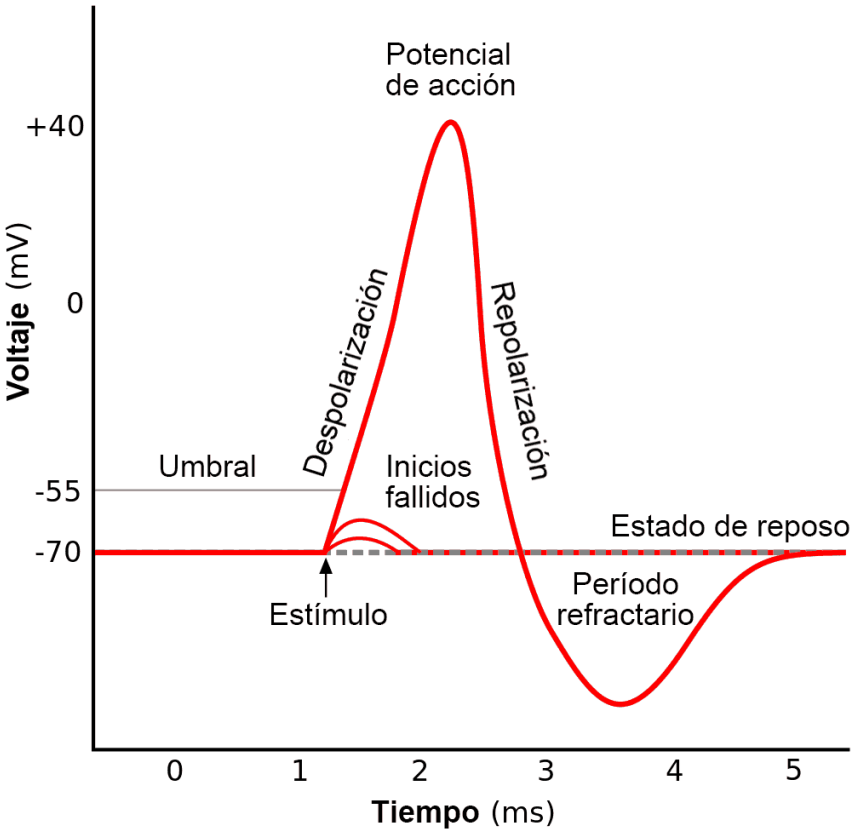


Figura 2.1: Potencial de Acción. (Ramos, 2019)

2.1.2 *Deep Brain Stimulation*

La estimulación de cerebro profundo (DBS) es una técnica que descarga impulsos eléctricos configurados con una amplitud, una frecuencia y un ancho de pulso determinados en una región específica del cerebro para tratar padecimientos como el Mal de Parkinson, Tremor Esencial (Perlmutter et al., 2006).

Esta técnica saca provecho del fenómeno de potencial de acción al inyectar estos impulsos eléctricos para estimular la despolarización de las neuronas circundantes al electrodo y desencadenen potenciales de acción.

Los electrodos del Generador de Impulsos Implantable (IPG) deben ser colocados en una región específica del cerebro dependiendo de la condición que se busca tratar (Perlmutter et al., 2006):

- Para Tremor Esencial (ET) se deben implantar los electrodos del dispo-

sitivo en el núcleo intermedio ventral del tálamo (VIM).

- Parkinson: Se pueden estimular distintas áreas de acuerdo al efecto que se desee producir. Si se estimula en el VIM se podría reducir el tremor (temblor) en las extremidades, pero podría provocar efectos secundarios en otras características de la enfermedad. Si se estimula el Globo Pálido interno (GPi) se logra reducir casi todas las manifestaciones motoras principales del PD (*Parkinson Disease*), incluyendo movimientos involuntarios provocados por medicamentos. Estimular el GPi también ayuda a reducir las manifestaciones de dolor y sensoriales del padecimiento, sin embargo no permite eliminar el uso de otros medicamentos para tratar la condición.

- Núcleo subtalámico (STN): El DBS aplicado en esta región del cerebro tiene reducción de síntomas motores similares al del GPi.

También se ha investigado sobre los efectos de DBS para tratar Distonía y el Trastorno Obsesivo-Compulsivo.

2.1.3 Información de Orientación en el Cerebro

El hipocampo es una parte del cerebro frontal localizada en el lóbulo temporal que está directamente relacionado con la memoria y que hace uso de la mayoría de sus regiones, especialmente la Corteza Entorrinal (EC) (Buzsaki, 2011). Esta región interfaza el hipocampo con la neocorteza y es la principal entrada al circuito de información de la figura 2.2, mientras que la principal salida es la sub-región CA1 (iniciales de *Cornu Ammonis*). La información en este circuito fluye principalmente de forma unidireccional. Se debe destacar que el sistema recurrente más grande del cerebro está formado por las neuronas de la región CA3, así como también es de tamaño significativo la red entre las capas entorrinales 2 y 3.(Buzsaki, 2011)

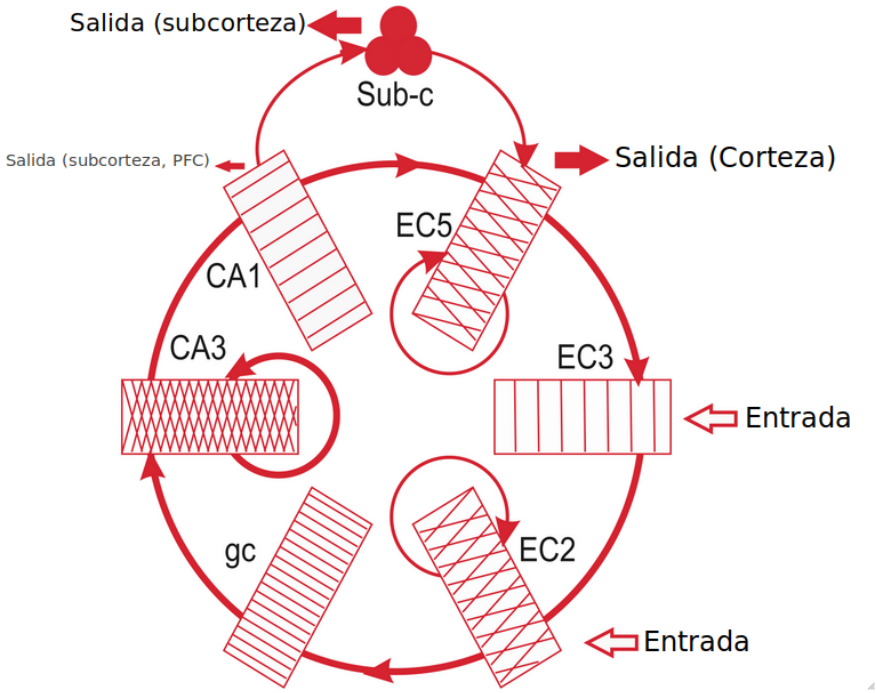


Figura 2.2: Circuito principal del hipocampo. (Buzsaki, 2011)

En el hipocampo se encuentran un tipo de neuronas piramidales conocidas como *células de lugar* debido a que sus patrones de disparos son relacionados con la ubicación de los sujetos en un ambiente dado. La actividad está dada por el reconocimiento del ambiente por parte del sujeto y no está relacionada con la ubicación geográfica (Buzsaki, 2011). Por otro lado, en la corteza entorrinal se encuentran otro tipo de neuronas llamadas *células de dirección de cabeza* o *HD Cells* (del inglés "Head Direction Cells").

Las células de dirección de cabeza son neuronas que se activan en relación con la dirección a la que se dirige la cabeza del animal con respecto al ambiente en el plano horizontal (Taube, 2009). Estas neuronas tienen una dirección de preferencia como se muestra en la figura 2.3, similares a una brújula en el aspecto de que su descarga está siempre afinada a una dirección particular y pueden ser disparadas en cualquier lugar del ambiente siempre y cuando la cabeza del animal esté apuntando a dicha dirección. En contraste con las brújulas, estas células no reaccionan ante el campo magnético de la Tierra, sino a puntos de referencia y señales de auto-movimiento. La actividad de las *HD cells*, es indiferente a la posición y orientación del cuerpo y a si el sujeto está en movimiento o no. Cada neurona tiene una dirección a la que se dispara al máximo (Taube, 2009).

Figura 2.3: Actividad neuronal contra ángulo de dirección a la que se dirige la cabeza. (Taube, 2009)

Las células de dirección de la cabeza se pueden encontrar en distintas partes del cerebro tales como el cuerpo estriado, núcleos mamilares laterales, corteza retrosplenial, la corteza entorrinal y son particularmente abundantes en el núcleo anterodorsal del tálamo. Postulaciones recientes indican que, de los núcleos mamilares laterales que contiene tanto células sensibles a la velocidad angular de la cabeza como células de dirección de la cabeza, se proyecta información al tálamo anterodorsal, luego al *postsubiculum*, seguidamente a la corteza entorrinal, para finalizar en el hipocampo. De esta forma, la información de la dirección de la cabeza se complementa e integra con la información sobre la ubicación del animal en el hipocampo y en la corteza entorrinal. Juntas, la información de lugar y dirección de la cabeza proveen el sentido de orientación espacial en el ambiente (Taube, 2009).

2.1.4 Técnicas de Adquisición de señales Cerebrales

Existen distintas técnicas para adquirir datos provenientes del cerebro para determinar regiones de mayor actividad ante ciertos estímulos.

Una de las técnicas más utilizadas es el **Electroencefalograma** (EEG) el cual refleja la actividad cerebral a través de varios electrodos posicionados en el cuero cabelludo. El EEG es muy utilizado para diagnosticar enfermedades en el cerebro como convulsiones epilépticas o evaluar desordenes del sueño. Existe

otro método para obtener datos de actividad cerebral situando los electrodos directamente en la superficie del cerebro de forma epidural, conocido como **Electrocorticografía** (ECoG). La razón por la que estos métodos son tan utilizados es que son poco invasivos, en comparación con métodos como el de Potenciales de Campo Local (LFP) los cuales se describirán seguidamente. (Gibson et al., 2012)

2.1.5 Ritmos cerebrales

En general, las formas de onda de EEG van en frecuencias desde 0,5Hz y 30-40Hz, mientras que presentan unos cuantos micro Volts hasta los 100μV. Las formas de onda cerebrales adquiridas utilizando EEG usualmente se clasifican en 5 bandas distintas y varían de acuerdo a la edad y estado mental del sujeto (Sornmo y Laguna, 2005).

- **Delta <4 Hz:** Típicamente se presenta en sueño profundo, no es normal encontrarlo en personas despiertas dada su gran amplitud, sin embargo de ser así, suele ser indicativo de enfermedades mentales o daños cerebrales.
- **Theta 4-7 Hz:** Ocurre cuando se está soñoliento o en algunas etapas del sueño.
- **Alpha 9-13Hz:** Es la más común en sujetos despiertos y relajados con los ojos cerrados, se pierde al abrir los ojos. La amplitud de señales Alpha es la más grande en regiones occipitales del cerebro.
- **Beta 14-30 Hz:** Es observado mayoritariamente en las regiones frontales y centrales de la cabeza. Es un ritmo rápido y de baja amplitud y se presenta en algunas etapas del sueño.
- **Gamma >30 Hz:** Están relacionadas a una alta actividad en la corteza. Conectando un electrodo en el área motora-sensorial, la señales Gamma se pueden encontrar durante el movimiento de dedos.

El uso de dichas división en bandas de frecuencia se extiende a otras técnicas de adquisición como los Potenciales de Campo Local (LFP).

2.1.6 *Local Field Potentials* (LFP)

Los Potenciales de Campo Local provienen de varias fuentes, pero principalmente de actividad sináptica. Esto debido a que las membranas de las células cerebrales están hechas de lípidos y tienen un efecto eléctrico capacitivo que modela un filtro pasa bajos. De forma que, los componentes de alta frecuencia

de las señales neuronales se ven altamente atenuadas en el medio extra-celular. De igual manera, las señales lentas logran viajar más lejos que las de alta frecuencia. Como resultado, la medición en un punto del cerebro ubicado en el medio extracelular contiene la sumatoria de la actividad de las sinapsis de gran cantidad de neuronas. Por lo tanto, se podría decir que las señales LFP son el resultado de 'acciones cooperativas' de neuronas (Gibson et al., 2012).

Los "*spikes*" en contraste, provienen de la actividad de una sola neurona cerca de la punta del electrodo. Por la amplia esfera de adquisición, los LFPs se ven menos afectados por la cicatrización del tejido al insertar los electrodos (Maling y McIntyre, 2016).

Adicionalmente, el análisis de *spikes* no es sostenible en el tiempo debido a precisamente la cicatrización e incompatibilidad de los materiales de las sondas y electrodos con el tejido del cerebro. Otra ventaja que presentan los LFP en comparación con los picos provocados por la actividad de una neurona es que se pueden muestrear a frecuencias más bajas y por lo tanto la adquisición consume menos potencia. El disparo de una neurona es muy rápido, aproximadamente menor a 1ms, que implica una frecuencia de muestreo de 10KHz para tener una señal con suficiente muestras por periodo (Jackson y Hall, 2017). En términos de aplicaciones médicas, particularmente de dispositivos implantables, dicha frecuencia de muestro así como la potencia requerida presentan una limitante.
Las frecuencias de análisis de los Potenciales de Campo Local se pueden dividir en tres, las cuales se relacionan a la naturaleza de su origen (Jackson y Hall, 2017):

1. **Gamma alta**: Se encuentran entre los 60Hz a 200Hz (banda ancha) la cual es positivamente correlacionada a la tasa de disparos de las neuronas y podría reflejar la sumatoria de potenciales de acción y corrientes sinápticas asociadas con poblaciones de neuronas altamente activas y desincronizadas.

2. **Alfa y Beta**: Son dos bandas intermedias entre 7-13Hz y 15-30Hz respectivamente (banda estrecha) que son bien conocidas por su relación con ritmos sensoriales-motores.

3. **Bajas frecuencias**: Son aquellas menores a 5Hz y llamadas potenciales motores locales.

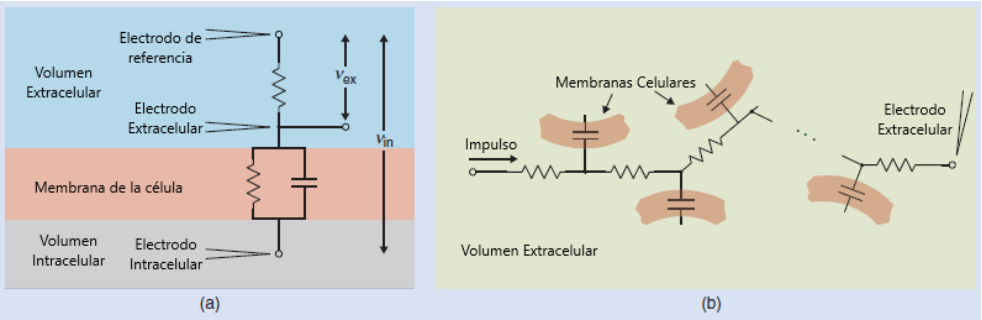


Figura 2.4: a) Modelo de circuito eléctrico de grabación extracelular. b) Atenuación de componentes de alta frecuencia en el medio extracelular. (Gibson et al., 2012)

En la figura 2.4 se muestra en la parte a) el modelo de un circuito eléctrico asumiendo que se pudiera colocar un electrodo dentro de la célula para ejemplificar el circuito del filtro pasa bajas provocado por la membrana de la célula entre el electrodo intracelular y el extracelular. Así se tiene que la señal obtenida fuera de la célula V_{ex} es igual a la derivada de la señal medida dentro de la célula V_{in}. (Gibson et al., 2012)

Ya que las frecuencias de interés de los LFPs están por debajo de los 250Hz, se recomienda digitalizar los LFPs a por lo menos 1000Hz para cumplir con el teorema de Nyquist y además conservar la habilidad trabajar con señales más rápidas y no perder tanta información de la señal ya que Nyquist define una restricción mínima (Souza et al., 2018).

2.1.7 Análisis de Local Field Potentials

En general, los datos de actividad cerebral suelen ser separadas en las distintas bandas de frecuencia descritas en 2.1.6 o clasificaciones similares sin importar el método de adquisición; ya sea EEG o LFPs. Lo anterior es debido a que múltiples estudios han revelado relación cercana entre bandas específicas de frecuencia con eventos de distinta índole, como se detallaba en las secciones previas.
Posteriormente se pueden interpretar las amplitudes en las banda de interés mediante la separación en ventanas de tiempo. Esta es una estrategia muy utilizada en el análisis de datos que sigan el comportamiento de una serie de tiempo, o sea, señales variantes en el tiempo.

Por ejemplo, (Islam et al., 2017) identificaba el movimiento voluntario de presionar un botón con el dedo índice izquierdo y derecho. En ese experimento la señales de LFP se preprocesaron utilizando varios métodos y separaron en

distintas bandas de frecuencia. Seguidamente se definieron ventanas de tiempo centradas en el evento de estímulo para enviar la señal de presionar el botón. De estas ventanas se pretendía clasificar dicho impulso de actividad neuronal extrayéndole 5 puntos de tiempo que correspondían al promedio de la amplitud de la señal en ventanas de 100ms como se observa en la figura 2.5.

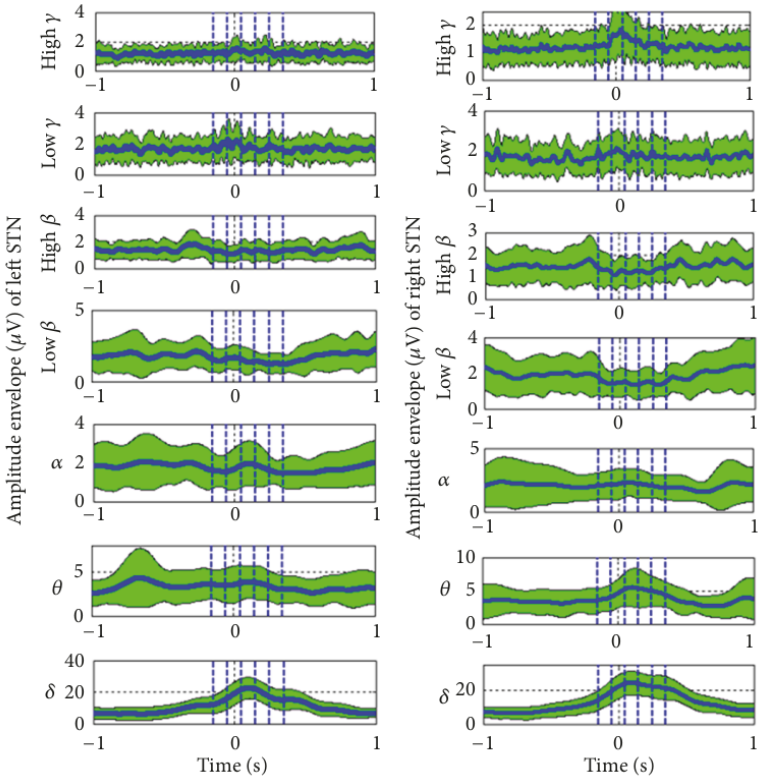


Figura 2.5: Envolvente (verde) y promedio (azul) de la señal LFP del experimento de (Islam et al., 2017) señalando las 5 ventanas de tiempo (líneas punteadas) para extraer los 5 puntos en los distintos canales y mano del evento.

Finalmente, los 5 puntos en el tiempo de la señal promedio, por cada banda de frecuencia por cada mano resultaban en los 70 "*features*" con los que se alimentarían las 3 redes neuronales propuestas para clasificar los eventos de presionar un botón entre los realizados con la mano derecha o la izquierda.
A pesar que este experimento no analiza los LFPs como series de tiempo, ya que a priori habían identificado los eventos de presionar el botón, sí sacan provecho de la forma de la onda y su variación en el tiempo para clasificarlos.

Otra estrategia es extraer componentes de grabaciones multi-canal que

reflejen la taza de disparos simultáneos de las neuronas locales capturados mediante LFPs como lo plantea (Jackson y Hall, 2017). Estos mismo autores concluyen de su estudio que aunque no se conoce aún cuáles son las características de los LFPs (amplitud, frecuencia, fase, correlación) que reflejan mejor la actividad neuronal y que tienen mayor potencial de utilizarse como biométricas; sugieren utilizar aquellas que capten mejor la estructura espacio-temporal de los LFPs. Los cuales pueden maximizar la información obtenida de múltiples canales colocados en diferentes partes de la región del cerebro.

Una vez más, dado que los LFPs son señales variantes en el tiempo, es fundamental sacar provecho de su dinámica a la hora de analizarlos e intentar extraer información de la actividad neuronal contenida en ellos.

2.2 Banco de datos a utilizar

La página web *Collaborative Research in Computational Neuroscience - Data sharing* es un medio para compartir y discutir datos y herramientas de neurociencia creado en el 2008 con la colaboración de distintos autores (Teeters y Sommer, 2019). En este sitio se encuentra una gran cantidad de datos recolectados para distintas investigación con datos de muchas partes del cerebro como el tálamo, hipocampo, corteza motora y visual; sin embargo no todas contienen las señales puras de LFP adquiridas, un tamaño suficiente como para entrenar una red neuronal apropiadamente o un dato que sea altamente correlacionado con estas señales, tal como el ángulo al que la cabeza del animal se dirigía. Los resultados de la investigación de (Mizuseki et al., 2014) fueron compartidos a través de dicha página, proporcionando un banco de datos de señales de LFP adquiridas en ratas. (Mizuseki et al., 2013)

La adquisición de las señales en bruto se hizo a una frecuencia de $20kHz$ utilizando entre 64 y 96 canales simultáneamente. Seguidamente, los datos en bruto se procesaron para extraer las señales de LFP y detectar picos. La detección de picos se logró utilizando filtro pasa bajo y el algoritmo de "Principal Component Analysis" (PCA) que permite definir agrupaciones a partir de datos de entrada dados de acuerdo a la proximidad entre los datos.

Se escogen las grabaciones del set de datos *HC-3* que contiene información de las regiones CA1 y CA3 además de la Corteza Entorrinal (EC) EC2, EC3, EC4, EC5 (Mizuseki et al., 2014).

2.2.1 Experimentos

Los datos se obtuvieron en 442 sesiones de grabación e incluye respuestas de 7736 neuronas en 11 animales mientras realizaban 8 tareas distintas, utilizando entre 31 y 127 canales. Dichos experimentos son (Mizuseki et al., 2014):

1. **Camino lineal elevado:** En este se requería que los animales corrieran ida y vuelta en búsqueda de agua en los extremos como recompensa.

2. **Espacio abierto:** Las ratas perseguían gotas de agua y piezas de Froot Loops dispersas aleatoriamente en una plataforma abierta elevada de (180cm × 180cm, 120cm × 120cm o 100cm × 200cm).

3. **Rueda:** Se recompensaba al sujeto después de correr de forma continua en una rueda durante 10 segundos.

4. **Laberinto T:** Se somete al animal a correr por 10 segundos en una rueda y al terminar tiene acceso al brazo central de una plataforma en forma de T donde podría escoger entre derecha e izquierda y era recompensado con agua si este elegía la dirección contraría a la vez anterior.

5. **Laberinto '+':** Consiste en 2 brazos abiertos y 2 brazos sin salida los cuales tienen una copa de agua en sus extremos. Una vez que la rata consumía el agua de las 4 copas, se rellenaban 2 de ellas que dependía del último brazo visitado. Si la vez anterior había consumido el agua de un brazo sin salida, el agua se rellenaba en los brazos abiertos y viceversa.

6. **Laberinto en Zigzag:** En un laberinto con 11 corredores, los animales aprendían a recorrer ida y vuelta entre 2 fuentes de agua.

7. **Rueda y Casa:** Una rueda era colocada en una jaula utilizada como la casa del animal durante los experimentos. Las ratas podían utilizar la rueda libremente sin recompensa.

8. **Sesiones de dormir:** Se grababa el animal durmiendo en su jaula casa.

2.2.2 Sondas utilizadas

En los experimento se utilizaron sondas con 4, 8, 12 o 16 puntas, con 8 canales por punta para un total máximo de 128 canales como se muestra un ejemplo en la figura 2.6.

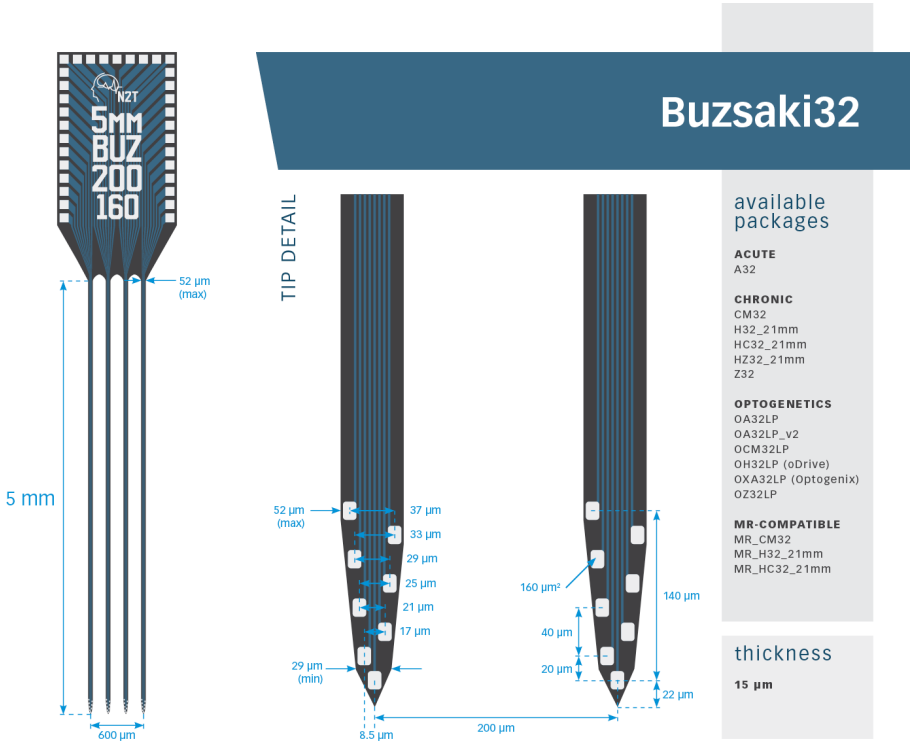


Figura 2.6: Ejemplo del tipo de sondas de adquisición de utilizadas. NeuroNexus (2020)

2.2.3 Tipos de Archivos

Los datos fueron agrupados en archivos comprimidos que contienen las sesiones hechas en un mismo día con el mismo animal y con la misma combinación de posicionamiento de los electrodos en él.(Mizuseki et al., 2013) La información fue almacenada en distintos tipos de archivos que se describen seguidamente (Mizuseki et al., 2013):

- **.dat:** Datos en bruto o "de banda ancha" amplificada por 1000x, muestreada a 20 KHz utilizando un sistema DataMax o a 32552 Hz con un dispositivo NeuraLynx y en una banda de 1Hz a 5KHz. Contiene ambos, las señales de LFP y los picos o *spikes*. Entre las sesiones grabadas con el sistema DataMax se encuentra las del animal ec014 a excepción de la ec014.n329. La frecuencia de muestreo de la sesión también está disponible en los archivos .xml. Su formato por muestra es de 2 bytes con signo (*short integers*) correspondientes a las captura de cada canal por cada muestra de tiempo. En los archivos .xml se especifica la cantidad

real de canales utilizados y no siempre es múltiplo de 8 debido a canales con problemas relacionados con alta impedancia o puntas quebradas, los cuales se removieron de los datos.

- **.xml:** Detalla la información de las grabaciones.
- **.eeg:** Contiene únicamente las señales de LFP por debajo de 625 Hz. Las señales de LFP son la parte de los datos con frecuencias por debajo de las de los spikes, y que incluyen solamente corrientes de sinapsis o de fluctuaciones de membrana. Tiene la misma cantidad de canales pero 16 veces menos muestras (1.25 KHz contra 20 KHz) o 26 veces menos (1252 Hz contra 32552 Hz).
- **.mlv ó .mpg:** Video que muestra el movimiento del animal a través de 2 LEDs colocados en la cabeza de la rata. No todas las sesiones incluyen video.
- **.whl:** Contiene la posición del animal durante la sesión. Extraído del archivo de video. Cada fila es un cuadro del video. La frecuencia de muestreo es de 39.06 Hz. Las primeras dos columnas corresponden al par ordenado (x,y) del primer LED, las últimas 2 corresponden al segundo LED. Un '-1' significa que no fue posible obtener una captura adecuada para esos cuadros.
- **.led:** Tiene grabaciones a 20KHz de los tiempos de sincronización de un LED de referencia visible en el video. Su formato por muestra es de 2 bytes con signo. Las sesiones (directorio/sesión) que incluyen este tipo de archivos son: ec014.20/ec014.260, ec014.25/ec014.350, ec014.39/ec014.723, ec014.39/ec014.729, ec014.41/ec014.765, ec014.41/ec014.771 y ec016.20/ ec016.288.

Aparte de los formatos anteriormente mencionados, existen más archivos con información de los spikes y su clasificación en distintos *clusters* creados utilizando PCA. Cualquier información adicional se puede encontrar en los archivos de documentación del banco de datos de (Mizuseki et al., 2013).

2.3 Inteligencia Artificial y Aprendizaje Automático

2.3.1 Tipos de Aprendizaje Automático

Aprendizaje Supervisado

En el aprendizaje supervisado se ajustan los parámetros de la red haciendo una comparación directa entre la salida deseada y la obtenida. Este puede verse como una retroalimentación de lazo cerrado, en donde el error es la señal retroalimentada. El error entre la señal deseada y la obtenida se utiliza para corregir la salida de la red y de esta forma guiar el proceso de aprendizaje de la red. (Du y Swamy, 2014)

Como medida del error se suele utilizar el error cuadrático medio (MSE) (Du y Swamy, 2014), aunque dependiendo del problema, puede resultar de más utilidad la medida de exactitud de una tabla de confusión.

En cuanto a la corrección del error, el objetivo final es minimizar la medida de error entre la señal deseada y la obtenida mediante la técnica de descenso de gradiente. Para esto, el algoritmo de descenso de gradiente más popular es el de **Retropropagación** o *Backpropagation* el cual hace converger el error a un mínimo local. Du y Swamy (2014)

En este caso, se proveen las posibles clasificaciones que se les puede dar a un conjunto de datos, así como las características principales que afectan la clasificación de estos datos. Duda et al. (2012)

Por ejemplo si se conoce que los datos en estudio representan la intención de mover el brazo derecho o el brazo izquierdo, se limita al clasificador a encajar los datos en alguna de estas dos categorías únicamente, basado en los electrodos con mayor amplitud en la señal.

Este tipo de aprendizaje es muy utilizado principalmente para problemas de clasificación, aproximación, control, modelado e identificación, procesamiento de señales y optimización. (Du y Swamy, 2014)

El aprendizaje supervisado, depende de datos de referencia para conocer *la señal deseada*, estos datos se utilizan para el entrenamiento y validación del algoritmo. Por lo que es necesario contar con un conjunto de datos a analizar que tenga asociado la salida de la red esperada con la cual se pueda completar el lazo de retroalimentación.

Aprendizaje No Supervisado

Estos métodos no requieren del 'valor deseado' como en el aprendizaje supervisado. Se intenta asociar automáticamente los datos de entrada basado en reglas intrínsecas de reducción dimensional. El aprendizaje no supervisado se basa únicamente en la correlación que existe entre los datos de entrada y se

busca encontrar patrones que los relacione sin ayuda externa, pero sí de un criterio que determine cuándo detenerse. (Du y Swamy, 2014)

El aprendizaje no supervisado es muy utilizado para análisis de datos, *clustering*, reducción dimensional, *feature extraction* y codificación de señales.

Aprendizaje Reforzado

Es una clase de algoritmo computacional que enseña a un agente artificial a tomar decisiones que lo lleven a tener la mayor cantidad de 'premios' o refuerzo positivo. El aprendizaje reforzado es un caso especial del aprendizaje supervisado cuando no se conoce con exactitud el resultado esperado, sin embargo el ambiente 'premia' al agente si el resultado obtenido fue una decisión acertada de acuerdo a una estimación o lo 'castiga' si tomó una decisión equivocada.

Este método es muy utilizado para control, sin embargo es muy lento. Por ejemplo, si la salida del controlador ante una determinada entrada no afecta el proceso, se toma como una buena decisión o se castiga en caso contrario.

2.3.2 Preparación del conjunto de entradas para aprendizaje supervisado

Series de Tiempo y Secuencias

Una serie de tiempo es una secuencia ordena de valores espaciada equivalentemente sobre el tiempo (Moroney, 2020).
Hay diferentes tipos de series de tiempo: Tendencias, estacionales o "seasonality" y la combinación de ambas. Así también, existen series de tiempo aleatorias, que se relacionan con ruido blanco y series de tiempo auto-correlacionadas en la cuales los datos se relacionan con una versión retrasada de sí misma.
Cada paso es dependiente de los anteriores por lo que se dice que la serie de tiempo tiene memoria. Adicionalmente, los picos o "Spikes" que son impredecibles se les conoce como *innovaciones* pues no pueden predecirse basado en los datos anteriores.(Moroney, 2020)

Por lo general, las series de tiempo van a presentar un poco de los cuatro características: tendencia, ruido blanco, estacionalidad y auto-correlación. Sin embargo, a pesar de que una tendencia o estacionalidad se mantenga a través del tiempo, pueden ocurrir eventos que la cambien drásticamente. A estas series de tiempo se les conoce como *no estacionarias*, ya que varían a través del tiempo.(Moroney, 2020)

Normalización de los datos

Cuando se pretende entrenar algoritmos utilizando características que tienen distintas escalas podría suceder que se les dé mayor relevancia en el entre-

namiento a algunas más que a otras. Esto ya que si por ejemplo 2 de las características son la estatura de una persona en metros y su edad en años, la red podría darle más peso a la edad pues sus valores son más grandes, minimizando el aporte de la estatura. Esto podría ser un problema para lograr un buen desempeño del algoritmo.
Para solucionar este problema es necesario normalizar todas las características a la misma escala de manera que se tome considere el aporte de cada una por encima de su valor real (Vasilev et al., 2019).
Entre los métodos para normalizar datos están:

Escalamiento: A todos los datos se les aplica la fórmula 2.1 para ajustarlos al rango [0, 1].

$$x = \frac{x - x_{min}}{x_{nax} - x_{min}} \tag{2.1}$$

Estandarización: Se aplica la fórmula 2.2 asumiendo que los datos siguen una distribución normal, para lograr centrarlos en cero.

$$x = \frac{x - \mu}{\theta} \tag{2.2}$$

2.4 Redes Neuronales

2.4.1 Perceptrón

Es un algoritmo de clasificación similar a una regresión. Como tal, es una función que aplica el producto punto entre un vector de entrada $\vec{x}$ y un vector de pesos $\vec{w}$: $f(\vec{x} \cdot \vec{w})$. La diferencia entre un perceptrón y una regresión es que la salida es una simple función escalón. De manera que si $f(\vec{x} \cdot \vec{w}) > 0$ entonces $f(\vec{x} \cdot \vec{w}) = 1$, o $f(\vec{x} \cdot \vec{w}) = 0$ en caso contrario (Vasilev et al., 2019). En la figura 2.7 se ilustra la estructura de un perceptrón.

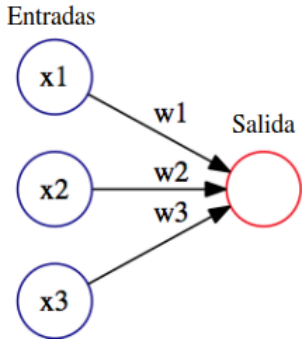


Figura 2.7: Un perceptrón con tres unidades de entrada (neuronas) y una neurona de salida (Vasilev et al., 2019).

2.4.2 Red Neuronal Prealimentada: *Feedforward NN*

Ya que el perceptrón por si solo no lograba aprender de la manera esperada, surgió la idea interconectar múltiples perceptrones organizadas en capas secuenciales (una capa de entrada, una o más capas escondidas y una capa de salida) para formar una "perceptrón multicapa clásico" o simplemente *Multilayer Perceptron (MLP)*.
La salida del perceptrón (también llamado: unidad, neurona o nodo) se conecta a todos los nodos de la siguiente capa. Primero, todas las entradas se reciben en la capa de entrada y se calculan las salidas y_i según su función de activación para cada unidad de la primer capa. Se propaga su salida hacia adelante como entrada de las neuronas de la siguiente capa, por eso el término *feedforward*, y así hasta la capa de salida (Vasilev et al., 2019).

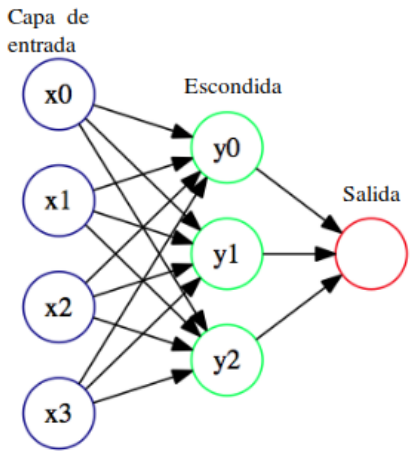


Figura 2.8: Red neuronal con una capa escondida. (Vasilev et al., 2019)

Cuando se utiliza un MLP que tiene una o incluso 2 capas escondidas, como la de la figura 2.8, se le llama una *Shallow Neural Network* o red neuronal poco profunda. Para más de dos capas se les llama *Deep NN* o redes neuronales profundas.

2.4.3 Entrenamiento de Redes Neuronales

Con respecto al entrenamiento de redes neuronales, lo más común es utilizar el algoritmo de descenso de gradiente en combinación con el de *backpropagation*. El entrenamiento de redes es un proceso iterativo, ya que una sola iteración producirá un resultado con un error muy grande comparado al valor esperado. Para esto, se usa el algoritmo de optimización de **Descenso de Gradiente** que consiste en minimizar el error del resultado de la red. Sin entrar en mucho detalle de la matemática detrás de este proceso, se calcula la primera derivada de la función de costo J que corresponde al error entre el valor esperado y el obtenido, para encontrar el mínimo global. Como la entrada a las redes es un vector $\vec{x}_i$ de n elementos, se calcula el gradiente de la función de costo, que sería la primer derivada con respecto a todas las variables de la función de costo. Se suelen utilizar como **función de costo** el Error Cuadrático Medio, el Error Absoluto Medio, entre otros.
El error encontrado en el resultado de la red entonces, indica cuánto se debe corregir cada peso que multiplica las distintas entradas de la capa anterior para entonces llegar al mínimo global. Cada neurona aplica el mismo procedimiento para corregir su resultado "propagando el error hacia atrás", método que se le conoce como ***backpropagation***. De esa manera, la red ajusta sus pesos y repite el proceso utilizando el mismo conjunto de datos de entrada, esperando

que idealmente el error resultante sea menor al de la iteración anterior. A cada iteración sobre todo el conjunto de datos se le conoce como ***epoch***, o época. Este proceso se termina cuando se alcanza el mínimo global o se alcanza la cantidad de épocas predefinidas (Vasilev et al., 2019).

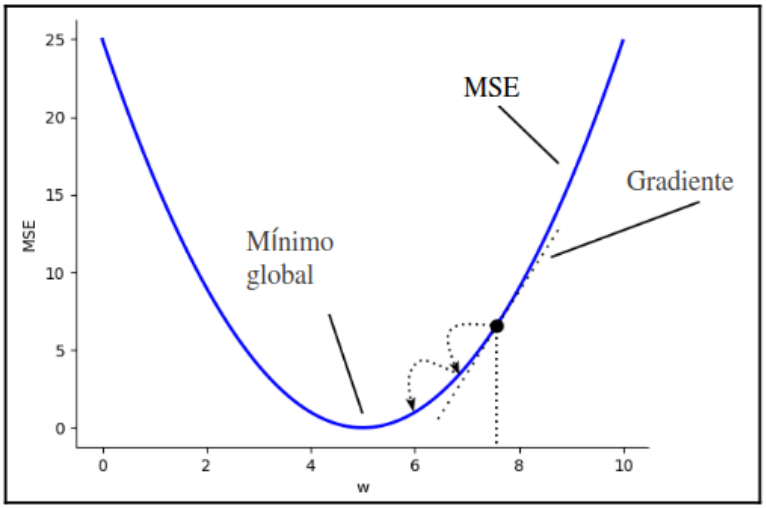


Figura 2.9: Proceso de aprendizaje por descenso de gradiente (Vasilev et al., 2019).

Sin embargo, podría suceder que el ajuste que se calcula para los pesos sea mayor al requerido, de manera que nunca se alcance el mínimo global porque se estaría "rebotando" de una pared a otra de la curva de la figura 2.9. Para evitar esto, se define el parámetro de ***learning rate*** η que funciona como un factor para escalar el ajuste que se le debe hacer a los pesos de la red. De manera que si se usa un *learning rate* bajo, se tardará más en converger el error al mínimo global pero es probable alcanzarlo como en la figura 2.9; y si se usa uno muy grande se corre el riesgo de nunca converger como se observa en la figura 2.10.

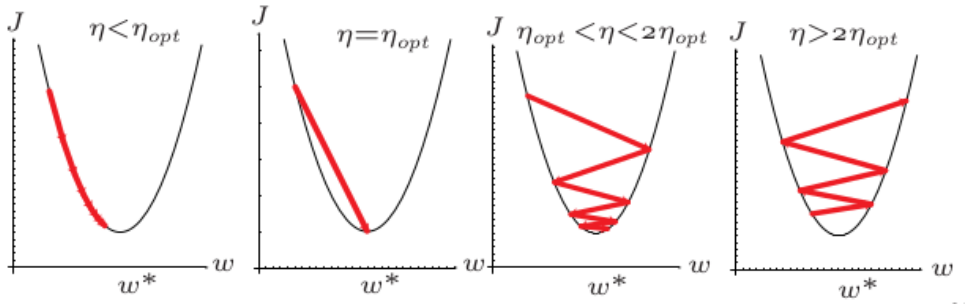


Figura 2.10: Escenarios de convergencia con distintos valores de *learning rate* η comparado con el valor de η óptimo (Duda et al., 2012)

Ambos algoritmos presentan variaciones para ajustarse a otros tipos de redes neuronales que tienen otra estructura de conexión de nodos como las RNN o las CNN.

Existe un problema con este proceso iterativo de entrenamiento, y es que el ajuste de los pesos ocurre una vez que se calcula el error para todas las muestras de entrenamiento. Por ejemplo, se podría querer clasificar imágenes de perros y gatos proporcionándole a la red 1000 imágenes de perros, 1000 de gatos. La red obtendrá un resultado por cada imagen de muestra y calculará el error. Finalmente, una vez calculados los 2000 errores, se podrá aplicar la función de costo (por ejemplo un MSE), calcularle el gradiente y ejecutar el proceso de ajuste de los pesos de la red. De forma que para mejorar el rendimiento de la red se tuvo que esperar que se alimentara la red con todas las muestras, lo cual puede resultar muy lento especialmente si se trata de conjuntos de datos muy grandes.

El extremo opuesto al sistema de entrenamiento de Descenso de Gradiente es el *Stochastic Gradient Descent (SGD)* que ajusta los pesos después de cada muestra. El problema con este enfoque es que puede producir un entrenamiento mucho más lento.

Por este motivo se introduce la metodología de entrenamiento por ***mini-batch***, el cual consiste en separar el conjunto total de muestras en lotes más pequeños del mismo tamaño, como se ilustra en la figura 2.11. Así, se puede alimentar la red con subconjuntos más manejables de datos y realizar el ajuste de los pesos una vez terminado cada lote. De nuevo, una vez que se entrena la red con todos los lotes (*batches*), se puede iterar de nuevo sobre todos los lotes, lo que correspondería a una nueva época de entrenamiento (Vasilev et al., 2019).

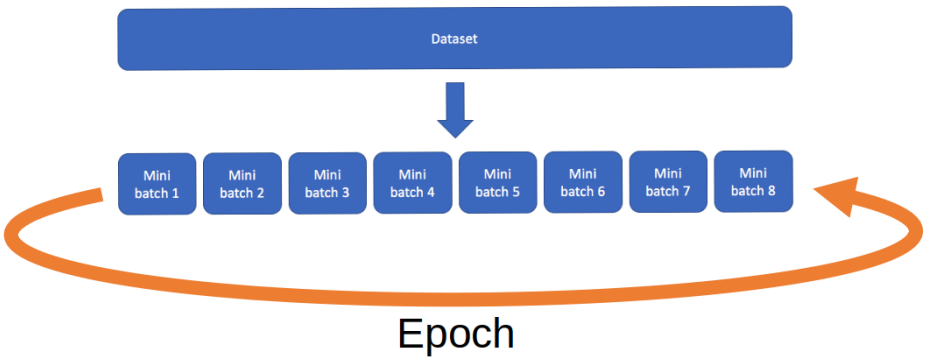


Figura 2.11: Descenso de gradientes por mini-lotes (Rojas, 2019)

Regularización

Si se entrenan las redes por muchas épocas para lograr un error prácticamente nulo, se corre el riesgo de que la red se sobre-ajuste al conjunto de datos de entrenamiento fenómeno conocido como *overfitting*. Esto es contraproducente pues entonces la red tendrá un desempeño pobre ante conjuntos de datos nuevos o desconocidos para ella.
Se suelen aplicar técnicas para evitar este sobre-ajuste y se conocen como regularización. La **regularización** en AI, es una forma de limitar la capacidad de aprendizaje de una red neuronal. Entre las técnicas más conocidas de regularización está el poner un valor máximo a los pesos entre nodos, el cual no pueden sobrepasar. Otra forma es eliminando (no utilizando el resultado de) nodos de forma aleatoria durante una época, cambiando los nodos eliminados entre una época y otra, con el fin de que los nodos utilizados, asuman el aprendizaje de los faltantes; este método se conoce como ***dropout***. (Rojas, 2019).

2.4.4 Funciones de Activación

Se puede determinar de la sección 2.4.1 que la función $f(x)$ corresponde a una función lineal, y aunque se interconecten varios perceptrones para formas estructuras más complejas como se detallará más adelante, el resultado seguirá siendo una función lineal. Esto es una limitante pues aunque una red de perceptrones es capaz de clasificar datos en clases linealmente inseparables, si el resultado sigue siendo el de una regresión lineal que no es capaz de determinar a qué clase pertenece sin algún paso extra.
Para transformar estas redes en funciones no lineales se deben usar funciones de activación no lineales en cada perceptrón. O sea, sustituir $f(\vec{x} \cdot \vec{w})$ por algu-

na de las siguientes funciones de activación no lineales más comunes (Vasilev et al., 2019):

- **Activación por Umbral:**

$$f(a) = \begin{cases} 1 \, si \, a \geq 0 \\ 0 \, si \, a < 0 \end{cases} \tag{2.3}$$

- **Logística o Sigmoide:** Esta es la más utilizada comúnmente. Se puede interpretar como la probabilidad de activación de la neurona.

$$f(a) = \frac{1}{1 + e^{-a}} \tag{2.4}$$

- **Tangente hiperbólica (tanh):**

$$f(a) = \frac{e^{a} - e^{-a}}{e^{a} + e^{-a}} = \frac{1 - e^{-2a}}{1 + e^{-2a}} \tag{2.5}$$

- ***Rectified Linear Unit* (ReLU):**

$$f(a) = \begin{cases} a \, si \, a \geq 0 \\ 0 \, si \, a < 0 \end{cases} \tag{2.6}$$

Inicialmente la función de la ecuación 2.3 fue muy utilizada pero luego sustituida por la función logística, tangente hiperbólica y ReLU representadas en las ecuaciones 2.4, 2.5 y 2.6 respectivamente. Estas últimas 3 difieren en su rango de operación y el comportamiento de su derivada durante en el entrenamiento de la red (Vasilev et al., 2019).

Ya que el gradiente de la función de activación es un factor a considerar en el entrenamiento de las redes neuronales, esta segunda diferencia resulta de interés.
La derivada de la función logística es $f'(a) = f * (1 - f)$, mientras que la de tangente hiperbólico es $f'(a) = (1 + f) * (1 - f)$. Por otro lado, la de la función ReLU es mucho más fácil de calcular ya que está dada por la ecuación 2.7:

$$f'(a) = \begin{cases} 1 \, si \, a \geq 0 \\ 0 \, si \, a < 0 \end{cases} \tag{2.7}$$

Como se comentará en la sección 2.4.6, después de varias iteraciones el ajuste que se debe hacer a algunos pesos de la red se vuelven despreciables dependiendo de la función de activación que se utilice. A este efecto se le llama **desvanecimiento del gradiente** y la función ReLU presenta una ventaja

ya que logra atenuar este problema pues su derivada (de la ecuación 2.7) es constante y no tiende a cero cuando a se hace cada vez más grande (Vasilev et al., 2019).

2.4.5 Redes Neuronales Convolucionales

Las redes neuronales convolucionales (CNN) son una topología distinta a la de perceptrones densamente conectados, en los cuales cada nodo de una capa está conectado a todos los nodos de la capa anterior. Las CNN utilizan patrones simples para generar patrones más complejos a partir de ensambles de la información de entrada. Esta técnica es muy utilizada para clasificación y reconocimiento de patrones en imágenes, ya que aprovechan la localidad espacial de la información de entrada, a diferencia de una NN densamente conectada, la cual trata pixeles lejanos con la misma importancia que pixeles cercanos. Las redes neuronales convolucionales se basan en tres ideas principales: Campos receptivos locales, pesos compartidos y reducción (Nielsen, 2015).

Campos receptivos locales: En redes neuronales densamente conectadas, se tiene capas de neuronas alineadas verticalmente, para las CNN es más sencillo pensar en que la capa de entrada es de nxn neuronas, siendo n la cantidad de pixeles en filas y columnas de la imagen. Las capas escondidas no reciben la salida de todas las neuronas de la capa anterior, sino solamente una porción de la imagen, por ejemplo, un cuadro de 5x5 pixeles. La conexión de una neurona particular en una CNN puede apreciarse en la figura 2.12.

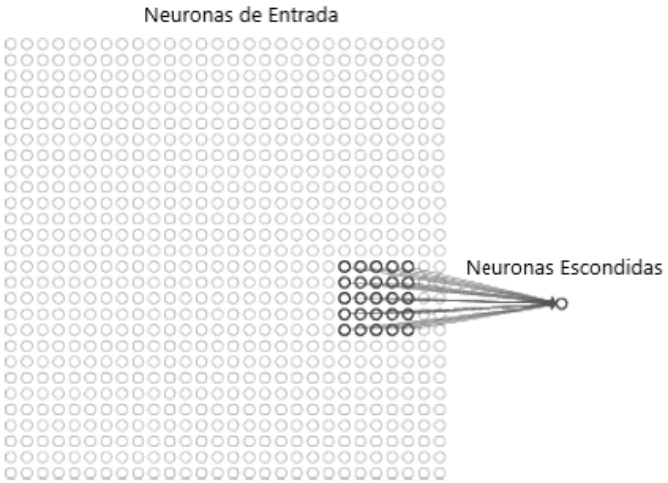


Figura 2.12: Conexión entre capas de una Red Neuronal Convolucional (Nielsen, 2015).

La ventana de pixeles de las neuronas de entrada se conoce como el *campo receptivo local* para la neurona de la capa escondida. Podría decirse que cada neurona de la capa escondida aprende a identificar una pequeña región de la imagen. Nótese que en un ejemplo de una imagen 28x28, utilizando un *kernel* de 5x5, la primer capa escondida será de un tamaño 24x24, ya que solo se podrá desplazar la ventana 23 veces a la derecha o hacia abajo. Pero, si en lugar de desplazarse 1 pixel a la vez se mueve la ventana de 2 en 2, entonces se estaría utilizando una *longitud de zancada* de 2.

Pesos y sesgos compartidos: Ahora, cada neurona de la primer capa escondida procesará la información de una región específica de la imagen, pero utilizando el mismo peso y sesgo, de forma que la salida de la neurona escondida j, k-ésima es:

$$\sigma\Big(b + \sum_{l=0}^{4}\sum_{m=0}^{4} w_{l,m} a_{j+l,k+m}\Big) \tag{2.8}$$

Con σ como la función de activación, generalmente un sigmoide, b el sesgo o bias, $w_{l,m}$ es un arreglo 5x5 de pesos, y $a_{x,y}$ denota la activación en la posición x, y. Al aplicar la misma función en los 24x24 nodos escondidos, es posible reconocer la misma característica o *feature* en todos ellos. En otros términos, las CNN se adaptan bien a cambios ligeros en imágenes.

El conjunto de pesos y sesgos compartidos se dice que definen un *kernel* o filtro (ver figura 2.13). Al utilizarse distinto kernels para la misma imagen se crean distintos *mapas de características*, los cuales son necesarios para hacer reconocimiento en imágenes, haciendo uso de todas las características encontradas con los distintos kernels.

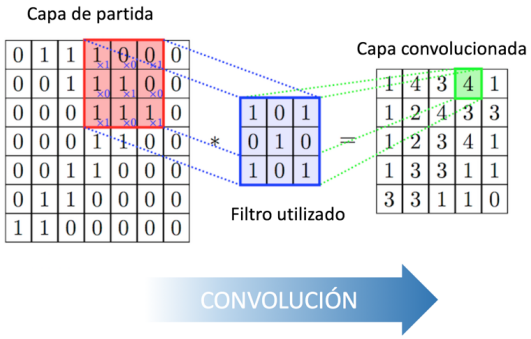


Figura 2.13: Capa Convolucional. (Calvo, 2015)

Una ventaja de las CNN en comparación con las redes densamente conectadas es que la cantidad de pesos a calcular es considerablemente reducida

gracias al uso de los campos receptivos locales, resultando en tiempos de entrenamiento más pequeños que permitiría hacer redes más profundas.

Si se presta atención, la ecuación 2.8 es una convolución, que en otra notación podría escribirse como $a^1 = \sigma(b + w * a^0)$, donde a^1 denota el conjunto de salidas de activación de un mapa de características, a^0 el conjunto de entradas de activación y $*$ el operador de convolución.

Capas de reducción: Generalmente se utilizan justo después de una capa de convolución para simplificar la información de la capa de convolución anterior. Las capas de reducción o *pooling*, toman los distintos mapas de características y los condensan en uno solo.

Una neurona en la capa de reducción puede analizar una región de 2x2 neuronas de la capa anterior. Una técnica común conocida como *max-pooling*, en la cual una unidad entrega como salida la máxima activación de una región de 2x2, como se muestra en la figura 2.14.

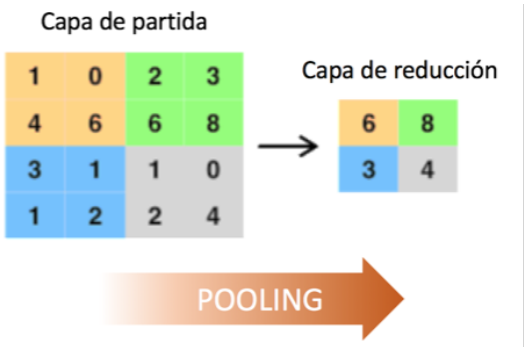


Figura 2.14: Capa de Reducción. (Calvo, 2015)

Nótese que si se tenían 24x24 neuronas en la capa de convolución, en la capa de reducción se tendrán 12x12 unidades. Recordando que una capa de convolución involucra más de un mapa de convolución (por ejemplo 4 mapas), se aplica max-pooling a cada mapa de características por separado. Entonces, la combinación de capas convolucionales y de reducción se vería como en la figura 2.15:

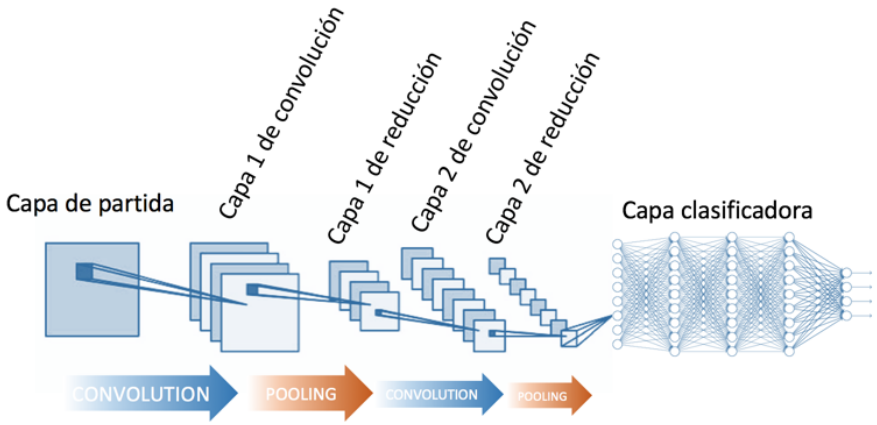


Figura 2.15: Red Neuronal Convolucional. (Calvo, 2015)

Otro tipo de reducción diferente a max-pooling, es la reducción L2, que en lugar de tomar el valor máximo del grupo escogido, calcula la raíz cuadrada de la suma del cuadrado de las entradas de activación (Nielsen, 2015).

Al igual que en redes densamente conectadas, para optimizar el resultado de la clasificación y reducir el costo, se utiliza *backpropagation* y *stochastic gradient descent* para ajustar los pesos. Sin embargo, es necesario ajustar la derivada de ajuste ya que una CNN no es densamente conectada.

2.4.6 Redes Neuronales Recurrentes

"El cerebro es una estructura fuertemente recurrente. Esta recurrencia masiva sugiere un rol mayor de dinámicas de retroalimentación en procesos de percibir, actuar y aprender, y en mantener vivo el organismo. Las redes recurrentes aprovechan el poder de computación basada en el cerebro". (Du y Swamy, 2014)

El texto citado anteriormente refuerza la motivación de explorar el uso de la Redes Neuronales Recurrentes (RNN) para el análisis de señales de LFP cerebrales, pues además son datos dependientes del tiempo, y de los cuales se puede sacar provecho de su localidad temporal.

Las RNN tienen la ventaja de ser *Turing Complete*, lo que implica que son capaces de aplicarse a cualquier problema, ya que teóricamente pueden simular cualquier programa (Vasilev et al., 2019).

Las RNN tienen al menos un lazo de retroalimentaación, y por lo general suelen ser redes más compactas que la redes neuronales *feedforward*. Para sistemas dinámicos, es necesario predecir la entrada del tiempo $t+1$ en el tiempo t y así crear una red de asociación temporal, las cuales requieren de recurrencia

para poder manejar la relación en el tiempo. Para generar una red neuronal dinámica, se necesita de memoria, cuya unidad básica suele representarse por un retraso en tiempo discreto cuya función de transferencia es: $H(z) = z^{-1}$. Un conjunto de retrasos en serie son la base de lo modelos lineales dinámicos como las Respuestas Finitas al Impulso (FIR).

Un MLP puede volverse dinámico introduciendo lazos de retrasos en las capas de entrada, escondidas y/o de salida. Una arquitectura de este tipo, que involucra retrasos se conoce como red neuronal de retardo de tiempo.

Existen redes de recurrencia global o de recurrencia local y globalmente de avance (*locally recurrent, globally feedforward network*) (Du y Swamy, 2014). Las redes de recurrencia global por lo general presentan problemas de estabilidad durante el entrenamiento además de requerir algoritmos muy complicados para su entrenamiento ya que la estrategia de retropropagación no aplica como en un MLP pues la conexión entre neuronas es diferente. En contraste, las redes de recurrencia local implementan retroalimentaciones internas en las neuronas, las cuales se conectan estrictamente hacia adelante como una red neuronal convencional.

Las neuronas de las redes neuronales recurrentes, a diferencia de las redes usuales, no solo toman en cuenta la entrada a la red, si no también el estado anterior s_{t-1} de sí mismas de la siguiente forma:

$$s_t = f(s_{t-1}, x_t) \tag{2.9}$$

Donde f es la función de activación, s_t es el vector de estados actuales de la red en el instante t y x_t es la entrada a la red en el instante t (Vasilev et al., 2019). Se puede interpretar s_{t-1} como la sumatoria de todas las entradas anteriores de la red. Este lazo de retroalimentación se podría desenrollar como en se muestra en la figura 2.16.

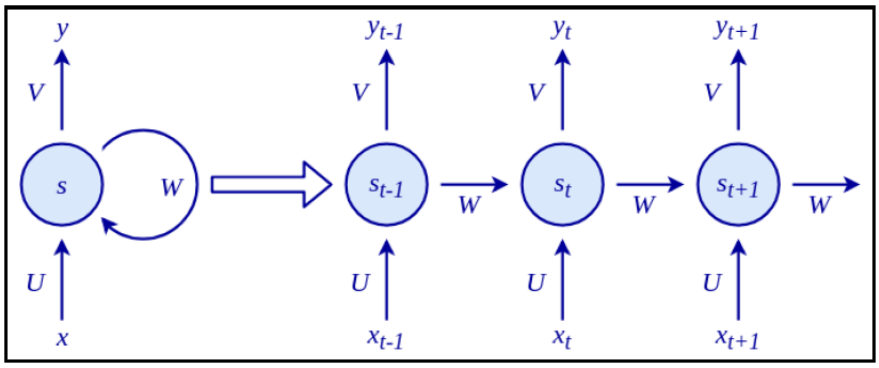


Figura 2.16: Red Neuronal Recurrente. (Vasilev et al., 2019)

En donde la RNN tiene 3 tipos de pesos:

- U Que transforma la entrada x_t al estado s_t.
- W Transforma el estado anterior s_{t-1} al estado actual s_t.
- V Mapea el nuevo s_t calculado a la salida y_t.

Las transformaciones son por lo general sumatorias con los respectivos pesos, de manera que se puede definir ahora el estado interno y la salida y_t como:

$$s_t = f(s_{t-1} * W + x_t * U) \tag{2.10}$$
$$y_t = s_t * V \tag{2.11}$$

Donde f es una función de activación por ejemplo ReLU, tangente hiperbólica o sigmoide. En teoría, las RNN almacenan la información de todos los estados anteriores, sin embargo, también sufren del fenómeno de 'desvanecimiento del gradiente' en el cual la corrección de un peso de la red es tan pequeño que no modifica el comportamiento de la red. Para este problema, se pueden seleccionar funciones de activación que no se saturen como sí lo hace una sigmoide, por ejemplo ReLU.

Además, la RNN de la figura 2.16 muestra el comportamiento de un solo nodo de la red, estos pueden distribuirse en distintas capas. Sin embargo, las capas de una RNN no suelen ser densamente conectadas. Según la relación de entradas con salidas se puede tener conexiones (Vasilev et al., 2019):

- Uno a uno: Esta es un procesamiento no secuencial, pues ocurre en un solo instante de tiempo, por ejemplo la clasificación de imágenes.
- Uno a muchas: La cual entrega como salida una secuencia a partir de una sola entrada.
- Muchas a una: La cual entrega una sola salida a partir de múltiples entradas. Esta es posiblemente la configuración que mejor se acopla al problema propuesto de predicción de un ángulo a partir de múltiplos datos de la señal de LFP.
- Muchas a muchas indirectas: En las cuales se pueden tener más salidas que entradas o viceversa, como en traducción de idiomas.
- Muchas a muchas directa: Como sucede en reconocimiento de voz.

Redes *LSTM*

Las redes neuronales de Memoria Larga de Corto Plazo o *LSTM*, por sus siglas en ingles, son un caso particular de las redes neuronales recurrentes que fueron introducidas por Hochreiter y Schmidhuber para disminuir el problema del desvanecimiento del gradiente en las RNN. (Vasilev et al., 2019)

El desvanecimiento de gradiente se produce después varias iteraciones en el entrenamiento de la red, donde el gradiente decae exponencialmente, viéndose opacado por los gradientes más recientes y la habilidad de la RNN para retener la historia de sus estados anteriores es desvanecida. Este efecto produce que teóricamente las RNN sean capaces de *recordar* estados anteriores y que no dependa solamente de la entrada actual; pero que en la práctica sea capaz de recordar solamente unos cuantos pasos previos. Este problema es común en redes neuronales regulares, pero es especialmente notorio en RNN's.

De manera que, un nodo de una red LSTM representada en la figura 2.17 se compone de las compuertas de olvido, entrada y salida: f_t, i_t y o_t respectivamente, c_t como la memoria del estado del nodo en el momento t, mientras que c'_t corresponde al estado candidato de la celda. Finalmente x_t y h_t como la entrada y salida del nodo en el momento t. Las compuertas de olvido y entrada son capas completamente conectadas con función logística de salida, lo que quiere decir que tendrán salidas básicamente binarias que permitan o impidan considerar sus datos de entrada en el cálculo de la salida y la memoria de estado de la celda.

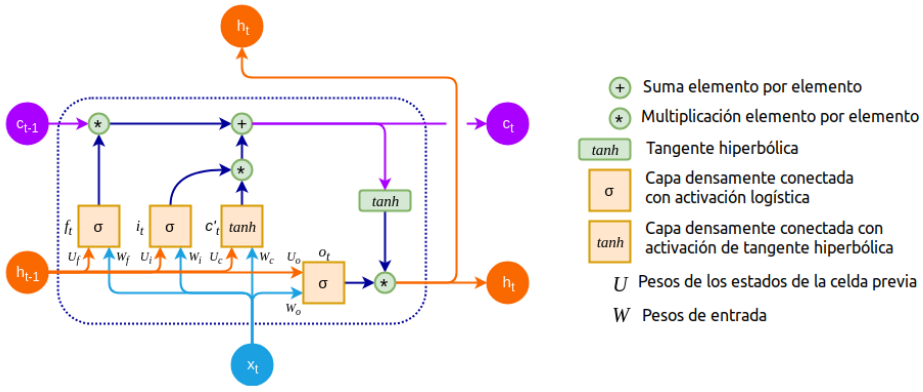


Figura 2.17: Celda de una red LSTM. (Vasilev et al., 2019)

Del diagrama anterior se pueden obtener las ecuaciones que definen cada una de las variables de la red. Sin embargo, es importante destacar principalmente la que corresponde al cálculo de la memoria de estado del nodo:

$$c_t = f_t * c_{t-1} \oplus i_t * c'_t \tag{2.12}$$

Nótese que si en la ecuación 2.12 la compuerta de olvido $f_t = 1$ y la compuerta de entrada $i_t = 0$ el estado de la celda se mantendrá exactamente igual al anterior, lo cual es precisamente lo que permite que una red LSTM memorice secuencias largas evitando así el desvanecimiento de gradientes. Solamente la compuerta de olvido es capaz de borrar la memoria de la red basando su comportamiento en la salida de la celda anterior h_{t-1} y la entrada x_t.

2.4.7 Métricas de Evaluación

El objetivo es encontrar el modelo más simple que se acople adecuadamente al problema a resolver con una exactitud aceptable y sea suficientemente flexible y generalizable a datos nuevos. El rendimiento de generalización de una red, determina una métrica de calidad del modelo seleccionado. (Du y Swamy, 2014).

Una métrica de rendimiento puede definirse como una construcción lógica matemática diseñada par medir qué tan cerca están los resultados obtenidos de los esperados o reales (Botchkarev, 2019).
De acuerdo al problema en cuestión, existen métricas que permiten evaluar el rendimiento del modelo de aprendizaje automático y estas se pueden separar en los casos de clasificación y regresión. El Área Bajo la Curva (AUC por sus siglas en inglés) de una gráfica ROC (del inglés *Receiver Operating Characteristic*), así como matrices de confusión, la exactitud, precisión y el *recall* son métricas muy utilizadas en términos de clasificación, ya que proveen relaciones entre la cantidad de clasificaciones acertadas contra erróneas. Mientras que en problemas de regresión, donde existe solo una predicción correcta e infinita cantidad de predicciones erróneas, la información relevante no es cuántas veces la predicción fue equívoca, ya que sería la mayoría de los casos, sino qué tan lejos estuvo del valor real. Es por esto que el Error Absoluto Medio(MAE), el Error Cuadrático Medio (MSE) y su Raíz Cuadrada (RMSE) son métricas más adecuadas para evaluar la calidad de algoritmos de regresión. (Rojas, 2019)

Entre las más populares para regresión están MAE, RMSE y el Error Porcentual Absoluto Medio (MAPE), teniendo la segunda características intrínsecas útiles como su indiferencia ante el signo del error así como su capacidad de destacar *outliers* o muestras muy lejanas al dato esperado por su potencia al cuadrado; además de ser simple de calcular por una computadora lo cual facilita su optimización. Dichas características son ideales para utilizarla además como función de costo a minimizar mediante el algoritmo de descenso de gradiente que permite a las redes neuronales realizar correcciones con base en la función de costo.

Según el análisis de (Botchkarev, 2019), la métrica de evaluación debería ir acorde a la naturaleza de los datos bajo estudio. De manera que datos dimensionados por una unidad como kilómetros o segundos, deberían ser analizadas con métricas como el MAE o RMSE que van proporcionan un resultado en la misma unidad. Mientras que datos sin dimensión como decibeles, grados, radianes o porcentajes que no tiene una unidad relacionada y son independientes de escala, se deberían medir utilizando por ejemplo MAPE.

3 Desarrollo

3.1 Selección de las grabaciones del banco de datos

En cada uno de los animales se recolectó información de distintas regiones del cerebro y a su vez, distinta cantidad de neuronas fueron grabadas. Los 3 animales con más información capturada contemplando regiones y cantidad de neuronas muestreadas son "EC013", "EC016" y "EC014".

El "EC014" particularmente contiene información de las regiones de la corteza entorrinal 2, 3 y 5, así como del CA1. De acuerdo con (Taube, 2009) y (Buzsaki, 2011), estas zonas del cerebro contienen células de dirección de cabeza en primer lugar; y en segundo lugar neuronas piramidales de lugar. Además, dichos autores resaltan el flujo de información entre el EC y el hipocampo, así como el principal circuito del hipocampo de la figura 2.2 que tiene como principal entrada el EC2 y como principal salida el CA1, involucrando a su paso la gran red recurrente del EC5 como se observa en la figura 2.2.

Solamente algunos sets de datos proporcionados en (Mizuseki et al., 2013) contienen las señales de LFP en formato .eeg, la información de la posición de los LEDs colocados en la cabeza del animal y los datos del LED de sincronización. Es por estas razones que se decide comenzar la investigación a partir de 2 de sesiones del sujeto **ec014**. Dichas sesiones se encuentran en el directorio **ec014.41** del banco de datos HC3 de (Mizuseki et al., 2013) y corresponden a **ec014.765** y **ec014.771**, las cuales se detallan a continuación.

Animal	EC014
Directorio	ec014.41
Experimento	MidSquare
Familiaridad	3
Sesiones	ec014.765 ec014.771

Cuadro 3.1: Información General del banco de datos

En el cuadro 3.1 la Familiaridad corresponde al número de veces que el animal a realizado el experimento en cuestión. En este caso *MidSquare* es el experimento de Espacio Abierto mencionado en la sección 2.2.1 utilizando una plataforma de $120cm$ x $120cm$ el cual se representa en la figura 3.1.

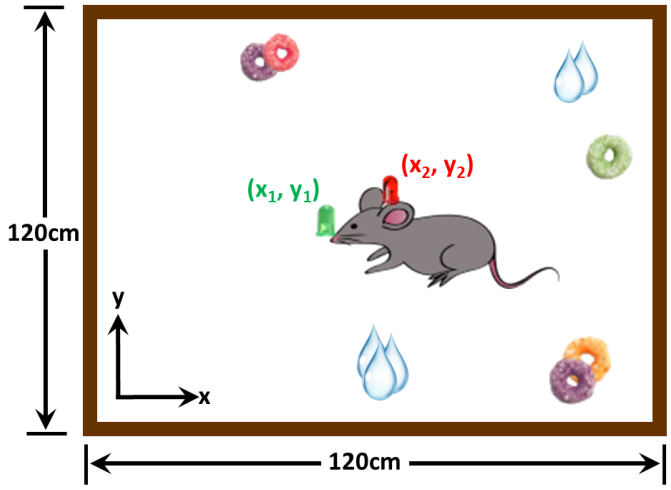


Figura 3.1: Representación del Experimento

En la figura 3.1 se ilustra uno de los sujetos con los 2 LEDs que permiten ubicar la cabeza del animal con respecto a un punto fijo el cual sería el eje coordenado de la esquina inferior izquierda, el cual era un LED de referencia. La rata se colocaba en este espacio abierto donde había agua y comida a su alrededor para incentivar su movimiento en el espacio.

Sesión	ec014.765	ec014.771
ID	990	996
Duración (s)	2811.494	2746.778
Tamaño (GB)	1.8	1.42
Extensión Video	mpg	mpg
Tamaño Video (MB)	601.8	437.9

Cuadro 3.2: Información de los archivos de las sesiones.

Ambas sesiones tienen una duración cercana a los 46 minutos como se detalla en el cuadro 3.2 donde además se mencionan lo identificadores únicos de cada sesión para ubicarlos apropiadamente en las tablas preparadas por los autores las cuales contienen características de cada grabación como la duración, familiaridad, experimento, cantidad de *spikes* por célula identificada, posición de los electrodos, etc.

Tomando en cuenta que los datos de actividad neuronal fueron adquiridos utilizando el sistema DataMax a $20KHz$ y los datos de posición de los LEDs fueron muestrados a $39{,}06Hz$, se considera suficiente cantidad de datos como para alimentar la red, incluso tomando en cuenta la presencia de datos de

posición inválidos por oclusión de los LEDs y de más factores ambientales. Es necesario recordar que los datos de estas sesiones fueron guardados en formato .eeg, lo cual significa que contienen 16 veces menos muestras que los datos crudos originales, ya que fueron diezmados de $20KHz$ a $1{,}25KHz$ según (Mizuseki et al., 2013).

Región del Cerebro	Células		Cantidad de *Spikes*			
	Tipo	Total	765	(%)	771	(%)
CA1	Piramidal	12	18947	13.55	14661	11.12
	Interneurona	2	120878	86.45	117158	88.88
	Indefinida	0	0	0.00	0	0.00
EC2	Piramidal	4	24233	31.08	21625	24.64
	Interneurona	1	53737	68.92	66133	75.36
	Indefinida	0	0	0.00	0	0.00
EC3	Piramidal	4	15700	8.61	18561	11.87
	Interneurona	2	166637	91.39	137757	88.13
	Indefinida	0	0	0.00	0	0.00
EC5	Piramidal	2	4575	18.47	4978	45.14
	Interneurona	1	12238	49.41	1640	14.87
	Indefinida	0	7955	32.12	4411	39.99

Cuadro 3.3: Información de las células grabadas

Con el propósito de analizar mejor los detalles del sujeto bajo estudio y específicamente las grabaciones propuestas, en el cuadro 3.3 se observa primeramente la gran cantidad de células piramidales en comparación a las interneuronas. Esto es deseable ya que las células de dirección de la cabeza así como las células de lugar son piramidales y las que contienen la información requerida en la presente investigación. Por otro lado, cerca de la mitad están en el áreas CA1, región del hipocampo donde se concentran las células de lugar, mientras que en las regiones de la EC está el resto que corresponden a *HD cells*.

Sin embargo, también es notable que la mayor actividad es de las interneuronas, las cuales hacen circular la información obtenida de células piramidales por el circuito de la figura 2.2; además de comunicar distintas regiones del cerebro. De forma que, ante la necesidad de escoger entre los distintos canales, los que estén ubicados en las regiones EC y particularmente en la EC2 podrían ser los más adecuados ya que es donde se encuentran las células de dirección de la cabeza y además la que tiene mayor porcentaje de actividad de ellas en comparación con las interneuronas. Del cuadro 3.4 se observa que la sonda 9 es la que se ubica en la región EC2 y la que contiene entonces los electrodos capaces de capturar mayor cantidad de actividad relacionada a la orientación

del sujeto.

Con respecto a la definición de los *features* que alimentarán las distintas redes neuronales para entrenarlas en la predicción del ángulo al cual la cabeza del sujeto se dirige, es importante contemplar la mayor cantidad de información disponible. Utilizar la mayor cantidad de canales como entradas de la red puede ser una opción a considerar, siempre y cuando se reduzca al máximo la información redundante que podría ralentizar el entrenamiento al requerir más cálculos de los pesos entre nodos y dificultar la obtención de un modelo suficientemente bueno para cumplir con una predicción lo más cercana posbile al dato real.

Número de Sonda	Región del Cerebro
1	CA1
2	
3	
4	
5	
6	
7	
8	
9	EC2
10	EC3
11	
12	EC5

Cuadro 3.4: Ubicación de las sondas en el cerebro.

La posición de las sondas se puede observar en el cuadro 3.4. Se debe aclarar que el banco de datos presenta un total de 99 canales, 3 de ellos contienen información muy distinta la del resto. Como se mencionó en la sección 2.2.3, algunos electrodos no se incluyeron en el conjunto de datos por fallas o errores en sus mediciones debido a distintas razones allí detalladas. Por lo tanto, ya que cada una de estas 12 sondas contiene 8 electrodos, se tiene un total de 96 canales, a sabiendas de que algunos de los que se utilizaron no se incluyeron en los archivos finales.

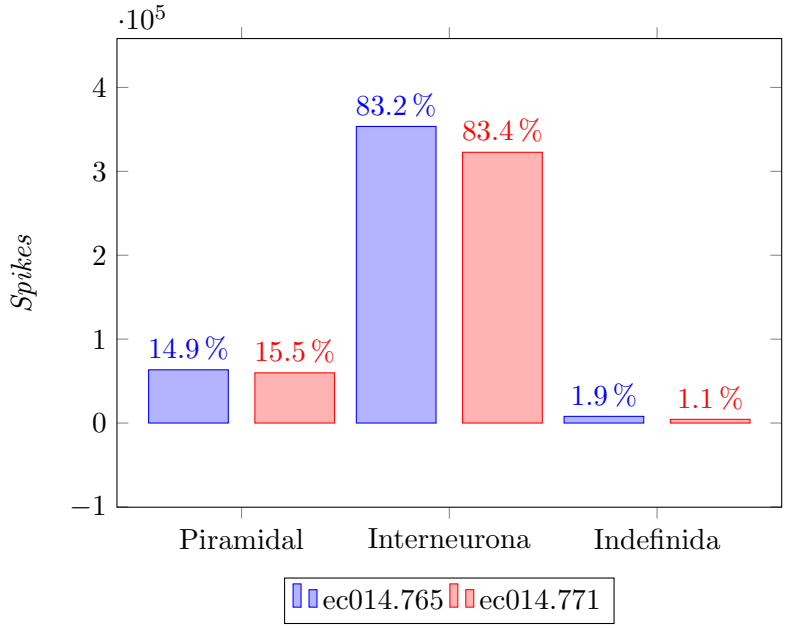


Figura 3.2: Gráfico de barras de cantidad de *Spikes* por tipo de neurona

Tipo de célula	Cantidad total de *Spikes*			
	ec014.765	(%)	ec014.771	(%)
Piramidal	63455	14.93	59825	15.46
Interneurona	353490	83.19	322688	83.40
Indefinida	7955	1.87	4411	1.14

Cuadro 3.5: Resumen de cantidad de *spikes* por tipo de célula en los bancos de datos.

Como se observa en el cuadro 3.5 y en la gráfica 3.2 el porcentaje de actividad de las células piramidales en comparación con las interneuronas es menor en el conjunto completo de datos que si se tomara en cuenta únicamente la región EC2 obtenidos con la sonda 9. De acuerdo con la teoría consultada, la información correspondiente a la orientación es procesada en el cerebro por las células de dirección de la cabeza. De manera que, bajo ese supuesto **se debe considerar primordialmente los electrodos de la sonda 9 para entrenar las redes con entradas** que realmente tengan relación estrecha con la característica a predecir, en este caso el ángulo de la dirección de la cabeza.

Sin embargo, como se observa en las figuras 3.3 y 3.4 las señales obtenidas con la sonda 9 (las terceras amarillas de arriba a abajo) no presentan gran

variación, y por lo tanto no contienen tanta información si se comparan con las de otros canales por ejemplo los de la sonda 6 o 11. Por lo que electrodos que serían buenos candidatos como la característica única de entrada a las redes podrían ser por ejemplo **los canales 40, 70 u 80**.

De forma que, inicialmente se puede considerar entrenar las redes neuronales con información de un solo canal. Aunque al notar que cada sonda contiene información ligeramente distinta de las otras, se podría considerar una solución que entregue como entrada a las redes las señales de LFP de múltiples canales con el fin de buscar una predicción lo más cercana posible a los valores reales de ángulo de la cabeza. Dicho subconjunto de datos debe de estar altamente relacionado a la característica a predecir o en su defecto utilizar la totalidad de los datos.

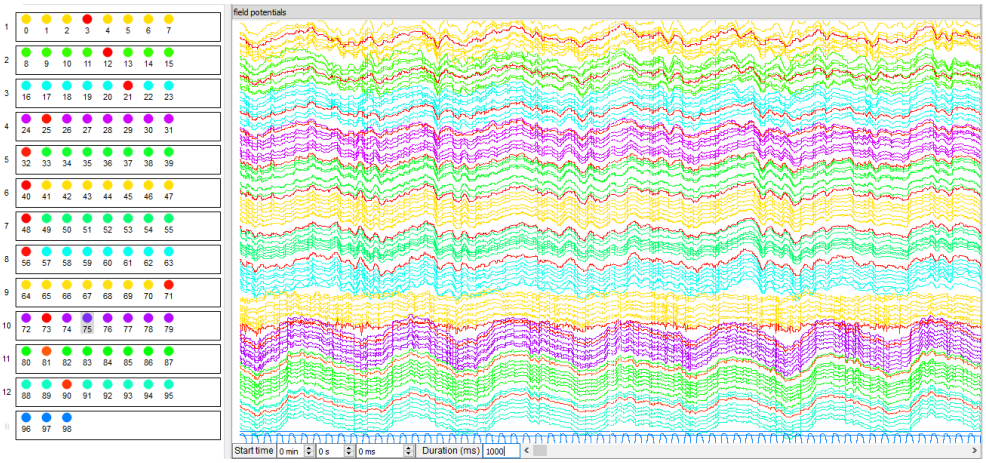


Figura 3.3: Primer segundo de señales de LFP por canal de la sesión 765 visualizadas con el software Neuroscope.

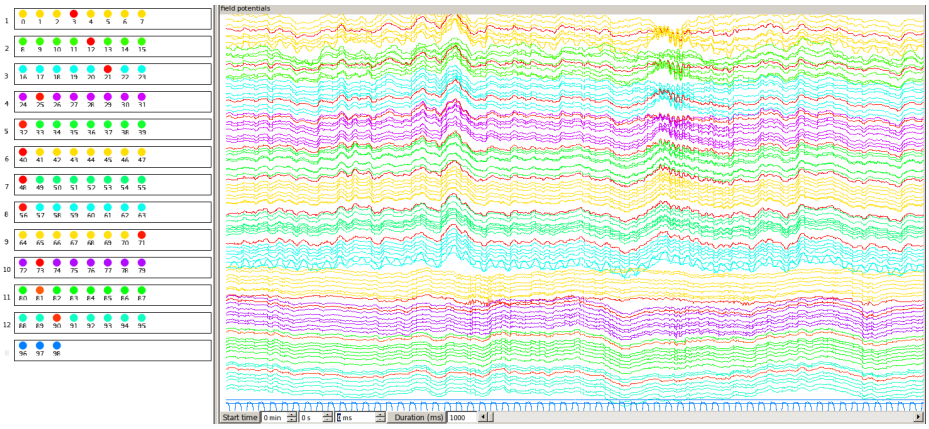


Figura 3.4: Primer segundo de señales de LFP por canal de la sesión 771 visualizadas con el software Neuroscope.

En las figuras 3.3 y 3.4 se muestra que los canales 96, 97 y 98 no contienen información relacionada a LFPs debido posiblemente a daños o desconexión de la sonda 13. Por esta razón, estos 3 canales deben excluirse del análisis para no aportar información errónea que pueda perjudicar la predicción.

3.2 Preparación de los datos y extracción de características

Una vez explorado el conjunto de datos a utilizar, estos se deben manipular y limpiar para darles un formato adecuado que sirva como entrada a las distintas redes neuronales. En esta sección se describirá detalladamente los pasos seguidos, así como los métodos creados para preparar las entradas de las redes. Uno de las consideraciones contempladas a la hora de implementar los pasos de procesamiento es que cada función fuera lo más general y flexible posible, con la intención de que siga siendo útil ante ligeras variaciones de los datos de entrada, y le proporcione al desarrollador distintas alternativas en la preparación de los *features* de los algoritmos de predicción.

Los pasos que se siguieron y detallaran a continuación son:

1. Lectura del archivo con señales de LFPs y organización de estas por canales e instantes de tiempo.

2. Lectura del archivo con los datos de posición de los LEDs, cálculo del ángulo al cual se dirige la cabeza para cada instante de tiempo.

3. Sincronización entre frecuencias de muestreo de los ángulos y los LFP para que coincidan las muestras uno a uno por diezmado o por rellenado de muestras.
4. Unión de muestras de LFPs con muestras de ángulos en una misma estructura de datos.
5. Limpieza de instantes de tiempo con ángulos inválidos.
6. Interpolación (en caso de rellenado de muestras de ángulos).
7. División de datos en subconjuntos para entrenamiento, validación y prueba.
8. Separación por ventanas de tiempo como entradas directas a las redes neuronales.

3.2.1 Preparación de las señales LFPs como entradas

Lectura de los archivos de señales de LFP

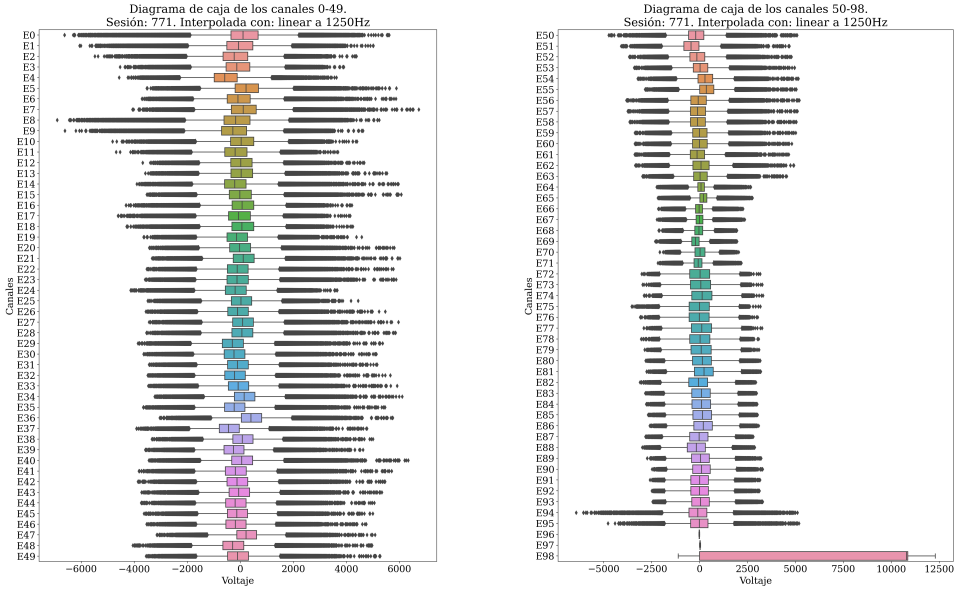
Para comenzar a preparar los datos de las grabaciones 765 y 771 lo primero que se debe hacer es leerlas del archivo .eeg y guardarlas en alguna estructura de datos. En este caso un arreglo de dos dimensiones, donde la primera corresponde a las muestras en el tiempo de la señal LFP en cada canal y la segunda al canal. De manera que se define la función *load_lfp_data(file, channels)* la cual toma la dirección del archivo a leer y la cantidad de canales que la grabación de ese archivo contiene y retorna el arreglo de 2 dimensiones descrito anteriormente.

Es necesario recordar que los archivos .eeg proporcionados por (Buzsaki, 2011) y descritos en descrito en la sección 2.2.3 contienen los datos de LFP en el mismo formato que los .dat. Ambos presentan las mediciones de cada canal como números enteros tipo *short* (2 bytes) consecutivos, sin ningún tipo de separador (datos crudos). El XML correspondiente a la sesión indica la cantidad exacta de canales que tiene cada grabación. No siempre será múltiplo de 8 debido a altas impedancias en los canales o sondas rotas. En el caso de ambas sesiones bajo estudio, se tienen 99 canales, o sea, 99 *shorts* por instante de tiempo.

Exploración de los datos de LFP

En la figura 3.5 se muestra el diagrama de cajas de todos los canales presentes en la sesión 771. Los puntos negros fuera de los rectángulos de colores repre-

sentan los *outliers* o datos que se salen del 1.5 de la distancia entre cuartiles después del tercer cuartil. La longitud de las líneas y cajas representan la dispersión de las mediciones en voltaje. De dicha figura se observa que la mayoría de canales tienen cajas relativamente pequeñas, lo que indica que el 50 % de los datos de cada canal se encuentran concentrados en una estrecha banda de voltaje. Este comportamiento es esperable pues los archivos leídos contienen únicamente datos de LFPs, ya que los *spikes* fueron filtrados en la conversión de archivos .dat a .eeg. De modo que se esperan valores de voltaje siempre dentro de un rango bastante definido.
Se aprecia además, que los últimos 3 canales presentan unos diagramas de caja bastante distintos a los del resto de canales. Los electrodos 96 y 97 parecen tener mediciones de voltaje por mucho inferiores al resto y el 98 por el contrario valores de voltaje muy por encima al resto de canales.



(a) Canales 0-49. **(b)** Canales 50-98.

Figura 3.5: Diagrama de cajas de los datos de LFP de la sesión 771 a su frecuencia original del archivo .eeg.

Esta figura, junto con la figura 3.6 en donde se observan los canales 0, 1, 96, 97 y 98 refuerza la hipótesis de que estos últimos 3 canales de las grabaciones analizadas, presentan información errónea dado a desperfectos en el equipo o en la colocación del mismo. En la figura 3.6 es notorio que los canales 96 y 97 tienen valores cercanos a cero, mientras que el 98 tiene una respuesta cuadrada de mucha mayor amplitud a la de los canales 0 y 1.

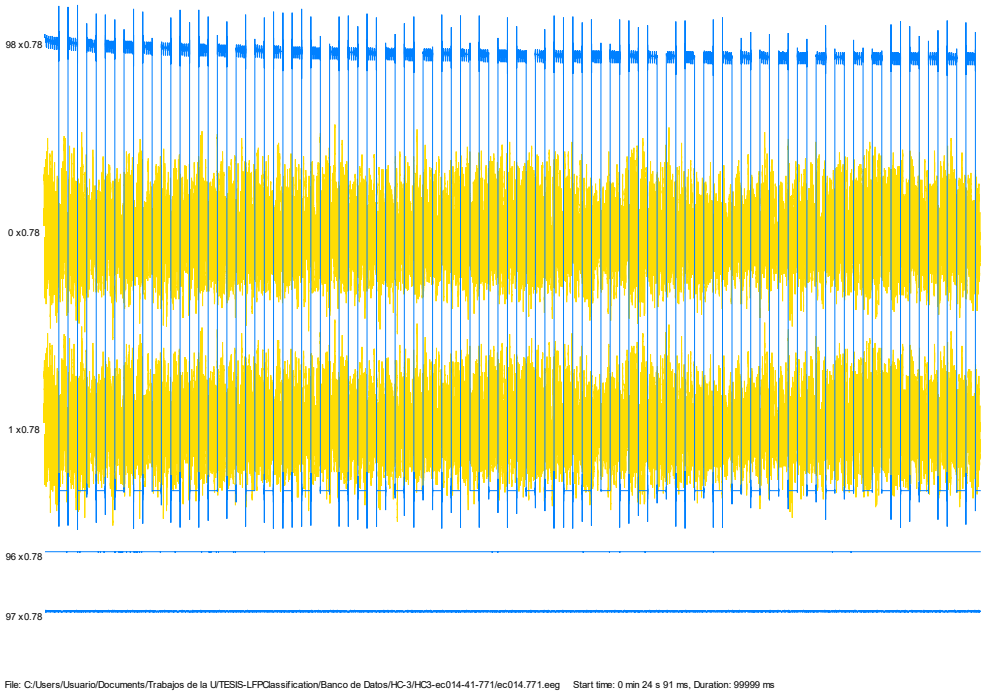


Figura 3.6: Primeros 100 segundos de los datos de los canales 0, 1 (en amarillo), 96, 97 y 98 (en azul).

Definición de la escala de los datos de entrada

Por otro lado, de la figura 3.5 surge la duda si es necesario aplicar algún tipo de normalización sobre estas señales, las cuales se utilizarían como las entradas a los algoritmos de clasificación. Se considera, sin embargo, que se deben mantener sus valores originales por las siguientes razones:

1. Los LFPs se definieron como la actividad conjunta de un grupo de neuronas. No solo es de interés la dinámica de la señal, sino también su media. Una media alta implica más actividad neuronal, que si se trata de células de dirección de la cabeza indica que el sujeto está viendo hacia el ángulo de preferencia de dichas neuronas. La normalización las señales de LFPs sería contraproducente ya que intencionalmente se estaría substrayendo información de interés para el entrenamiento de los algoritmos.

2. Todos los datos, aun si se utilizan múltiples canales, son de la misma naturaleza. Todas son señales voltaje de potencial de campo de local del mismo sujeto, utilizando el mismo sistema de adquisición. Por lo que la escala de todos los datos es la misma.

Análisis de señales LFPs en el tiempo

En la sección 2.1.6 se mencionó que las señales de LFPs se encuentran en el rango de frecuencias menores a $\sim 250Hz$. En la sección 2.2.3 se detalla que los archivos .eeg (los cuales son los que se utilizan en esta investigación) contienen únicamente los LFPs (frecuencias por debajo de los 625Hz), filtrando los *spikes*. De manera que al utilizar estos archivos, se estaría analizando información previamente filtrada en frecuencia para recuperar únicamente información de potenciales de campo local que, como se discutió anteriormente, tienen alta relación con la actividad neuronal principalmente en la banda Gamma. Esta es la principal característica en materia de la predicción del ángulo ya que las células de dirección de la cabeza presentan mayor actividad, incrementando así la amplitud de los LFPs cuando el sujeto se dirige al ángulo de preferencia de dichas neuronas y que aporta al sentido de orientación del sujeto (Taube, 2009).

Con el fin de utilizar toda la información contenida en las bandas de LFPs, se decide no añadir una etapa de filtrado por frecuencias adicional y analizar la amplitud de los datos proporcionados por (Mizuseki et al., 2014) en el dominio del tiempo. Además, el análisis en el dominio del tiempo permite interpretar los datos como una serie de tiempo y utilizarla como entrada para una red neuronal recurrente que aprovecha la relación entre muestras en el transcurso del tiempo. Esta característica es de interés porque para realizar la predicción de la próxima dirección de la cabeza es conveniente revisar no solo la posición actual, sino también algunas muestras previas que en conjunto permitan definir una tendencia, aclarar la dirección movimiento y finalmente predecir el siguiente ángulo.

3.2.2 Obtención del ángulo de dirección de la cabeza

De forma análoga a los LFPs, para obtener el ángulo real al que se dirigía la cabeza de la rata en cada instante de tiempo durante el experimento se deben leer los datos de la posición de los LEDs que el animal portaba del archivo .whl descrito en la sección 2.2.3.

Para esto se define la función *load_angles_data(file, degrees)* que extraerá el par ordenado *(x, y)* de los dos LEDs, y los guardará como números flotantes de 16 bits. La función espera que el archivo a leer tenga los cuatro valores correspondientes a un instante de tiempo en una sola fila, separados por una tabulación o el comando de control ASCII $\backslash t$. De forma que cada fila corresponde la adquisición de la posición de los LEDs en un momento determinado y las columnas serán x_1, y_1, x_2, y_2 en ese orden.

Para calcular el ángulo con base en las posiciones de los LEDs se utiliza la siguiente fórmula trigonométrica:

$$\angle HD = \arctan\left(\frac{y_2 - y_1}{x_2 - x_1}\right) \tag{3.1}$$

Donde $\angle HD$ en la ecuación 3.1 corresponde al ángulo al que se dirige la cabeza. Sin embargo, los archivos de posiciones podrían contener valores de '-1' que indican que no fue posible recuperar apropiadamente la posición del LED. Por lo que estos valores se sustituyen con 'NaN' o "*Not a Number*" en inglés que permite a la función *arctan2* de la librería Numpy de Python devolver también NaN en caso de que alguno de sus parámetros sea NaN o el resultado de la operación en radianes si todos los parámetros son válidos.

Una vez obtenido el ángulo en radianes, se puede decidir mediante el parámetro '*degrees*' si se quiere retornar el arreglo de ángulos en radianes o en grados. Si se retorna en grados, estos podrán tomar valores en rango [0° y 360°[. Finalmente todos los elementos NaN del arreglo de ángulos, se vuelven a definir como '-1' ya que en la etapa siguiente se podrían agregar intencionalmente valores NaN los cuales se deben distinguir de estos datos erróneos.

Para efectos de esta investigación, se decide analizar los ángulos en grados ya que esta unidad permite comprender de una manera más intuitiva los datos, por ejemplo al representarlos en una gráfica. Mientras que, con respecto al entrenamiento de las redes, resulta indiferente la unidad que se elija pues ambas son no dimensionados, como se mencionaba en la sección 2.4.7

3.2.3 Definición de frecuencia de trabajo

Para entrenar las redes neuronales, al ser algoritmos de aprendizaje supervisado se necesita que cada conjunto de "features" de entrada tenga una etiqueta con el valor real esperado. De manera que al realizar la predicción la red pueda comparar su predicción contra el valor real, calcular el error y propagarlo a través de todos sus pesos con el fin de ajustarlos para obtener un menor error en la próxima iteración.

Esta necesidad puede ser abordada de diferentes maneras, dependiendo de lo que se tome como las entradas de la red. Por ejemplo, se podrían definir ventanas de tiempo donde se utilicen una determinada cantidad de muestras de LFP en el tiempo, para predecir un único ángulo. De forma que se podría utilizar un conjunto de $[x_i, x_{i+WS}]$ muestras de LFP para predecir el ángulo y_{i+WS}. Donde x_i es una muestra de LFP, WS es el tamaño de la ventana o

la cantidad de muestras a considerar para la predicción, y y_{i+WS} el ángulo a predecir, el cual coincide con la última muestra de entrada.

Dado que las frecuencias de adquisición de las señales de LFPs no coinciden con la frecuencia de adquisición de las posiciones de los LEDs se debe definir apropiadamente las entradas y etiquetas que se alimentarán a las redes neuronales. Como la frecuencia de los LFPs es mayor a la de las posiciones, se puede seguir la estrategia descrita en el párrafo anterior. Sin embargo, se estaría limitando a usar ventanas múltiplos de la relación de las frecuencias (en este caso, múltiplos de ~ 32). Esta condición restringe la libertad de definir las entradas y etiquetas de la red, ya que podría ser deseable predecir un ángulo a partir de 100ms de datos de LFPs. Además que no todas las muestras de LFP tendrán un ángulo relacionado, solamente cada 32 de ellas.
Es por esta razón que se decide buscar una manera de crear un conjunto de datos en donde los datos de LFPs coincidan uno a uno con los ángulos, con el fin de que dicho conjunto proporcione flexibilidad de definición de las ventanas de predicción.

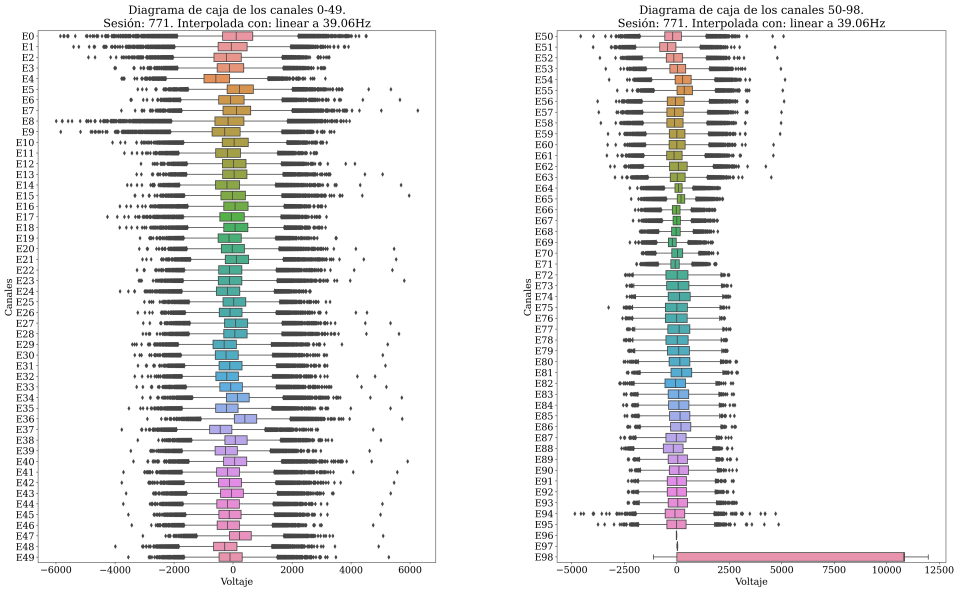
Diezmado de las señales de LFP: "*Downsample*"

Una forma de lograr empatar las muestras de LFP con los ángulos es diezmar los LFPs hasta la frecuencia de los ángulos. Para esto, en la función *downsample_lfps()* se utilizó *decimate* de la librería *scipy.singals* la cual aplica un filtro *anti-aliasing* Chebyshev de orden 8 por defecto. El propósito de diezmar utilizando este filtro en lugar de solamente botar muestras de la señal, es conservar el espectro en frecuencia de la señal lo más parecido al original, de manera que no se pierda o distorsione considerablemente la información de la señal de LFP.

Como el factor de diezmado q es de 32, o sea, de cada 32 muestras de la señal original se producirá solamente una se recomienda utilizar la función de *decimate* dos veces como lo recomiendan los autores para q's mayores a 13. (SciPy, 2021). De forma tal que, la cantidad de muestras originales se diezma por un factor de 8 y luego ese resultado por un factor de 4: $(n/8)/4$.

En la figura 3.7 se encuentran los diagramas de caja de la sesión 771, similar a los de la figura 3.5 con la diferencia de en este caso se eliminaron $P-1=31$ muestras de LFPs por canal por cada dato de posición para que estas calzaran uno a uno con las muestras de los ángulos a su frecuencia original de 39.06Hz mediante le método de diezmado de la sección 3.2.3. Donde P es la razón entre frecuencias calculada en la ecuación 3.2. Al comparar ambas figuras se observa que la distribución de las muestras se mantiene similar, con la diferencia de que la figura 3.7 contiene menos datos y que pasó por un proceso de diezmado. A

pesar de esto, la distribución de las muestras diezmadas de LFPs se mantuvo similar a la señal original.



(a) Canales 0-49. **(b)** Canales 50-98.

Figura 3.7: Diagrama de cajas de los datos de LFP de la sesión 771 a 39.06Hz, frecuencia de adquisición de la posición de los LEDs en el sujeto.

Relleno de ángulos faltantes: "*Padding*"

La otra alternativa es rellenar el arreglo de los ángulos para lograr una relación uno a uno con las muestras de LFPs y posteriormente interpolarlos de manera que los datos agregados se ajusten de manera coherente a los ángulos obtenidos directamente del experimento.
Se implementa entonces la función *angles_expansion(angles_data, orig_rate, new_rate)*. Esta calcula la razón entre la frecuencia nueva y la original, y lo redondea para obtener un número entero de ángulos a agregar, como se muestra en la ecuación 3.2.

$$P = \left\lfloor \frac{f_{nueva}}{f_{original}} \right\rfloor = \left\lfloor \frac{1250Hz}{39{,}06Hz} \right\rfloor = 32 \tag{3.2}$$

Una vez calculado el "factor de expansión" P, se procede a agregar $P-1$ valores NaN después de cada dato del arreglo de ángulos. Precisamente por este paso es que anteriormente en la sección 3.2.2 se marcaron los ángulos

inválidos con -1, para no ser confundidos con estos datos de relleno.

3.2.4 Adición de Etiquetas

Una vez igualadas la cantidad de muestras de LFP con la cantidad de ángulos, ya sea por *downsampling* de LFPs o *padding* de ángulos se procede combinar ambos arreglos en uno bidimensional donde cada fila corresponde a un instante de tiempo. Por su parte, las columnas corresponden a los canales de adquisición de señales LFP a excepción de la última columna la cual contiene el ángulo al cual se dirigía la cabeza del animal en ese instante de tiempo.

Se define la función *add_labels(lfps, angles, round_labels, start, offset)*. Si el parámetro *round_labels* es falso, simplemente se concatenaran ambos arreglos de datos. Caso contrario, se propone redondear los ángulos a números enteros definidos por *start* y *offset*. Por ejemplo, si se utiliza $start = 10$ y $offset = 40$, los ángulos tomarán los valores más cercanos en el conjunto {10°, 50°, 90°, 130°, 170°, 210°, 250°, 290°, 330°}. La resolución del *offset* es 1°.
La idea de proporcionar esta opción es que los ángulos calculados son valores flotantes de muchos decimales de resolución, y por ende puede resultar casi imposible que la predicción de las redes neuronales coincida sin error al valor real. Al redondear los ángulos, aunque sea a números enteros se reduce la cantidad de posibles valores que se podrían predecir de prácticamente infinitos a solamente 360.

3.2.5 Limpieza del banco de datos

En este paso se procede a eliminar los datos inválidos del conjunto de datos. Era conveniente tener tanto los datos de LFPs como de ángulos en el mismo arreglo bidimensional pues entonces se puede eliminar filas completas al identificar que el ángulo de dicha fila es inválido.

Con este propósito se implementa la función *clean_invalid_positional(labeled_dataset, is_padded)*. Primeramente, se identifican todos los índices en donde el valor del ángulo sea -1. Si los datos recibidos tienen ángulos expandidos, también se consideran como inválidos los NaNs añadidos inmediatamente después de cada -1. Por otro lado, si se trabaja con datos en donde se decimó las señales de LFP, se toman como inválidos únicamente las filas con ángulo igual a -1.
Seguidamente, se busca dónde comienza y termina cada 'discontinuidad' o conjunto de datos inválidos recorriendo el arreglo de índices inválidos. Si se cumple la condición de la ecuación 3.3 quiere decir que en el índice i del arre-

glo de índices inválidos A[] está comenzando una nueva discontinuidad y en A[i-1] está terminando la discontinuidad anterior.

$$A[i] - 1 \neq A[i-1] \tag{3.3}$$

Ahora, esta función no devuelve un solo arreglo sin discontinuidades de adquisición de posiciones. Si existen n discontinuidades en medio de ángulos válidos, devolverá n+1 arreglos con los subconjuntos de ángulos continuos y válidos. La razón detrás es mantener la continuidad de las mediciones, ya que si solamente se eliminan las filas con -1 como ángulo, se estaría cortando pedazos del experimento y uniendo los pedazos restantes como si fueran continuos, produciendo predicciones basadas en secuencias falsas de datos. Al dividir el conjunto original de datos en subconjuntos de datos válidos y continuos, se logra mantener les secuencias reales del experimento eliminando efectivamente los instantes de tiempo donde hubo problemas en la adquisición de la posición de los LEDs.

Cuando ya se tienen los índices donde comienzan y donde terminan las discontinuidades, se definirán como subconjuntos válidos los que estén bajo alguna de los siguientes casos:

1. Si el índice donde comienza la primer discontinuidad no es cero, quiere decir que la grabación de la sesión comienza con datos válidos. Por lo tanto se tomara como primer subconjunto válido desde el comienzo de los datos de la sesión hasta el primer índice de comienzo de discontinuidades.

2. Si el índice donde termina la última discontinuidad no es el último índice del arreglo original de datos, quiere decir que la grabación de la sesión termina con datos válidos. Por lo tanto se tomara como último subconjunto válido desde un elemento después de donde termina la última discontinuidad hasta el final de los datos de la sesión.

3. El resto de subconjuntos válidos se tomaran desde el índice donde termina la discontinuidad x hasta un elemento antes del índice donde comienza la discontinuidad $x + 1$.

Finalmente, no son deseables subconjuntos que queden con muy pocas muestras, por lo que se eliminarán aquellos que tengan menos de 32. Esto debido a que para el caso en que se agregaron ángulos de más, en la etapa de interpolación se necesitará al menos 2 ángulos reales para interpolar en medio de ellos, si se tienen menos de 32 muestras entonces sería solamente 1 ángulo real con 31 NaNs.

La creación de estos subconjuntos no disminuye el tamaño total del banco de datos limpios a utilizar, solamente lo divide. De igual forma, como se detallará posteriormente, se crearán ventanas una cantidad determinada de muestras para la predicción de los ángulos. Así que, se podrían tomar ventanas de todos los subconjuntos de datos entregados en esta etapa, manteniendo la continuidad de las ventanas y asegurando datos limpios. De todas formas, a la hora de entrenar las redes neuronales es recomendado barajar las ventanas creadas para mejorar la probabilidad de que el algoritmo obtenga ejemplos de todo el conjunto de datos destinado a entrenamiento y en cualquier orden, reforzando la independencia entre ventanas.

3.2.6 Interpolaciones

Dependiendo del método utilizado para obtener un conjunto de datos de LFPs del mismo tamaño que el de ángulos descritos en la sección 3.2.3, puede ser necesaria una etapa de interpolación. En la sección anteriormente mencionada se describió el proceso de "*downsampling*" el cual reducía la cantidad de muestras de una señal en un factor de diezmado q mediante un filtro *anti-aliasing* que mantuviera el espectro en frecuencia de la señal diezmada lo más similar a la original. El proceso opuesto es conocido como *upsampling* y consiste en 2 pasos: La expansión de los datos con ceros o *padding*, y la interpolación de los datos para cambiar los ceros agregados por valores que continúen la tendencia de los datos originales (Oppenheim y Schafer, 2009).
Si se decide rellenar los ángulos en lugar de reducir las señales de LFP entonces se debe además interpolar apropiadamente para que se mantenga la tendencia de los datos originales. Así que la función *interpolate_angles(angles_data, method)* permite aplicar cualquier interpolación que la función *interpolate* de la clase Series de la librería Pandas. Entre las opciones de métodos de interpolación están: lineal, cuadrática, cúbica, polinomial entre varias otras.

Rotaciones con cruce por 0°

Sin embargo, todas estas interpolaciones anteriores presentan un problema cuando el animal rote más allá de los 360° o a menos de 0°. Esto debido a que, por ejemplo en una interpolación lineal como la de la figura 3.8 si se desea interpolar un solo valor a la mitad de 350° y 30° el resultado sería 190°. Mientras que de acuerdo a la frecuencia de muestreo de las posiciones de los LEDs y la velocidad a la que una rata es capaz de girar su cabeza o su cuerpo completo, es más probable que la interpolación deba resultar en 20°, que correspondería a la mitad del recorrido en sentido anti-horario en un círculo.

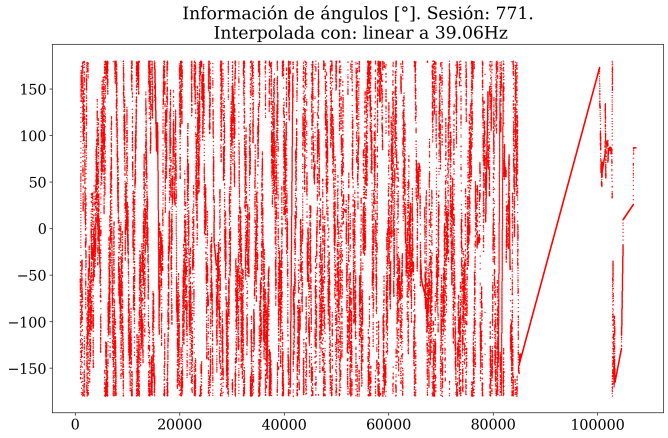


Figura 3.8: Ángulos de la sesión 771 a 39.06Hz.

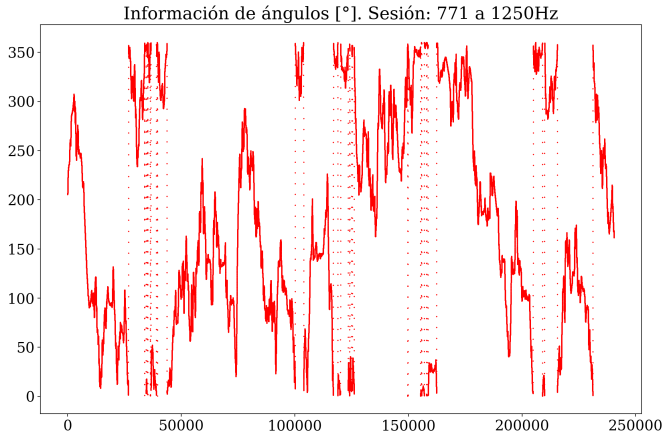


Figura 3.9: Interpolación Lineal de Ángulos de la sesión 771 con rotaciones con cruce por 0°.

En la figura 3.9 la cual corresponde a pequeña sección de la grabación 771 se observan más claramente algunos cruces por 0°. También se aprecia cómo la interpolación lineal coloca muestras a través del arco más grande entre los ángulos, dibujando así las lineas punteadas verticales que, conociendo el comportamiento natural del sujeto, es muy poco probable que realmente esta interpolación refleje el movimiento real grabado.

Para evitar este inconveniente, se podrían dividir el conjunto de datos en ventanas de tiempo que funcionarían como entrada a los algoritmos de entre-

namiento e identificar aquellas que contengan cruces por 0° y eliminarlos del conjunto de datos.

No obstante, estas rotaciones son movimientos naturales y por lo tanto deben de ser esperadas y analizadas adecuadamente por los algoritmos de predicción a implementar. Por esta razón se procede a evaluar un algoritmo distinto que permita interpolar ángulos buscando la menor distancia entre ellos. Contemplando la posibilidad de que la menor distancia de la interpolación sea cruzando por 0°.

Interpolación en el ángulo más corto

Con el fin de interpolar los ángulos de la dirección de la cabeza de los sujetos de forma que se ajuste a la naturaleza del movimiento se consideran las siguientes fórmulas para encontrar el ángulo más corto entre dos ángulos de una circunferencia.

$$S = ((x_1 - x_0 + 180)\ mod\ 360) - 180 \tag{3.4}$$

$$x_2 = (x_0 + S * t)\ mod\ 360 \tag{3.5}$$

La ecuación 3.4 encuentra el ángulo más corto S entre un ángulo de comienzo x_0 y un ángulo final x_1. Mientras que la ecuación 3.5 encuentra el ángulo interpolado x_2 sumándole el ángulo más corto multiplicado por un factor de posición t al ángulo de comienzo. Ambas ecuaciones utilizan la operación de módulo definida por la ecuación (Python, 2017):

$$r = a\,\%\,n = a\,mod\,n = a - n\left\lfloor \frac{a}{n} \right\rfloor \tag{3.6}$$

Donde r es el residuo de a dividido entre n. Nótese que en la ecuación 3.6 producirá un residuo que siempre será del mismo signo al divisor. Por ejemplo, la ecuación 3.7 detalla el cálculo de $-10\,\%3 = 2$:

$$r = -10 - 3\left\lfloor \frac{-10}{3} \right\rfloor = -10 - 3 * \lfloor -3.\overline{3} \rfloor = -10 - 3 * (-4) = -10 + 12 = 2 \tag{3.7}$$

Es necesario aplicar la operación de modulo en ambas ecuaciones 3.4 y 3.5 para asegurar que el ángulo resultante esté dentro del rango [0°, 360°[.

Una vez implementada la función *shortest_angle_interpolation(start, end, amount)* para interpolar entre dos ángulos aplicando las ecuaciones 3.4 y 3.5, se procede a aplicarla en el conjunto de datos completo para rellenar apropiadamente los valores agregados en el paso de *padding* de los ángulos originales.

Asimismo, la función *vectorized_sai(angles_data)* recorre el conjunto de datos buscando datos de ángulos válidos para usar como el ángulo de comienzo. Seguidamente se cuentan los elementos NaN hasta encontrar el siguiente valor de ángulo válido, el cual sería el ángulo final de una interpolación y de comienzo para la siguiente. Conocidos el ángulo de comienzo y final, así como la cantidad de valores a interpolar entre ellos se procede a calcular el factor t para cada dato de relleno:

$$t = \frac{i}{totaldeNaNs + 1} \tag{3.8}$$

Donde i de la ecuación 3.8 es la muestra de relleno i-ésima después del ángulo de comienzo, comenzando en uno. Este se divide entre la cantidad total de muestras a interpolar entre los dos ángulos reales encontrados, más uno. En el caso de que solo haya un dato por interpolar entonces t sería igual a $\frac{1}{1+1} = 0{,}5$.

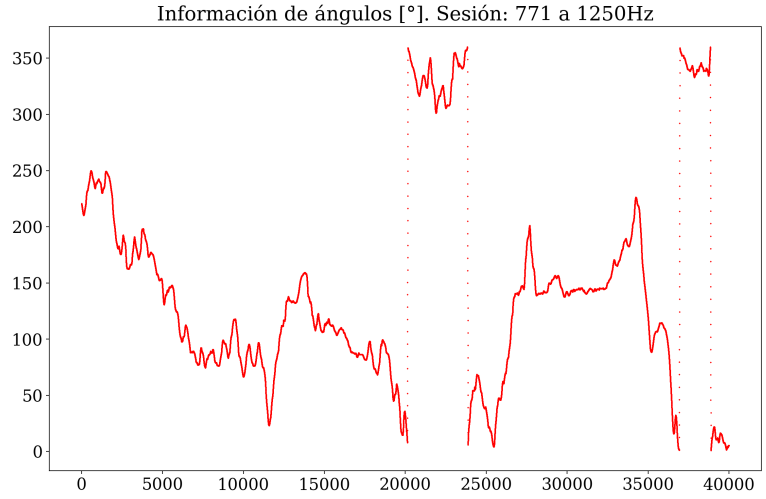


Figura 3.10: Interpolación Lineal de Ángulos de la sesión 771.

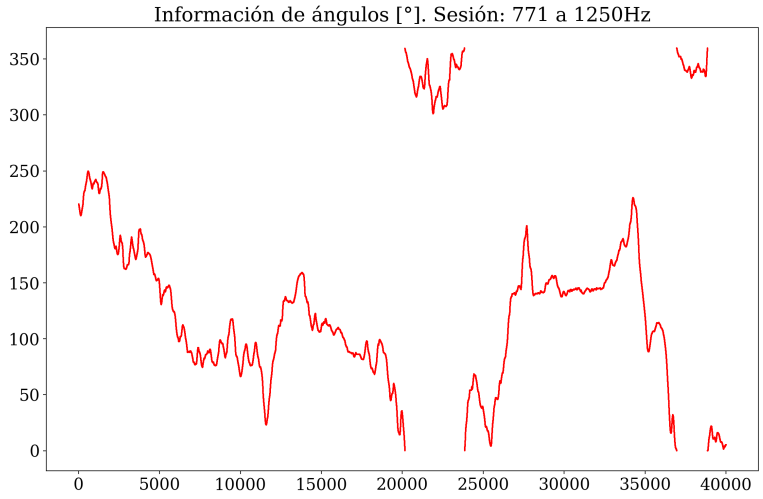


Figura 3.11: Interpolación por ángulo más corto de Ángulos de la sesión 771.

Se puede comparar la figura 3.11 de una interpolación que utiliza el arco más corto entre dos ángulos contra la figura 3.10 que utilizó una interpolación lineal. Ambas interpolaciones producen resultados prácticamente iguales en la mayoría de los datos de ángulos. La gran diferencia reside en que la interpolación por el arco más corto interpola mejor los datos de los ángulos que cruzan por 0°, eliminado las lineas verticales de datos interpolados erróneos.

La implementación de la función que aplica la interpolación del ángulo más corto en un conjunto de datos podría ser optimizada o al menos restringida a interpolaciones de 31 datos entre muestras válidas dada la ecuación 3.2. Sin embargo, se define como se describió interiormente para generalizar ante cualquier razón entre frecuencias, o sea, que sea capaz de interpolar cualquier cantidad de datos entre ángulos válidos, aún si dicha cantidad es variable.

Finalmente, el método de interpolación resulta prácticamente indiferente debido a que la frecuencia de muestreo de las posiciones de los LEDs con la que se calculan los ángulos es mucho más veloz que el movimiento del animal. Revisando los números, la frecuencia de muestro de las posiciones de los LEDs es de $f_{LEDS} = 39{,}06Hz$, lo que quiere decir que el tiempo entre muestras es de 25.6ms. Esta velocidad es suficiente para considerar que el cambio entre un ángulo y el siguiente inmediato es despreciable. Más aun contemplando que se interpolan 31 muestras entre cada dos ángulos reales.
No obstante, para asegurar una interpolación adecuada sin importar las frecuencias en cuestión, se opta por utilizar la que considera el ángulo más corto,

que como se mencionó anteriormente se aproxima más al comportamiento natural del movimiento del animal.

3.2.7 División de los datos

Ya que en el paso de limpieza de los datos descrita en la sección 3.2.5 los datos de la sesión fueron separados en subconjuntos de datos ininterrumpidos en el tiempo, se tomó únicamente el subconjunto más grande, o sea, el de mayor duración para entrenar las redes neuronales. Este se divide entonces en 3 partes de tamaños distintos que se utilizarán como se acostumbra en general en aprendizaje automático según (Vasilev et al., 2019):

- **Entrenamiento** (70 %): Como su nombre lo indica se utiliza para entrenar la redes neuronales.
- **Validación** (20 %): Se usa para evaluar la exactitud del algoritmo en un conjunto de datos desconocido. Este se le proporciona a la red una vez que fue entrenada por un determinado tiempo para medir su desempeño. Si el resultado no es satisfactorio, se pueden modificar los hiperparámetros del algoritmo y repetir el proceso. Al comparar el desempeño del algoritmo ante este conjunto y el de entrenamiento se puede identificar si hubo por ejemplo *overfitting*, fenómeno que se da cuando se tiene un muy buen desempeño en los datos de entrenamiento pero no así en los de validación.
- **Prueba** (10 %): Se utiliza solo una vez al final del proceso. Al igual que el de validación, este conjunto es desconocido para la red, ya que no fue utilizado en su entrenamiento. La diferencia reside en que al ajustar los hiperparámetros para lograr un mejor desempeño en la validación entre iteraciones de entrenamiento, se está sesgando el algoritmo para favorecer el resultado en el conjunto de validación y podría dejar de representar un desempeño confiable. Al medirlo además contra el conjunto de prueba, se intenta obtener un resultado lo más objetivo posible.

Se debe recordar que se platea el entrenamiento de algoritmos de aprendizaje supervisado, por lo que todos los conjuntos deben contener no solo las entradas a la red, sino además las etiquetas o valores esperados y reales de la predicción.

Los porcentajes de cada conjunto, aunque comunes, fueron escogidos siguiendo del ejemplo de la librería (Tensorflow, 2021) el cual trata sobre predicción de características de un conjunto de datos de clima. Este recomienda la división de los datos sin mezclarlos previamente por las siguientes dos razones:

1. Asegura que sigue siendo posible cortar los datos en ventanas tiempo de muestras consecutivas.

2. Asegura que los resultados de validación y prueba son más reales, considerando que estos dos conjuntos pudieron ser tomados posterior al entrenamiento de los modelos.

Ambos motivos son aplicables a la presente investigación pues también se trabaja sobre datos de series de tiempo.

3.2.8 Separación por ventanas de tiempo

En el análisis de series de tiempo se suele utilizar la muestra actual y varias muestras anteriores con el fin de aprovechar sus características de tendencia y estacionalidad. Una sola muestra de LFPs podría no ser suficiente para determinar el ángulo al cual se dirige el animal, pero si se utilizan varias muestras se proporciona un contexto. Adicionalmente, dado que la activación de las neuronas es mucho más veloz que el movimiento físico del sujeto, se tendrán múltiples datos de LFPs para el mismo ángulo a predecir.

Un camino es el seguido por (Islam et al., 2017) que promediaba la ventana para obtener un único valor y utilizarlo para la clasificación. Sin embargo, al intentar aplicar estrategia en el presente problema no sería muy claro cuál ángulo debería predecirse; si el primero de la ventana, el último, cualquier otro posición, o de igual forma el promedio de los ángulos en la misma ventana de tiempo.
Este paso entonces, consiste en tomar n cantidad de datos consecutivos de LFP, de uno o múltiples canales, y utilizarlos como entrada a las redes neuronales para predecir el ángulo del instante de tiempo de la muestra de la ventana.

Aunque la preparación de las ventanas de tiempo podría hacerse de distintas formas, se decide utilizar una que aprovechara al máximo los datos.
Por ejemplo, tomar una primer ventana desde la muestra 0 hasta la muestra *n-1*, la segunda ventana desde el dato n hasta el *2n -1* y así sucesivamente siendo n la cantidad de muestras por ventana.
Otra alternativa, y la cual es la que se escoge, es que la primer ventana se mantenga igual, pero la segunda vaya desde el dato 1 hasta el dato n, la tercer desde la muestra 2 hasta la muestra *n+1* y así sucesivamente.
La diferencia es que la creación de ventanas no depende solamente del parámetro del tamaño de la ventana n sino también de un segundo parámetro que indique el desface entre una ventana y otra. En el primer caso se tendría un desface del mismo valor que el tamaño de la ventana, mientras que en el

segundo se utiliza un desface igual a 1. En ese segundo escenario se podrían proporcionar más ejemplos de entrenamiento a la red, pues todas las ventanas seguirían siendo diferentes y se utilizarían todos los ángulos disponibles como las etiquetas o valores esperados de las predicciones.

De forma que se implementa la función *channels_to_windows(series, channel, window_size, batch_size, shuffle_buffer, offset)*. Esta toma un conjunto de datos y un canal específico del cual se extraerán la ventanas, por lo que **se procede a entrenar la red con la información de un solo canal** y no de múltiples electrodos. Los datos de este canal se separan en ventanas de un tamaño y un desface también obtenidos como argumentos, siendo 1 el valor por defecto del desface. En determinado caso en que la combinación del tamaño total de conjunto de datos, el de la venta y el desface escogido provoquen que "sobren" muestras, ya que no serían suficientes para crear una última ventana, estas se desecharían.
Después se revuelve el orden de las ventanas y se agrupan estas ventanas revueltas en *batches* para agilizar el proceso de entrenamiento.
Se provee adicionalmente una función *average_angles(angles, window_size)* en caso de que en lugar de utilizarse el último ángulo de la ventana como valor a predecir, se desee utilizar el promedio de los ángulos en dicha ventana.

Finalmente, a la redes neuronales se les proporciona como entrada dos arreglos de la forma (cantidad de ventanas por batch, cantidad de muestras por ventana, cantidad de canales):

- El arreglo de entrada con los *features* que en este caso son las ventanas de tiempo de la señal de LFPs de un solo canal. Para una forma por ejemplo de (32, 1250, 1), se tendrían 32 ventanas por *batch*, 1250 muestras por ventana de un solo canal.
  Por lo que se tendrían ventanas de $1250 * \frac{1}{1250Hz} = 1s$.

- El arreglo con las etiquetas. En este caso los ángulos de referencia con los cuales compara la predicción. Para el ejemplo anterior, con esa forma de entradas se tendría una forma en las etiquetas de (32, 1, 1) ya que se predice 1 solo ángulo por ventana, con 32 ventanas por *batch*.

Esta composición de entradas es la que los modelos elaborados en la librería que se utilizará para crear y entrenar las redes neuronales reciben por defecto. De manera que al prepararlos de esta forma estándar, se asegura que las redes están siendo alimentadas de manera apropiada.

3.3 Implementación de las redes neuronales

Como el principal objetivo del presente trabajo es una investigación y no la creación de nuevas herramientas, se opta por utilizar recursos de software que faciliten el desarrollo del proyecto. Desde la preparación de los datos, diseño y entrenamiento de modelos de redes neuronales para la predicción; y la conducción de las pruebas de rendimiento y obtención de resultados. Por este motivo se recurre a las librerías de inteligencia artificial y aprendizaje automático de Google: TensorFlow y Keras para la creación de las NN, su entrenamiento y obtención de métricas de rendimiento.
Particularmente, se basa esta parte del proyecto en el ejemplo ya mencionado (Tensorflow, 2021) que predice factores climáticos a partir de series de tiempo de las características del clima en cuestión.
Entre las principales ventajas de la utilizar estas librerías es que asegura que los modelos utilizados han sido validados y depurados, de manera que permite al desarrollador concentrarse en el problema de la investigación como tal. Aunque facilita considerablemente la etapa de modelado y entrenamiento de las redes, es necesario asegurar que se utilizan los recursos API en beneficio de la solución del problema, teniendo el conocimiento del algoritmo que la librería abstrae.

3.3.1 Red Neuronal Clásica

El primero de los modelos que se implementa para obtener la predicción es una red neuronal convencional de capas prealimentadas de perceptrones como nodos densamente conectados. La estructura de capas de esta red es de la siguiente forma:

1. Primeramente se pasa por una capa que “aplana” los datos de la forma (muestras, canales) a (muestras*canales), reduciendo de una matriz a un vector de datos. Esto debido a que se utilizan ventanas de tiempo como entrada en lugar de un solo conjunto de muestras del mismo instante de tiempo.

2. Una capa de entrada que recibe el vector anterior.

3. Una capa opcional de regularización

4. Capas escondidas, cuya cantidad de capas depende del usuario.

5. Otra capa opcional de regularización

6. Capa de salida con una sola neurona con activación lineal que entregará el resultado final de la predicción del ángulo.

7. Una capa que restaura la forma original de las entradas.

Tanto la capa de entrada como las capas escondidas son densamente conectadas y con activación ReLU y contienen la misma cantidad de neuronas por capa.

La función definida *MLP(layers, units_per_layer, dropout)* permite definir cuántas capas escondidas tendrá la red y cuantas neuronas habrán en las capas de entrada y escondidas (recordando que la capa final tiene solo una neurona). Las capas opcionales de regularización son de tipo *dropout*.

Cabe resaltar que se debe utilizar como activación en la neurona de la última capa una función lineal en lugar de una no lineal como ReLU pues lo que se busca es el valor de un ángulo en el rango de 0 a 360 grados y no una clasificación binaria.

3.3.2 Red Neuronal Convolucional

La red neuronal convolucional propuesta es bastante sencilla. Simplemente contiene:

1. Una capa convolucional de una dimensión (Conv1D de la librería Keras) que recibe múltiples instantes de tiempo por cada *feature*, que en este caso se trata como *feature* el canal seleccionado. Esta tiene una activación ReLU.

2. Una capa densamente conectada con activación ReLU

3. Una capa opcional de regularización por *dropout* y

4. La capa de salida que tiene una única neurona con activación lineal.

En la función *CNN(inputs, units_per_layer)* la cantidad de salidas de la capa convolucional debe coincidir con la cantidad de neuronas en la capa del medio (densa), este número es proporcionado por el usuario como argumento de la función.
Además, se especifica el tamaño del *kernel* a aplicar en la convolución. Se toma como tamaño del kernel, el mismo tamaño de la ventana de LFPs que se utilizará para predecir el ángulo, pues esta capa debe producir un solo resultado por convolución, que será uno del total de resultados que tiene que entregar a la siguiente capa. En caso de no hacerlo así, la siguiente capa debería ser otra convolucional hasta producir un solo valor que se entregue a una capa densamente conectada, tal como se muestra en la figura 2.15 del capítulo 2.

3.3.3 Red Neuronal Recurrente: LSTM

La tercer red neuronal que se plantea utilizar para la predicción del ángulo es una LSTM. Al igual que las otras redes, para implementarla se utiliza el modelo ya preparado en la librería de Keras. Por lo que la función *LSTM(units_per_layer)* simplemente contiene el modelo de capas secuenciales con una capa de entrada LSTM con la cantidad de neuronas especificada por el usuario, y la capa final de una sola neurona con activación lineal para la salida de la predicción.
Las activaciones de la capa LSTM son las convencionales, descritas en el diagrama de la figura 2.17. Además, se le pasa el parámetro de regularización por *dropout* entre capas.

3.3.4 Entrenamiento de las redes

Se crea la función *compile_and_fit(model, train, val, epochs, patience)* que permite entrenar un determinado modelo de entre los 3 anteriores. Esta recibe además, el conjunto de datos de entrenamiento y de validación y la cantidad de épocas que se entrenará la red.
La red por defecto se entrenará usando el Error Cuadrático Medio (MSE) como función de costo y el Error Absoluto Medio (MAE) como métrica de evaluación del desempeño. Sin embargo estos podrían ser parametrizados, principalmente la función de costo, con el objetivo de probar si con diferentes funciones se acelera el proceso de entrenamiento o se mejora su desempeño. Una buena alternativa de función de costo para este problema que tiene entradas no dimensionadas es la función de error MAPE.
Por último, en caso de que no se detecte mejora considerable en el error del conjunto de validación entre una cantidad determinada de épocas se detendrá prematuramente el entrenamiento, a esto se le conoce como *early stopping*. La cantidad de épocas que comparará para revisar si hubo mejoría está dado por el último parámetro de esta función.

3.3.5 Ambiente de pruebas

El diseño de soluciones de inteligencia artificial suele ser un proceso iterativo de prueba y error. En este se van ajustando los distintos hiperparámetros de preparación de datos de entrada y de entrenamiento de las redes neuronales en búsqueda de un mejor desempeño de los algoritmos en la clasificación o predicción según se requiera. Si no se maneja con detalle, es fácil olvidar una combinación de parámetros que proporcionó un buen resultado y de la cual valdría la pena basarse para continuar ajustando. Por esta razón resulta muy conveniente contar con una infraestructura de software flexible que permita

parametrizar estos procesos y documentar los resultados para un análisis posterior.

De manera que, para facilitar la ejecución de las pruebas, se decide crear un ambiente que permita:

- Parametrizar la generación de los conjuntos de datos a utilizar, así como el entrenamiento de las redes neuronales.
- Documentar los pasos seguidos en la ejecución de cada prueba, como también los resultados relevantes de esta. Entre los resultados se podrían incluir gráficas, valores de rendimiento de las redes y demás.
- Guardar los conjuntos de datos de entrada generados con los parámetros especificados. Y por ende, capacidad de cargar y utilizar fácilmente dichos archivos.
- Documentar las versiones de las dependencias utilizadas y requeridas para la utilización de librería y *scripts* de pruebas.

Dicho ambiente, en conjunto con la librería de preprocesamiento y de creación y entrenamiento de redes neuronales descritas en este capítulo (3) conforman la infraestructura de software creada para el desarrollo de este proyecto y para la obtención de los resultados que se discutirá en el siguiente capítulo (4).

4 Análisis de Resultados

4.1 Preparación de datos de entrada

Siguiendo los mismos pasos descritos en la sección 3.2, se irán mostrando los resultados obtenidos en las diferentes etapas, discutiendo las variaciones más relevantes en cada una.

4.1.1 Lectura de los LFPs

Después de leer las señales de LFP del archivo, se grafican los canales 0 y 97 para las sesiones 771 y 765 (figuras 4.1 y 4.2 respectivamente):

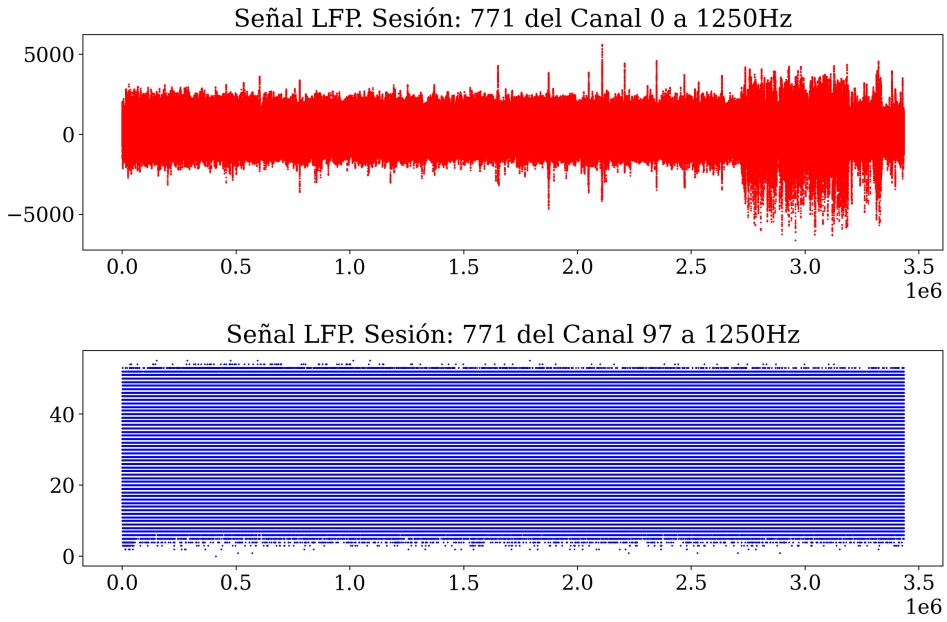


Figura 4.1: Gráfica de las señales de LFP de los canales 0 y 97 de la sesión 771 a su frecuencia original (1250Hz).

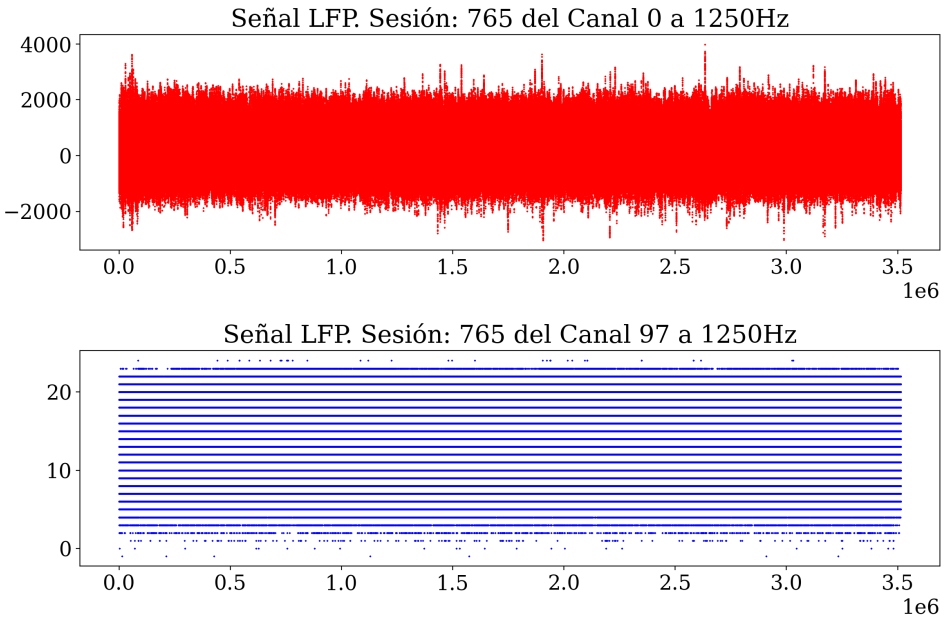


Figura 4.2: Gráfica de las señales de LFP de los canales 0 y 97 de la sesión 771 a su frecuencia original (1250Hz).

De las figuras 4.1 y 4.2 se confirma el daño de la última sonda usada en la grabación de los LFP al observar el canal 97. Como se reflejaba en la figura 3.5b los canales 96, 97 y 98 tienen datos inválidos que se excluyen del análisis.

Por otro lado, como la frecuencia de muestreo era muy alta (1250Hz) y los sesiones tardaban varios minutos, en las imágenes anteriores no se detalla apropiadamente la dinámica de los LFPs.
Para el caso de la sesión 771 la cual de acuerdo a la tabla 3.2 tiene una duración de $\sim 46min$ se obtuvo 3514368 muestras por cada uno de los 99 canales. Mientras que para la sesión 765 de $\sim 47min$ se cargaron 3433472 muestras por canal, poco menos que la sesión 765. Por esta razón se grafican en las figuras 4.3 y 4.4 los primeros 10 segundos de un canal de cada sesión para tener una visión más detallada de la variación de los LFPs en el tiempo:

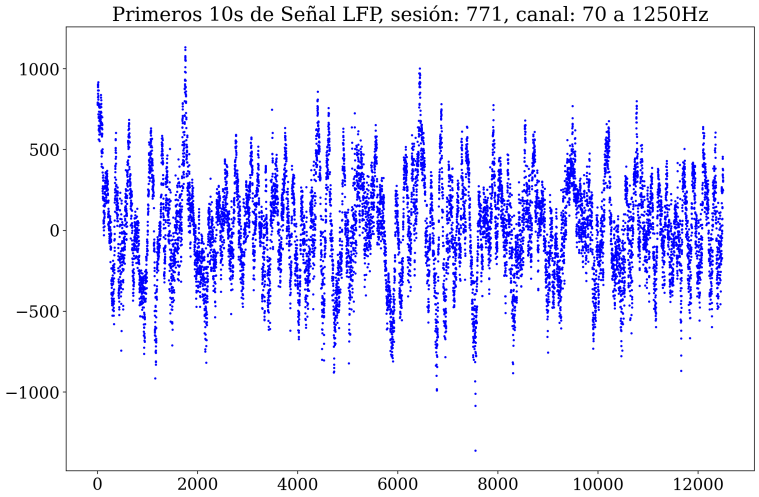


Figura 4.3: Gráfica de las señales de LFP a su frecuencia original (1250Hz) durante los primeros 10 segundos del canal 70 de la sesión 771.

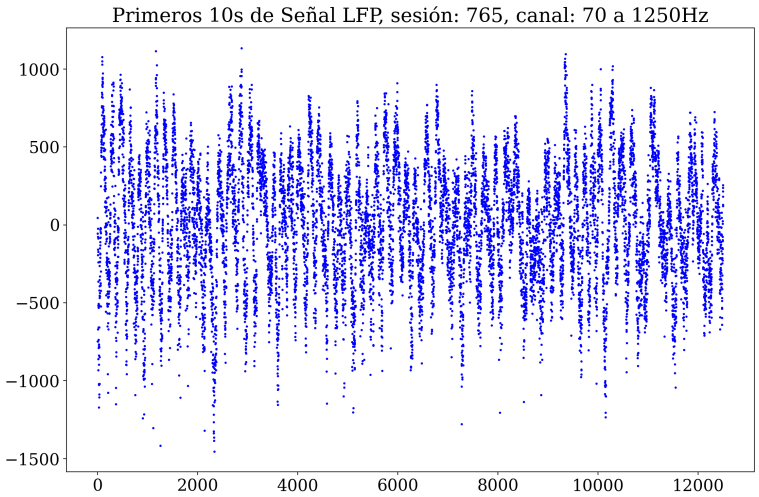
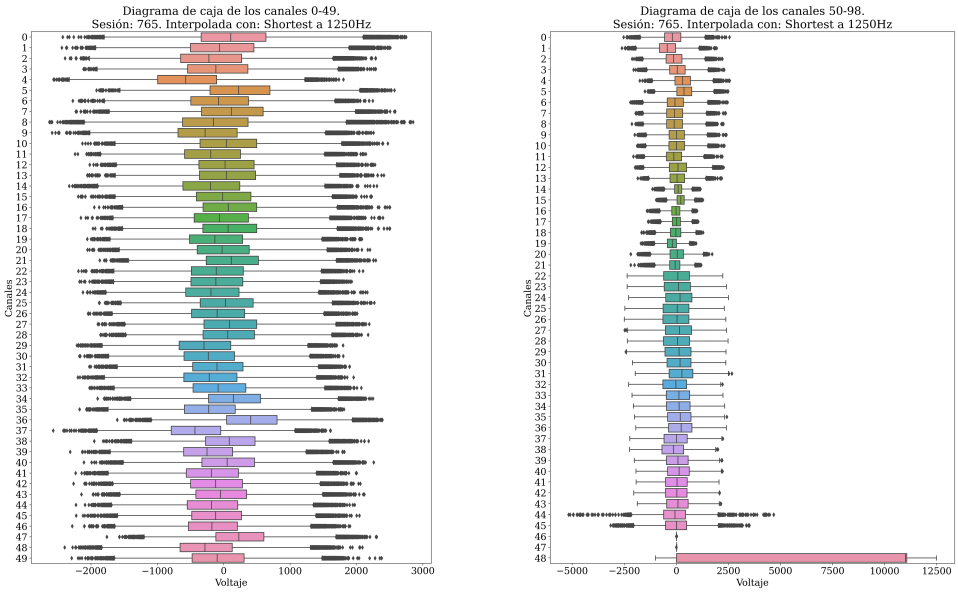


Figura 4.4: Gráfica de las señales de LFP a su frecuencia original (1250Hz) durante los primeros 10 segundos del canal 70 de la sesión 765.

De las figuras 4.4 y 4.3 se aprecia que encontrar un patrón de activación de las neuronas no es sencillo a simple vista, por lo que el uso de redes neuronales resulta apropiado, principalmente al contemplar que se tiene la disponibilidad de mucha cantidad de datos que se pueden utilizar para el entrenamiento de

estas.



(a) Canales 0-49. **(b)** Canales 50-98.

Figura 4.5: Diagrama de cajas de los datos de LFP de la sesión 765 a su frecuencia original del archivo .eeg.

Para complementar las figura 3.5 que mostraba el diagrama de cajas de la sesión 771, en la figura 4.5 se observa el correspondiente a la sesión 765, el cual presenta un comportamiento similar.
Ya que estos diagramas de cajas muestran la distribución de los distintos canales, se considera que no es relevante mostrar todos los canales ni con tanto detalle como el de las figuras 4.4 y 4.3 ya que son muy similares. Además en las figuras las figuras 3.3 y 3.4 se observa una pequeña muestra de 1 segundo de las señales LFP de cada canal de cada sesión.

4.1.2 Obtención de los ángulos

En cuanto al cálculo de los ángulos con base en la posición de los LEDs en la cabeza del animal:

- Para la sesión 765 se tienen 109825 muestras (figura 4.6) de las cuales 7671 (6.98 %) son inválidas.
- Mientras que en la sesión 771 habían 107297 (figura 4.7) con 25545 (23.81 %) de datos inválidos.

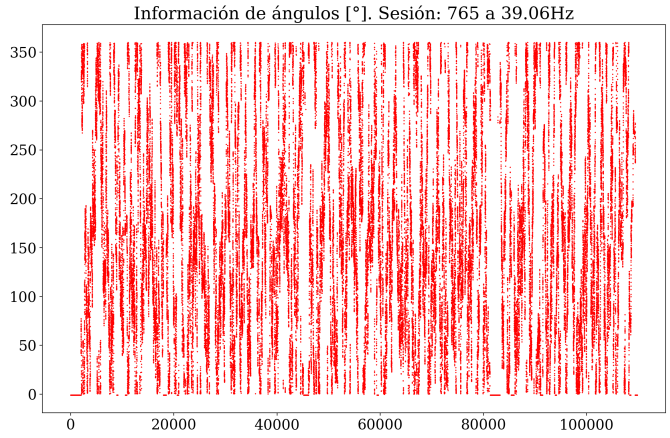


Figura 4.6: Gráfica de los ángulos de la sesión 765 a su frecuencia original (39.06Hz).

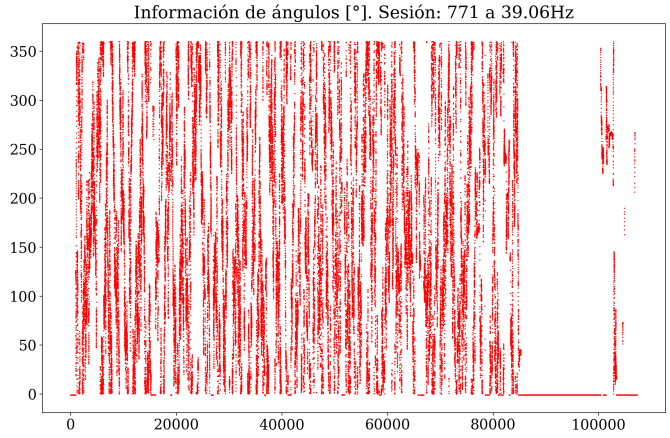


Figura 4.7: Gráfica de los ángulos de la sesión 771 a su frecuencia original (39.06Hz).

Los puntos y líneas ligeramente por debajo de 0 grados en las figuras 4.6 y 4.7 corresponden a los ángulos inválidos, a los que se les asignó el valor de "-1".
Por otro lado, es notable la diferencia porcentual de datos inválidos entre ambas sesiones pues la sesión 771 presenta un gran tramo de tiempo (poco después de las 80000 muestras) donde no fue posible obtener apropiadamente la posición de los LEDs y por lo tanto se tiene datos de ángulos inválidos.

Al igual que con los LFPs, se grafica una ventana de tiempo más pequeña para ambos conjuntos para apreciar mejor la variación de los ángulos en el tiempo. En las figuras 4.8 y 4.9 se muestran ventanas de 20s a partir del minuto 6 de ambas sesiones.

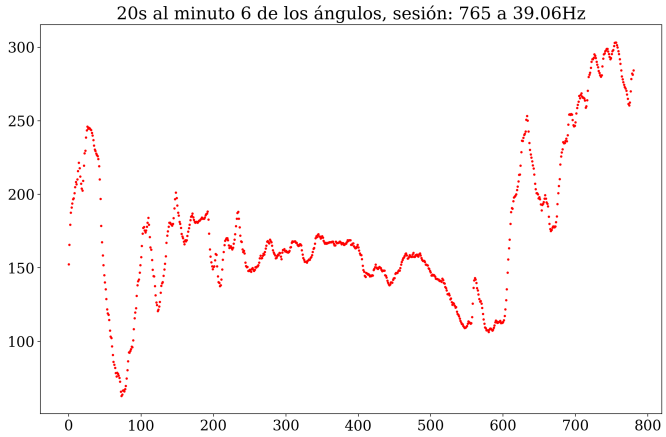


Figura 4.8: Gráfica de los ángulos a su frecuencia original (39.06Hz) en una ventana de 20 segundos al minuto 6 de la sesión 765.

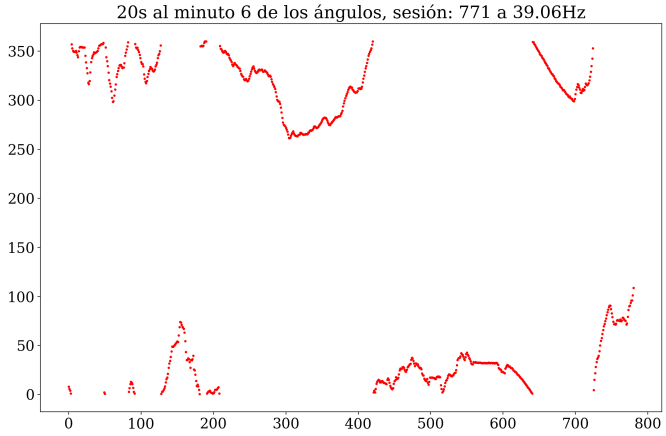


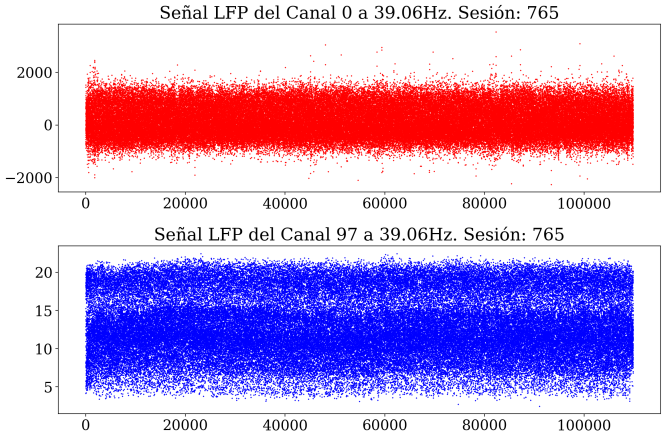
Figura 4.9: Gráfica de los ángulos a su frecuencia original (39.06Hz) en una ventana de 20 segundos al minuto 6 de la sesión 771.

En las imágenes anteriores se muestran dos ventanas de tiempo sin datos inválidos y por lo tanto continuas. La figura 4.8 presenta poco movimiento a excepción de un par de rotaciones de cabeza al inicio y al final del segmento mostrado, reflejado por los picos cerca de las 100 y 750 muestras.

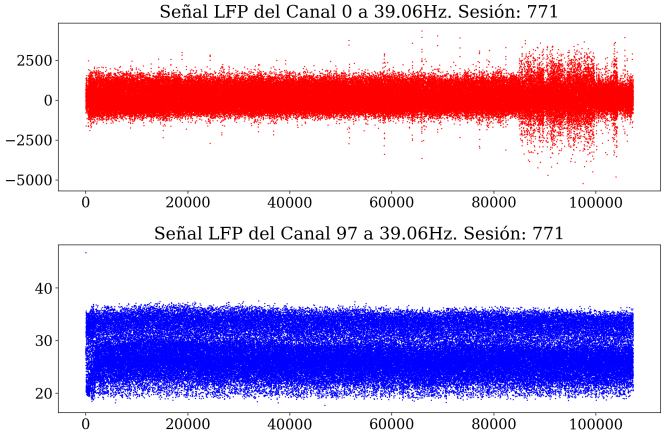
Por otro lado, la figura 4.9 ilustra muy claramente las rotaciones con cruce por cero grados que deben interpolar (de ser necesario) de manera que mantengan el sentido de movimiento del animal, en lugar de interpolar por el arco incorrecto de giro.

4.1.3 Diezmado de señales de LFP a 39.06Hz

Si el mecanismo de sincronización de las frecuencias que se escoge es el diezmando de LFPs, la información que se le suministrará a la red se ve decrementada drásticamente Como se aprecia la figura 4.10 contiene muchos menos puntos que las figuras 4.2 y 4.1.



(a) Sesión 765



(b) Sesión 771

Figura 4.10: Señal de LFP de los canales 0 y 97 diezmada a 39.06Hz

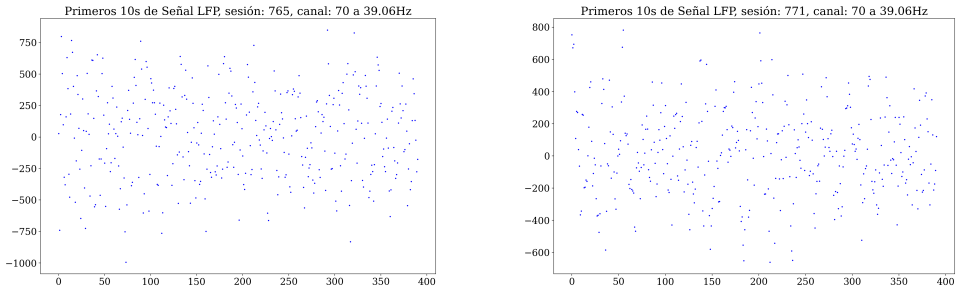


Figura 4.11: Gráfica de las señales de LFP a la frecuencia de muestreo de los ángulos (39.06Hz) durante los primeros 10 segundos del canal 70.

En la figura 4.11 apenas se aprecian los puntos correspondientes a las muestras de los LFPs. Además en la figura 4.12 se observa la señal de LFPs del canal 70 de cada sesión en comparación con los ángulos correspondientes durante alrededor de 5 minutos.

Por otro lado, la figura 4.12 muestra un tipo de un conjunto de LFPs (características de entrada a la red) más ángulos (las etiquetas o valores esperados) limpios y sincronizados al diezmar de manera exitosa entre las señales de potencial de campo local.

Tomando en cuenta que a la frecuencia de adquisición de los ángulos solamente se tendrían 39.06 muestras de LFPs por segundo y que la capacidad de movimiento del animal es más veloz que un segundo, entonces la información de los LFPs podría resultar poca para hacer una buena predicción.

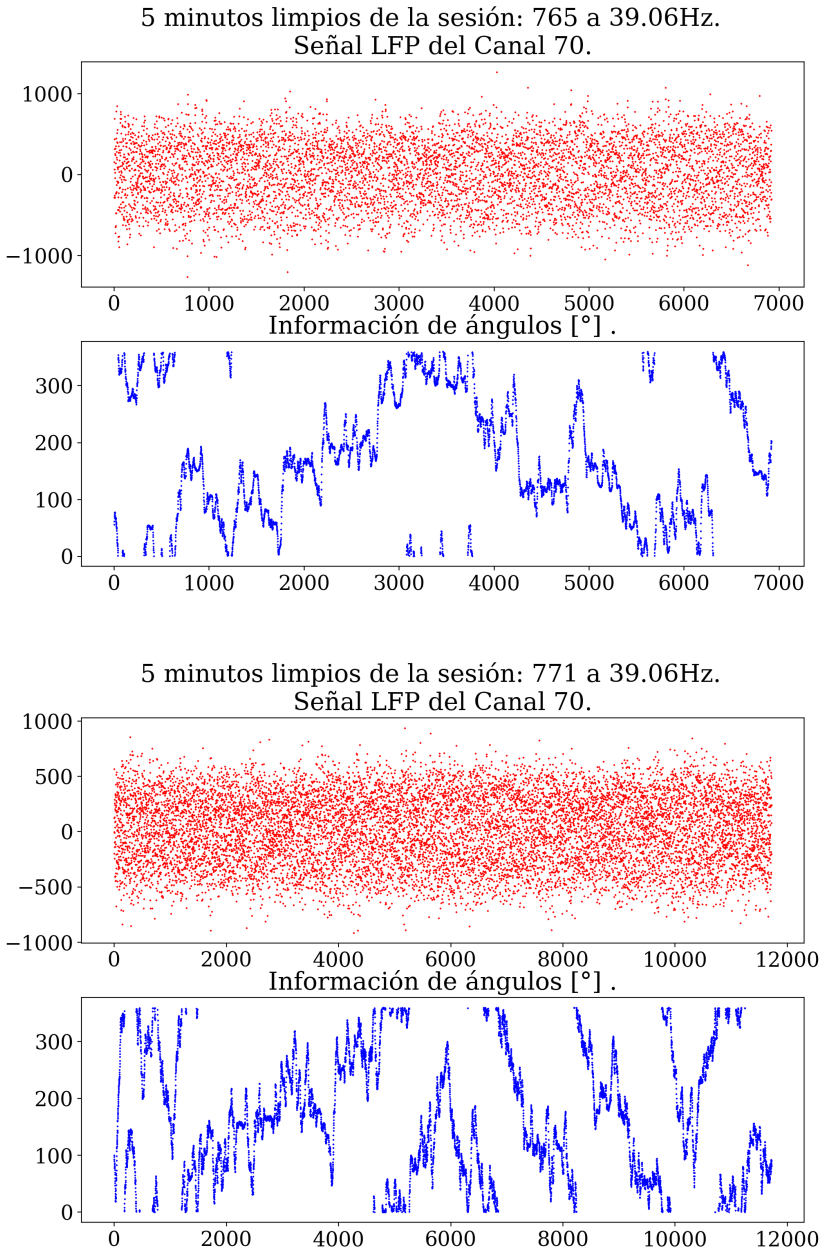
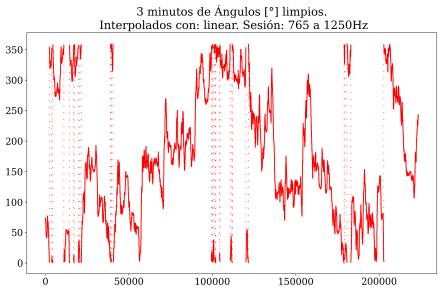


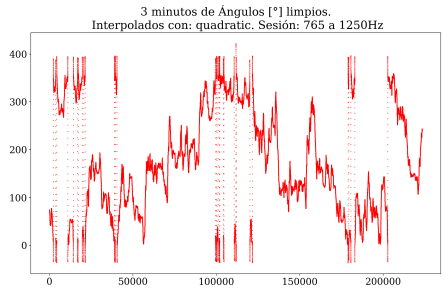
Figura 4.12: Gráfica de las señales de LFP del canal 70 y los ángulos a la frecuencia de muestreo de 39.06Hz en una ventana de $\sim 5min$.

4.1.4 Expansión de ángulos a 1250Hz e Interpolaciones

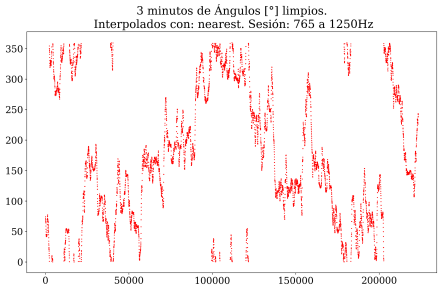
La expansión de ángulos por sí sola no es posible de apreciarla gráficamente pues se debe recordar que se rellenó con elementos "NaN". Una vez que estos valores de relleno se sustituyen con los calculados con los distintos métodos de interpolación es posible observar un cambio sutil en el conjunto de ángulos. La cantidad de muestras de relleno agregadas llevaron al incrementar el conjunto de los ángulos de las sesiones a 3433472 para la 771 y 3514368 para la 765. Este incremento del aproximado de 32 % responde a la diferencia de frecuencias entre los LFPs y los ángulos.



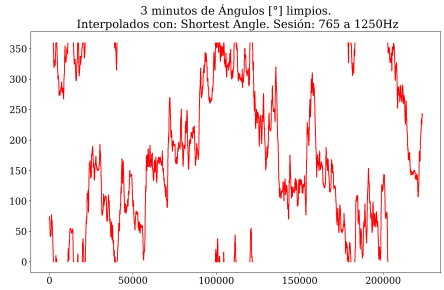
(a) Interpolación Lineal



(b) Interpolación Cuadrática

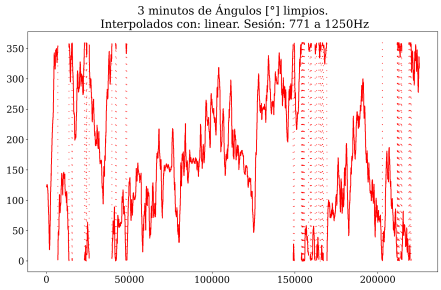


(c) Interpolación por valor más cercano (*nearest*)

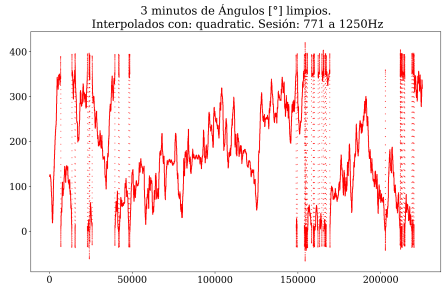


(d) Interpolación por el ángulo más corto)

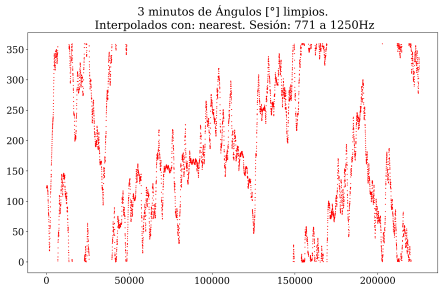
Figura 4.13: Ángulos de la sesión 765 limpios a la frecuencia de muestreo de las señales de LFP (1250Hz) durante $\sim 3min$. Utilizando las interpolaciones: lineal (arriba izquierda), cuadrática (arriba derecha), valor más cercano (abajo izquierda) y ángulo más cercano (abajo derecha).



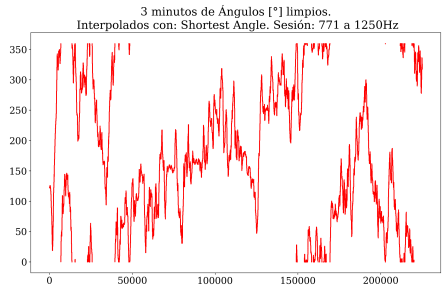
(a) Interpolación Lineal



(b) Interpolación Cuadrática



(c) Interpolación por valor más cercano (*nearest*)



(d) Interpolación por el ángulo más corto)

Figura 4.14: Ángulos de la sesión 771 limpios a la frecuencia de muestreo de las señales de LFP (1250Hz) durante $\sim 3min$ Utilizando las interpolaciones: lineal (arriba izquierda), cuadrática (arriba derecha), valor más cercano (abajo izquierda) y ángulo más cercano (abajo derecha).

De las figuras 4.13 y 4.14 se observa que la los métodos lineal y cuadrático interpolan por el ángulo incorrecto al movimiento de la rata, mientras que la interpolación al valor más cercano simplemente utiliza el dato más cercano directamente en la posición que se debe interpolar. A pesar de que este último es simple y no tan preciso en señales con muchas variaciones, en este caso proporciona un mejor resultado con la interpolación lineal o cuadrática. Lo anterior por dos principales razones, no interpola datos que vayan en contra de la naturaleza del movimiento como sí las 2 primeras. Y luego, considerando que la frecuencia de trabajo es lo suficientemente rápida comparada al movimiento físico del animal, utilizar el valor cercano es una aproximación bastante válida pues él ángulo de dirección de la cabeza entre una muestra y otra a la frecuencia original (39.06Hz) no varía tanto.

Por su parte la interpolación por el ángulo más corto parece ser la mejor opción ya que interpola siguiendo el movimiento real del sujeto y además, en rotaciones donde no se cruza por cero se produce una interpolación lineal que

permite completar la secuencia de datos a la frecuencia incrementada. De forma que, **para el resto del análisis que comprende el entrenamiento de la redes neuronales se utilizará la interpolación por el ángulo más corto cuando se haya sincronizado los datos a la frecuencia de los LFPs (1250Hz).**

La figura 4.15 es la contra parte de figura 4.12 en la cual se mostraban una ventana de tiempo de cerca de 5min de ambas sesiones con las gráficas de los LFPs en el canal 70 y los ángulos en grados. En lugar de diezmar los LFPs a 39.06Hz, se interpolaron los ángulos a 1250Hz obteniendo como resultado las gráficas de la figura 4.15. La diferencia más visible por ende, es la cantidad de puntos en ambas gráficas.

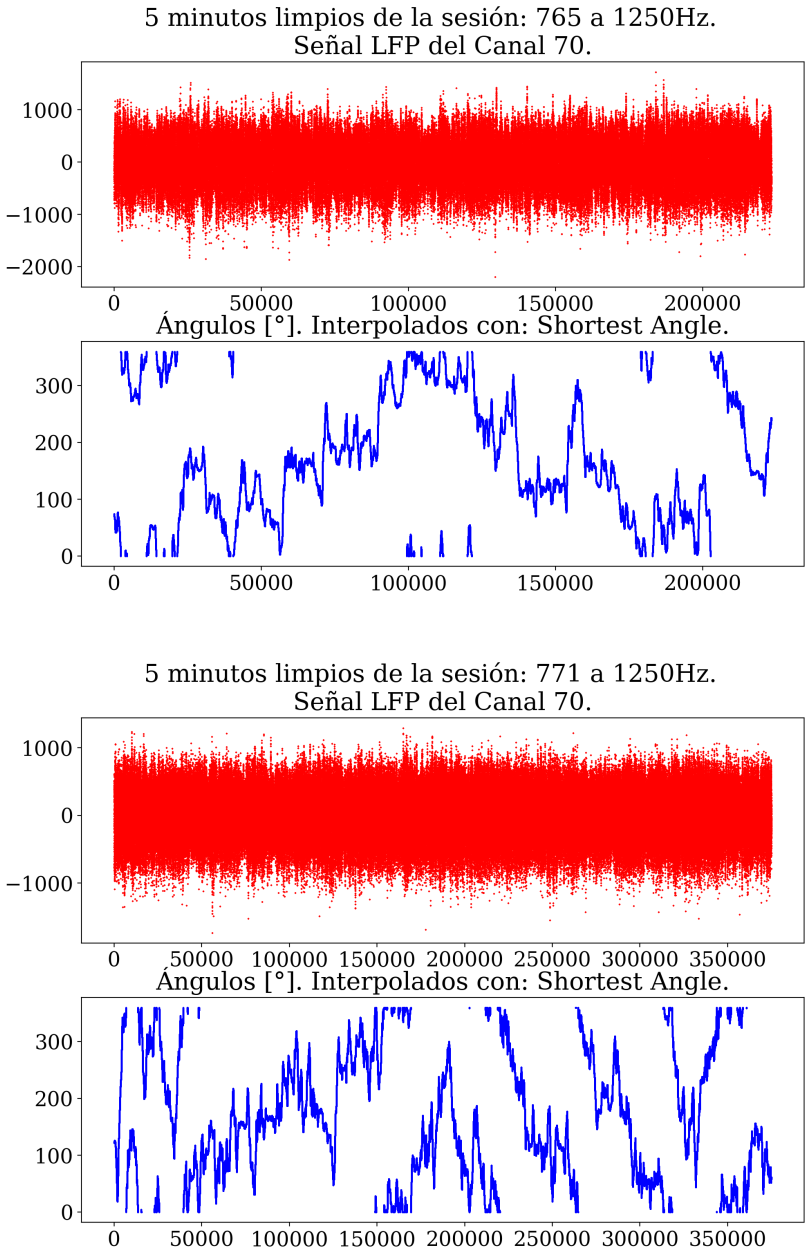


Figura 4.15: Gráfica de las señales de LFP del canal 70 y los ángulos a la frecuencia de muestreo de 1250Hz en una ventana de $\sim 5min$.

4.1.5 Limpieza de los datos

En cuanto a la limpieza de datos, este paso se realiza justo después de sincronizar frecuencias. Ya sea al diezmar los LFPs o rellenar los ángulos. En ambos casos, una vez sincronizados, se concatenan los datos en una sola estructura la cual pasa por el proceso de limpieza descrito anteriormente. La interpolación de los ángulos en caso de que hayan sido rellenados se realiza posterior a la limpieza y se interpola cada subconjunto resultante por separado de forma que asegure que se utilicen como puntos de referencia en la interpolación únicamente valores válidos.

En las tablas 4.1 y 4.2 se detallan los tamaños de los subconjuntos resultantes después de eliminar los datos inválidos. Si se comparan contra los tamaños contra los mencionados en la sección 4.1.2, vemos que los porcentajes en que se disminuyó el total de los datos es muy cercano al mencionado en esa sección. La diferencia está en que, en el caso de que se interpolaran los datos no se guardarían subconjuntos con menos de 32 muestras, pues no se habría logrado una interpolación correcta. Y en caso de que se diezmaran los LFPs no se guardaron subconjuntos con menos de 2 ángulos. Por esto, existe una muy ligera diferencia de los tamaños.

Subconjunto	Cantidad de Muestras	
	Interpolados (1250Hz)	Originales (39.06Hz)
1	223393	6982
2	46945	1468
3	228257	7134
4	68865	2153
5	**778593**	**24332**
6	353	12
7	425665	13303
8	694881	21716
9	65	3
10	70785	2213
11	166113	5192
12	78689	2460
13	457249	14290
14	28609	895
Total	3268462	102153

Cuadro 4.1: Cantidad de muestras por cada subconjunto generado después de la limpieza de ángulos inválidos de la sesión 765.

Subconjunto	Cantidad de Muestras	
	Interpolados (1250Hz)	Originales (39.06Hz)
1	**456481**	**14266**
2	801	26
3	641	21
4	92001	2876
5	240545	7518
6	311745	9743
7	136417	4264
8	311073	9722
9	212513	6642
10	227105	7098
11	8833	277
12	374113	11692
13	2689	85
14	56929	1780
15	9441	296
16	37665	1178
17	52321	1636
18	1185	38
19	737	24
20	16801	526
21	63361	1981
22	833	27
23	225	8
24	833	27
Total	2615288	80526

Cuadro 4.2: Cantidad de muestras por cada subconjunto generado después de la limpieza de ángulos inválidos de la sesión 771.

El subconjunto 5 de la sesión 765 y el 1 de la 771, los cuales aparecen resaltados en las tablas 4.1 y 4.2 respectivamente, son los que contienen más muestras. Estos son los que se dividirán en los datos en los conjuntos de entrenamiento, validación y prueba; y que se utilizarán para entrenar las redes neuronales.

4.1.6 Ángulo de Preferencia

Con el objetivo de comprender mejor la información contenida en las señales de potencial de campo local, se decide graficar la suma de los valores de LFPs del canal 70 por cada ángulo; con una resolución de 1 grado. De manera que cada muestra de LFP se asoció a un ángulo entre 0 y 360 grados.
Las gráficas para ambas sesiones se muestran en las figuras 4.16 y 4.17 para datos interpolados con el método del ángulo más corto; y en las figuras 4.18 y 4.19 cuando se sincronizaron los datos por medio del diezmado de los LFPS.

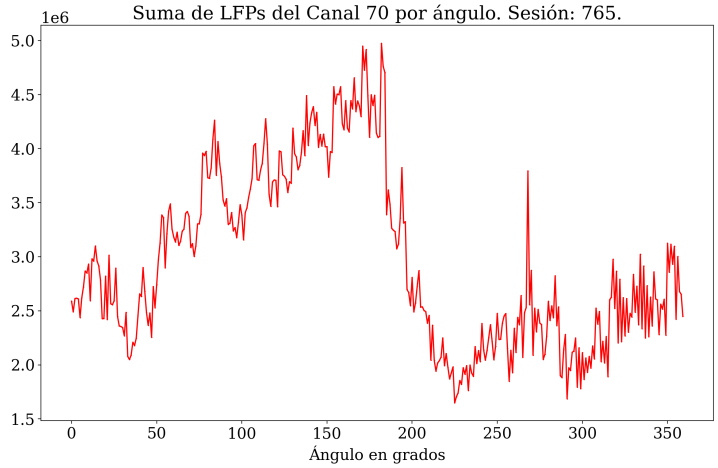


Figura 4.16: Gráfica de amplitud de los LFPs contra el ángulo de dirección de la cabeza en la sesión 765 a 1250Hz. Ángulo de preferencia: 182°

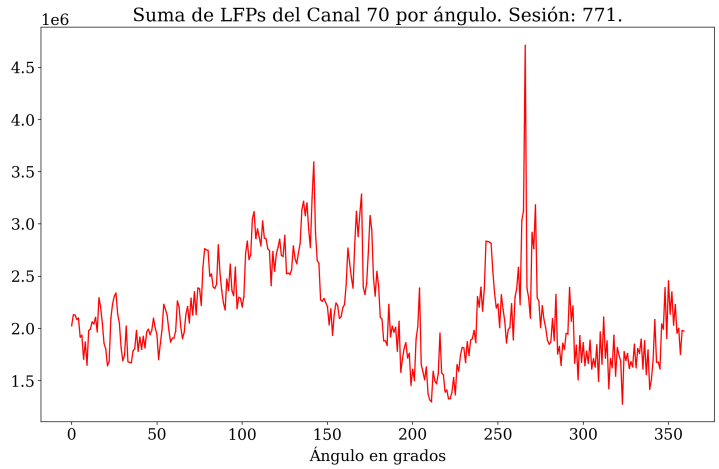


Figura 4.17: Gráfica de amplitud de los LFPs contra el ángulo de dirección de la cabeza en la sesión 771 a 1250Hz. Ángulo de preferencia: 266°

En las figuras 4.16 y 4.17 se aprecia claramente la predominancia de un ángulo de preferencia: 182° y 266° respectivamente. Estas gráficas demuestran la existencia de células de dirección de la cabeza (ver sección: 2.1.3) en la región del cerebro donde se encontraba el electrodo 70, tal y como se anticipaba en la sección 3.1.

Otro aspecto que se puede resaltar de las figuras 4.16 y 4.17 es que parece haber un segundo ángulo por región con bastante presencia de actividad. Esto podría deberse a que hayan neuronas de dirección de la cabeza con distinto ángulo de preferencia siendo grabadas por el mismo electrodo.

Por otro lado, de las figuras 4.18 y 4.19 se nota una respuesta similar al de los ángulos interpolados pero con más ruido. Tanto que incluso el ángulo de preferencia de la sesión 765 cambia de 182° a 154°. Este ruido pudo ser introducido por los filtros utilizados en el proceso de diezmado. Aunque el principal objetivo de reducir la frecuencia de las señales de LFPs por este método era conservar lo mejor posible la información de la señal, el diezmado no es perfecto.
Aunque este ruido podría estar alterando considerablemente los datos que se utilizarán como entrada a las redes neuronales, se decide de igual forma tomarlos en cuenta en el proceso de entrenamiento para comparar los resultados contra los obtenidos con datos interpolados.

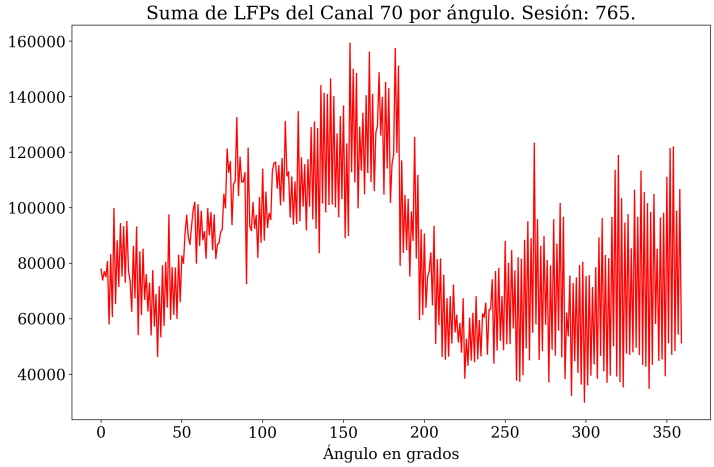


Figura 4.18: Gráfica de amplitud de los LFPs contra el ángulo de dirección de la cabeza en la sesión 765 a 39,06Hz. Ángulo de preferencia: 154°

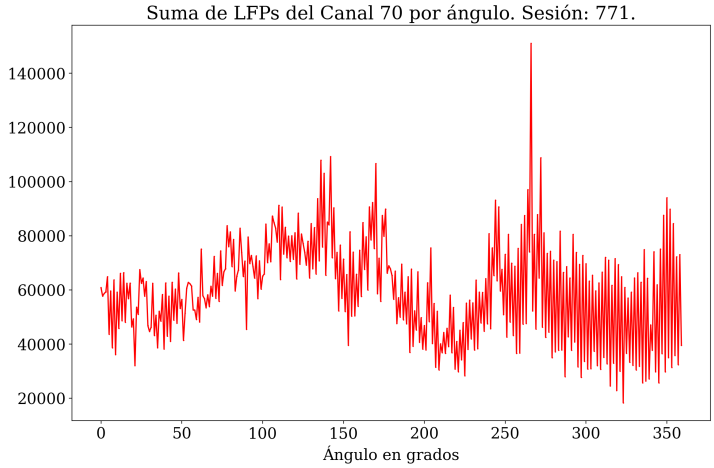


Figura 4.19: Gráfica de amplitud de los LFPs contra el ángulo de dirección de la cabeza en la sesión 771 a 39,06Hz. Ángulo de preferencia: 266°

4.2 Comparación de predicciones ante variaciones en los parámetros

Para comenzar el proceso de entrenamiento, se deben tener claro los hiperparámetros con los que se puede ajustar el aprendizaje de las redes neuronales; tanto desde el conjunto de datos con el que se le va a alimentar, como con la

configuración de entrenamiento de las mismas.
Los parámetros que corresponden al ajuste de los "*features*" y etiquetas de entrada se enlistan en el cuadro 4.3. Mientras que los factores que permiten potencialmente modificar la velocidad y rendimiento del entrenamiento en los modelos de predicción están en el cuadro 4.4.

Hiperparámetro	Valor (por defecto)
Sesión	765 ó 771
Método de Sincronización	"Diezmado" ó "Interpolación"
Interpolación	("Ángulo más corto") ver sección 3.2.6
Canal de LFP	[0, 96] (40, 70 y 80) ver sección 3.1
Tamaño de ventana	Entero
Tamaño de lotes	Entero (32)

Cuadro 4.3: Hiperparámetros para la generación del conjunto de datos de entrada para las redes neuronales.

Hiperparámetro	MLP	CNN	LSTM	Valor (por defecto)
Número de Capas	Sí	No	No	Entero (3)
Neuronas por capa	Sí	Sí	Sí	Entero (32)
Épocas	Sí	Sí	Sí	Entero
Dropout	Sí	Sí	Sí	[0, 1] (0.2)

Cuadro 4.4: Hiperparámetros para el entrenamiento de las redes neuronales.

Seguidamente se mostrarán los resultados usando distintas combinaciones de estos hiperparámetros con dos objetivos en el siguiente orden de prioridad:

- Obtener el mejor rendimiento en la predicción de los ángulos basándose en señales de LFP de entrada. En otras palabras, obtener la predicción con menor error en el conjunto de validación.
- Conseguir acelerar el entrenamiento sin sacrificar el rendimiento de las predicciones.

Como se aprecia en los cuadros 4.3 y 4.4, existe una enorme cantidad de combinaciones de los hiperparámetros anteriores. Solamente para analizar todos los canales se necesitarían 96 predicciones distintas dejando el resto de parámetros intactos. Es por esta razón que algunos parámetros se manejaran con valores preestablecidos, lo que permitirá concentrar en análisis en el resto (los resaltados en ambas tablas) que podrían repercutir en mayor medida al resultado de las redes.

Parámetros como el canal de LFP y la interpolación a utilizar se escogen del análisis en el capítulo 3. En el cual, conforme se desarrollaba la implementación para las pruebas, surgieron hallazgos que direccionaban a la mejor opción de acuerdo a las referencias consultadas, la naturaleza del problema y el objetivo de la investigación.

Existen otros hiperparámetros, como el porcentaje de regularización por *dropout*, la cantidad de ventanas por lote, el número de capas y la cantidad de neuronas por capa, que aportan principalmente a la velocidad de aprendizaje por encima de la mejora en el rendimiento de la predicción. Los valores escogidos son muy comunes para comenzar el entrenamiento de redes neuronales ya que al ser valores relativamente bajos, permiten un rápido entrenamiento. Una vez encontrada la mejor combinación con los otros parámetros, estos podrían ajustarse para terminar de "pulir" el resultado final.

4.2.1 Ángulos Interpolados

Para el análisis se comenzará utilizando como método de sincronización los ángulos interpolados a 1250Hz, o sea, la frecuencia de muestreo de los LFPs.

El primer parámetro que se decide estudiar es la cantidad de épocas necesarias para entrenar las redes.
Para esto se comienza con un valor alto, de manera que se pueda observar el rendimiento de las redes en el conjunto de entrenamiento y validación época por época. En caso de que a mayor cantidad de épocas no se aprecie una mejoría general en las predicciones, se podría decir reducir la cantidad de épocas para acelerar el entrenamiento.

LSTM

Se decide comenzar entrenando la red LSTM durante 10 épocas con el canal 40 la sesión 771 y ventanas de tiempo de 100ms:

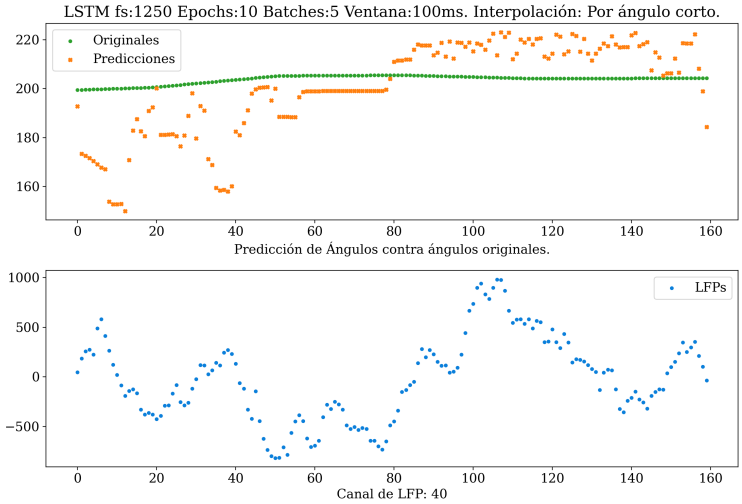


Figura 4.20: Gráfica de 5 lotes continuos del conjunto de prueba de la sesión 771 con la predicción de ángulos (naranja) contra ángulos esperados (verde) y LFPs (azul). La duración es de 5 lotes de datos de 32 ventanas de 100ms: 16 segundos. Modelo: LSTM

En la figura 4.20 parece mostrar que la red fue capaz de aprender particularmente 2 cosas:

- Que se están prediciendo valores entre 0 y 360 grados.

- Que existe una relación entre las muestras. De hecho, conforme avanza la predicción, parece que va corrigiendo su error al acercarse cada vez más a los datos originales.

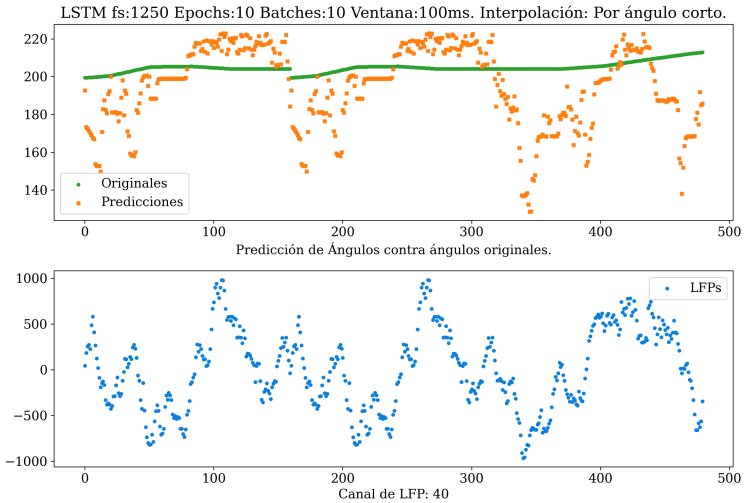


Figura 4.21: Gráfica de 10 lotes continuos del conjunto de prueba de la sesión 771 con la predicción de ángulos (naranja) contra ángulos esperados (verde) y LFPs (azul). Duración de 32 segundos. Modelo: LSTM

Sin embargo, cuando se amplía la ventana de tiempo al doble, se nota cómo la señal de LFPs es reflejada en las predicciones. Se aprecia un patrón periódico en la figura 4.21.
Al ampliar aún más la ventana de tiempo de la predicción a 20 y 50 lotes (64s y 160s respectivamente), los cuales se observan en las figuras 4.22 y 4.23, la tendencia es cada vez más clara.

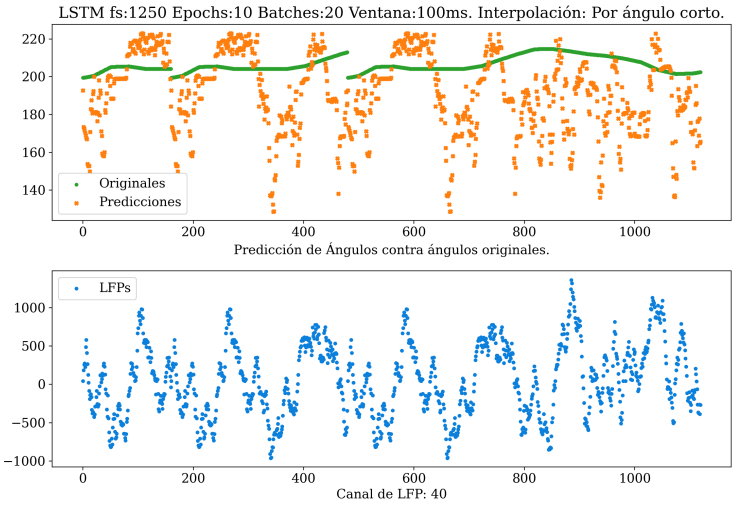


Figura 4.22: Gráfica de 20 lotes continuos del conjunto de prueba de la sesión 771 con la predicción de ángulos (naranja) contra ángulos esperados (verde) y LFPs (azul). Duración de 64 segundos. Modelo: LSTM

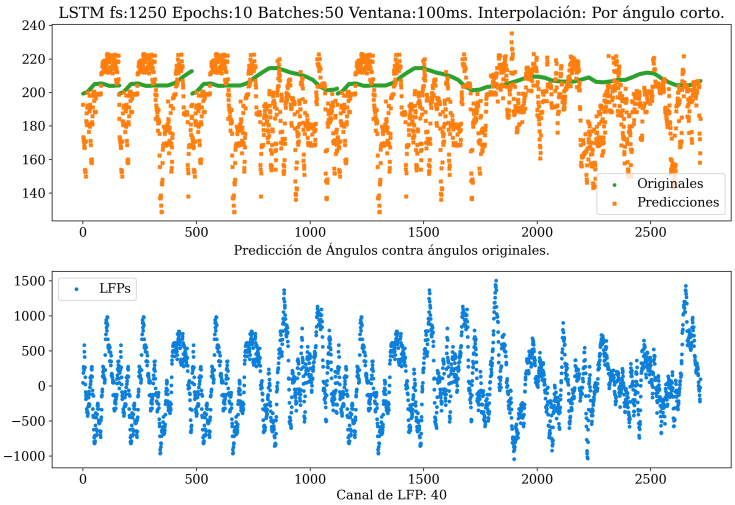


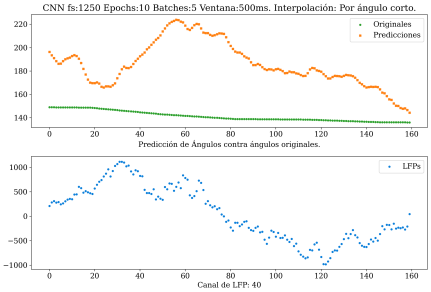
Figura 4.23: Gráfica de 50 lotes continuos del conjunto de prueba de la sesión 771 con la predicción de ángulos (naranja) contra ángulos esperados (verde) y LFPs (azul). Duración de 160 segundos. Modelo: LSTM

Al cabo de las 10 épocas, la red terminó con un error en el conjunto de entrenamiento de 40.16°. Mientras que en el conjunto de validación se obtuvo un error de 82.82°. Esto indica que la red se sobre ajustó a los datos de en-

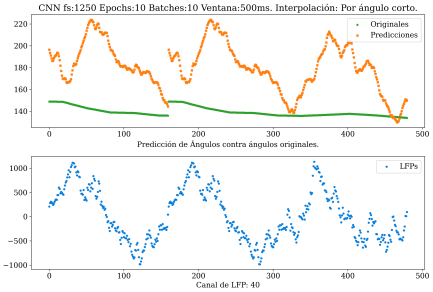
trenamiento. A pesar de que 40° es un error muy grande en la predicción de un ángulo, claramente se tiene un mucho mejor resultado en el entrenamiento que en la validación, provocando ***overfitting***.

CNN

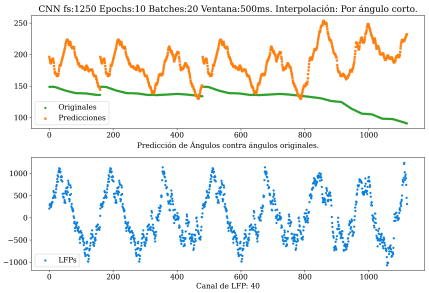
De igual forma, se utiliza la misma configuración para probar la red neuronal convolucional con 2 diferencias: Se utilizan los datos de la sesión 765 y se aumenta el tamaño de las ventanas de 100ms a 500ms.



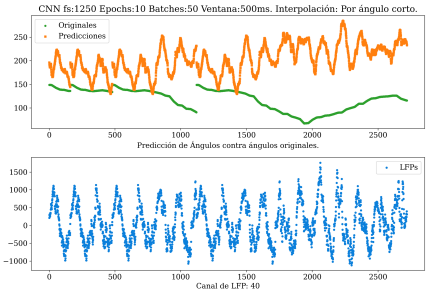
(a) 5 lotes: 1min 20s



(b) 10 lotes: 2min 40s



(c) 20 lotes: 5min 20s



(d) 50 lotes: 13min 20s

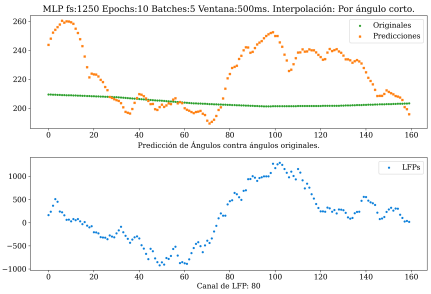
Figura 4.24: Gráficas de la sesión 765, canal 40 con la predicción de ángulos (naranja) contra ángulos esperados (verde) y LFPs (azul). Modelo: CNN

En las gráficas de la figura 4.24 se ve que la CNN "absorbe" aún más el comportamiento de los LFPS y los refleja en las predicciones; incluso desde la ventana más pequeña en la gráfica 4.24a. En la figura 4.24b se ve una pequeña corrección que hace la red con respecto a los ángulos.
Además, las predicciones tiene una media más alta en comparación con las obtenidas con la LSTM en referencia a los ángulos originales. En el caso de la LSTM, aunque variaban mucho las predicciones, mantenían una media similar a la de los ángulos originales.

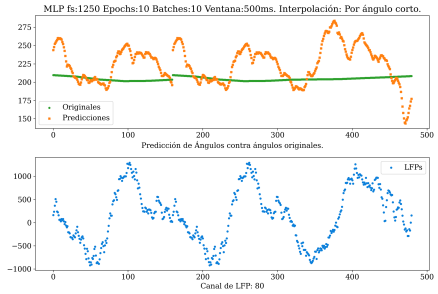
Se debe destacar que no se aprecia ninguna mejoría al aumentar la ventana a 500ms ni al usar los datos de la sesión 765. Los resultados de rendimiento en la última época fueron de 39.73° en el subconjunto de entrenamiento y de 98.18° en el de validación. Nuevamente se da un *overfitting*.

MLP

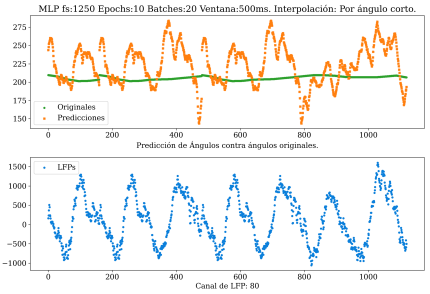
Con respecto a la red neuronal clásica, se decide igualmente hacer ligeras variaciones en los otros parámetros en busca de menor error. Se mantiene la ventana de 500ms, pero se devuelve al conjunto de la sesión 771 y se cambia a probar el canal 80 de los LFPs.



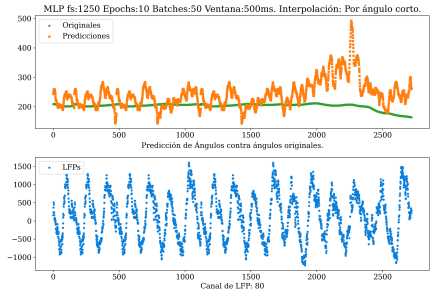
(a) 5 lotes: 1min 20s



(b) 10 lotes: 2min 40s



(c) 20 lotes: 5min 20s



(d) 50 lotes: 13min 20s

Figura 4.25: Gráficas de la sesión 771 con la predicción de ángulos (naranja) contra ángulos esperados (verde) y LFPs (azul). Modelo: MLP

En las gráficas de la figura 4.25 están los resultados de la predicción con una perceptrón multicapa. Esta red tiene la ventaja de ser la más rápida de entrenar de todas, sin embargo los resultados siguen sin ser mejores que los obtenidos con la LSTM.
La primer impresión es que la gráfica 4.25d tiene una predicción bastante prometedora, pues a pesar del ruido introducido por el sobreajuste a la señal de LFP, se mantiene bastante cerca de los datos reales de los ángulos. No obstan-

te si se revisa con detenimiento, la escala es más grande que la de por ejemplo la figura equivalente de la LSTM (4.23). De hecho, es la única red que predice ángulos por encima de los 360°, lo cual dado la escala definida desde el inicio de la implementación es imposible.

Revisando el error obtenido en la última época del entrenamiento de la red neuronal clásica, se consiguió un 41.98° de Error Medio Absoluto en el entrenamiento, y 108.5° en la validación. Es la red con peor rendimiento de las 3 revisadas en términos de error y sobre-ajuste.

Comparación de redes neuronales usando datos interpolados

En el cuadro 4.5 se encuentran un resumen del error obtenido para cada red neuronal bajo estudio, sobre los conjuntos de entrenamiento y validación.

Métrica	**Error Medio Absoluto (MAE) [grados]**							
Datos	**Entrenamiento**				**Validación**			
Canal	**80**	**40**		**70**	**80**	**40**		**70**
Ventana	**500**	**500**	**100**		**500**	**500**	**100**	
Sesión	**771**	**765**	**771**		**765**	**771**		
Época	**MLP**	**CNN**	**LSTM**		**MLP**	**CNN**	**LSTM**	
1	50.03	41.97	84.20	84.31	85.33	108.8	82.36	**78.61**
2	46.54	39.73	46.10	45.21	88.21	108.1	93.52	87.33
3	45.79	39.14	40.62	41.21	87.79	89.41	89.64	89.36
4	45.05	39.06	39.90	40.10	83.71	106.2	95.18	86.31
5	44.24	38.92	40.38	39.68	87.34	116.4	85.50	92.87
6	43.74	**38.88**	40.14	40.27	101.7	145.1	90.01	80.64
7	43.94	38.91	41.17	40.08	87.16	118.4	89.67	96.80
8	43.20	39.44	39.83	40.13	95.80	113.9	88.85	86.92
9	42.52	39.88	40.59	39.79	89.98	122.4	92.40	85.83
10	41.98	39.73	40.16	39.50	108.5	98.18	82.82	85.63
Tiempo	**0:29**	0:47	1:15	1:15				

Cuadro 4.5: Resumen de resultados de rendimiento de las 3 redes neuronales durante 10 épocas utilizando interpolación por el ángulo más corto de los ángulos (frecuencia de sincronización: 1250Hz). El tamaño de la ventana está dado en milisegundos [ms] y el tiempo total de la ejecución en [h:min].

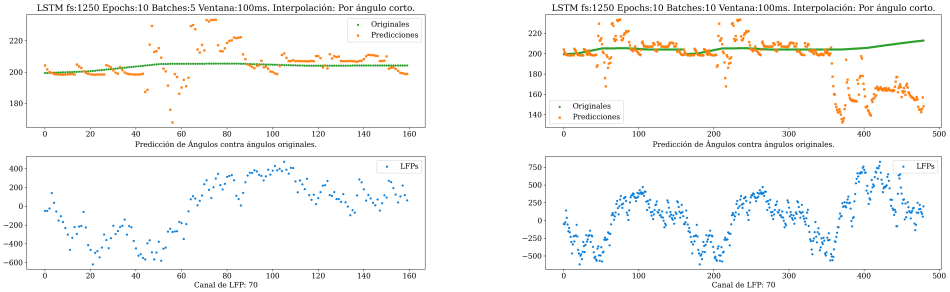
Del cuadro 4.5 se pueden resaltar los siguientes puntos:

- Todas las redes presentan *overfitting*. El mejor balance entre el error en el entrenamiento y en el de validación se da en **la primer época**.

- La red neuronal clásica es la más rápida de entrenar, pero la que produce el error más alto.
- La red LSTM es la que produce el menor error, pero es cerca del doble de veces más lenta que las otras 2 opciones.
- El error obtenido con todas las redes, aún con la LSTM, es muy grande. Lo suficiente como considerar que no es confiable para predecir el ángulo de dirección de la cabeza basándose en un canal de LFPs que contienen información de células de dirección de la cabeza.
- Las variaciones entre canales, ventanas de tiempo y sesiones de grabación no son un factor realmente determinante de mejora en las predicciones.

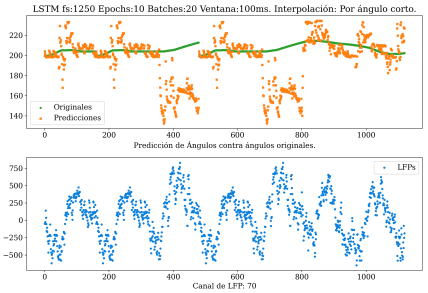
Mejorando el rendimiento de la LSTM

Después de revisar con más detalle los resultados de los tres tipos de redes neuronales, como se observa en el cuadro 4.5 se ejecutó una vez más el entrenamiento de la red LSTM para corroborar que realmente presente un mejor rendimiento en comparación a las otras redes. Se utiliza el mismo tamaño de ventana (100ms) y sesión (771) que la ejecución anterior de la LSTM. De igual forma, se entrena por 10 épocas, pero cambiamos el canal de LFPs al 70. Las predicciones se encuentran en la figura 4.26. Similares, aunque ligeramente mejores que las de la primer ejecución. De manera que se comprueba que la red LSTM tiene el mejor rendimiento en el caso de interpolar ángulos utilizando interpolación por el ángulo más corto.

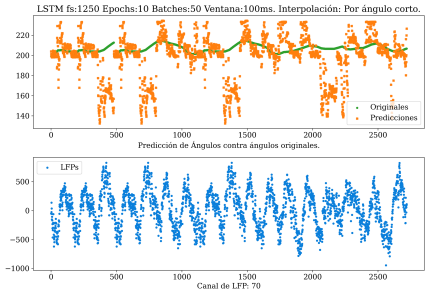


(a) 5 lotes: 16 segundos

(b) 10 lotes: 32 segundos



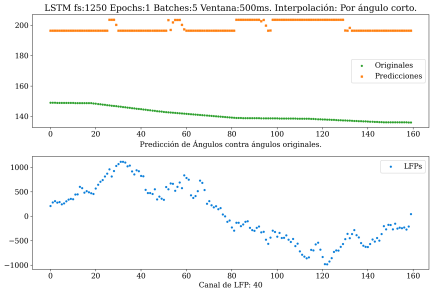
(c) 20 lotes: 64 segundos



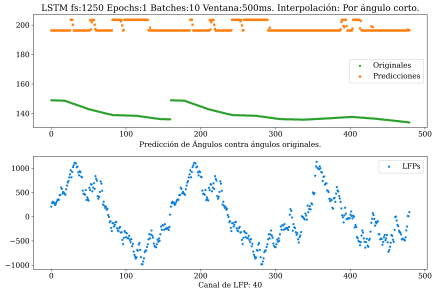
(d) 50 lotes: 160 segundos

Figura 4.26: Gráficas de la sesión 771, canal 70 con la predicción de ángulos (naranja) contra ángulos esperados (verde) y LFPs (azul). Modelo: LSTM

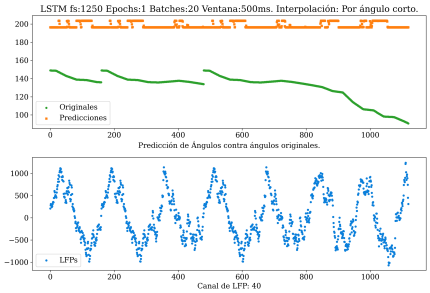
Tomando en cuenta que la red LSTM en la primer época presenta el mejor rendimiento, se procede a predecir los ángulos de la sesión 765 con una sola época, basado en el canal 40 de los LFPs. Lo anterior, para corroborar que la sesión y el canal no son factores que mejoren el rendimiento en la predicción. Además se incrementa el tamaño de la ventana a 500ms, mismo valor que se utilizó en la otras redes.



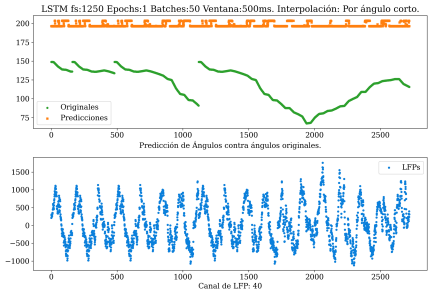
(a) 5 lotes: 1min 20s



(b) 10 lotes: 2min 40s



(c) 20 lotes: 5min 20s



(d) 50 lotes: 13min 20s

Figura 4.27: Gráficas de la sesión 765, canal 40 con la predicción de ángulos (naranja) contra ángulos esperados (verde) y LFPs (azul). Modelo: LSTM

En este último escenario se obtuvo un error en el entrenamiento de 70.14° y en la validación de 78.70°, lo cual era esperado de acuerdo al cuadro 4.5. Así también se obtuvo una mejor predicción de los ángulos con una época en la figura 4.27 que con 10 épocas y cualquiera de los 3 modelos antes analizados. Con este resultado se puede observar que no está sobreajustado a los datos de entrenamiento y por lo tanto, no absorbe tanto la dinámica de los los LFPs. Pero se siguen teniendo predicciones bastante lejanas a las esperadas. Adicionalmente, el tiempo de entrenamiento para esta configuración de ventanas de 500ms de datos con una frecuencia de 1250Hz en una sola época fue de 1 hora y 4min.

Después de graficar hasta 13 minutos de grabación, surge la inquietud de cómo se comportará la predicción ante cruces por cero, y a lo largo de una ventana más larga de tiempo. De manera que se genera la figura 4.28 que contiene la predicción de todos los datos del conjunto de prueba de la sesión 765 utilizando los mismos parámetros que la combinación anterior, con la única diferencia de que la ventana usada era de 100ms en lugar de 500ms.

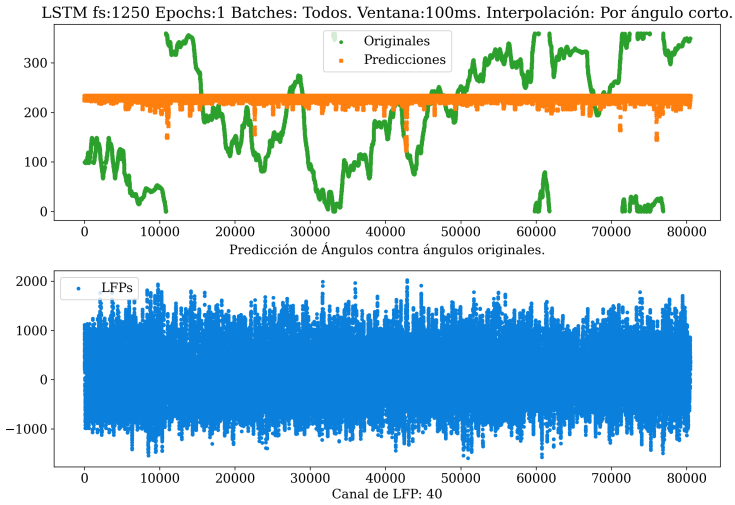
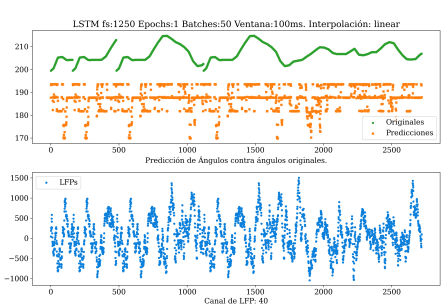


Figura 4.28: Gráfica de todo el conjunto de prueba de la sesión 765 con la predicción de ángulos (naranja) contra ángulos esperados (verde) y LFPs (azul). Modelo: LSTM

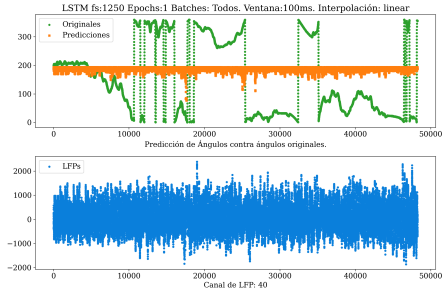
En la figura anterior (4.28) lo que más destaca es que a lo largo del tiempo, el predictor no se ajusta a las variaciones lentas de los ángulos. Se mantiene una linea recta a la mitad con ligeros intentos de ajuste, que terminan regresando a la tendencia que traía.

Interpolación Lineal

Con el fin de comparar el rendimiento de la interpolación por el ángulo más corto sobre el resto, se escoge la interpolación lineal para entrenar una LSTM con los datos de la sesión 771, con el canal 40 de los LFPs, un tamaño de ventana de 100ms y durante solo una época. Los parámetros anteriores son muy similares los mismos a los del primer escenario de la red LSTM (sección 4.2.1), con la diferencia de que fue entrenada solamente una época en lugar de 10.



(a) 50 lotes: 160 segundos



(b) Todo el conjunto de datos de prueba

Figura 4.29: Gráficas de la sesión 771, canal 40 con la predicción de ángulos (naranja) contra ángulos esperados (verde) y LFPs (azul). Modelo: LSTM. Interpolación Lineal.

Los resultados de la interpolación lineal presentes en la figura 4.29 muestran un comportamiento similar a las interpoladas por el ángulo más corto. Las líneas verdes verticales de la figura 4.29b corresponden a la interpolación lineal de rotaciones con cruce por 0°.

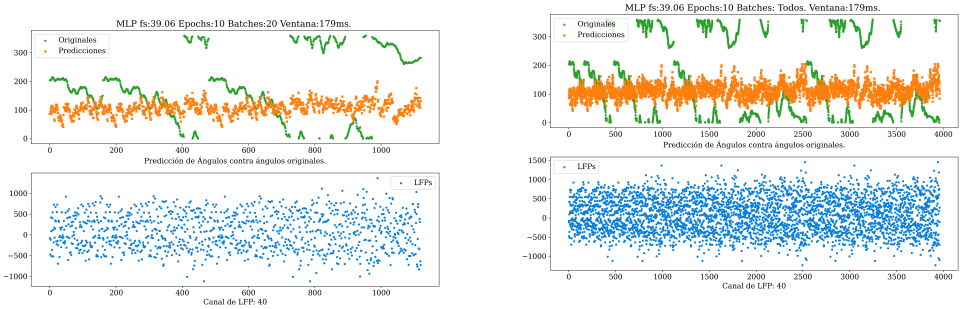
Este método consiguió un error de entrenamiento de 84.45° y de validación de 80.16°, muy similares obtenidos con el otro método de interpolación, aunque no mejores. La similitud de rendimiento con ambos algoritmos de interpolación es esperado, pues es solo en los cruces por cero donde difieren los cálculos. Estos eventos son poco comunes comparado a la gran cantidad de muestras que tiene cada conjunto de ángulos.

4.2.2 Señales de LFPs diezmadas

Ahora se busca analizar el rendimiento de las predicciones utilizando el otro método de sincronización de frecuencias: diezmar las señales de potencial de campo local.
Se parte del hecho de que tanto la sesión de grabación como el canal a utilizar en el entrenamiento de las redes no afecta significativamente la predicción. Además en este caso, no se necesitan probar los distintos métodos de interpolación pues los ángulos no pasan por este proceso.

MLP

Comenzando con la red neuronal clásica. Se utiliza una ventana de 179ms y se entrena por 10 épocas.



(a) 20 lotes: 1min 54segundos

(b) Todo el conjunto de datos de prueba

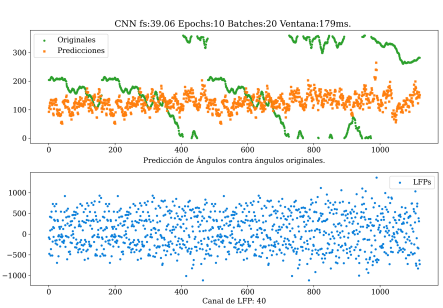
Figura 4.30: Gráficas de la sesión 771, canal 40 con la predicción de ángulos (naranja) contra ángulos esperados (verde) y LFPs (azul). Modelo: MLP. Diezmando las señales LFP.

En la figura 4.30 están las gráficas resultantes. No es tan claro apreciar un relación con los LFPs, sin embargo se mantiene la tendencia ruidosa de la MLP que no contempla la localidad espacial ni temporal entre muestras. Obtuvo como error en el entrenamiento al cabo de las 10 épocas de 85.7° y en la validación de 85.92°. Lo cual indica que no son datos sobre ajustados.

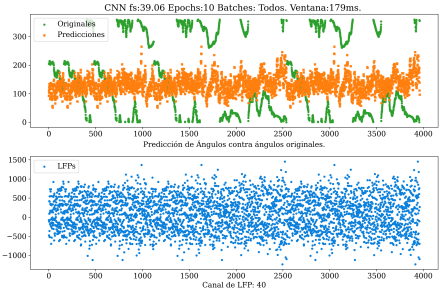
El entrenamiento fue mucho más rápido que el de los datos interpolados. Se logró entrenar el perceptrón multicapa en tan solo 14 segundos.
Aunque ciertamente la red LSTM con datos interpolados sigue proporcionando mejores resultados.

CNN

La red neuronal convolucional, que sí considera la localidad espacial de los datos obtuvo un resultado muy similar al MLP, tal y como se observa en la figura 4.31.



(a) 20 lotes: 1min 54segundos



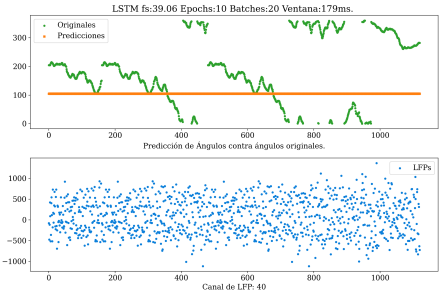
(b) Todo el conjunto de datos de prueba

Figura 4.31: Gráficas de la sesión 771, canal 40 con la predicción de ángulos (naranja) contra ángulos esperados (verde) y LFPs (azul). Modelo: CNN. Diezmando las señales LFP.

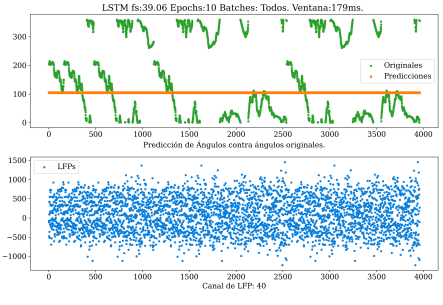
Esta red obtuvo después de las 10 épocas 85.44° de error en los datos de entrenamiento y 81.33° en los de validación, mejorando por poco los resultados del MLP. Su entrenamiento también fue corto, de 15 segundos.

LSTM

En lo que corresponde a la red neuronal recurrente LSTM, sus predicciones se muestran en la figura 4.32



(a) 20 lotes: 1min 54segundos



(b) Todo el conjunto de datos de prueba

Figura 4.32: Gráficas de la sesión 771, canal 40 con la predicción de ángulos (naranja) contra ángulos esperados (verde) y LFPs (azul). Modelo: LSTM. Diezmando las señales LFP.

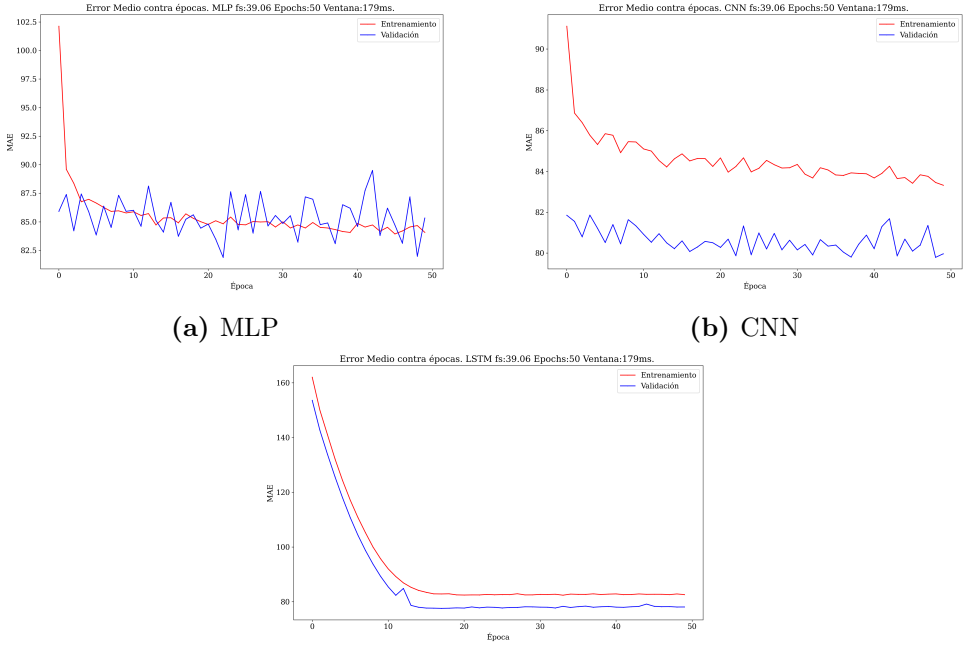
Contrario a los resultados anteriores, el error de la LSTM entrenada durante 10 épocas con datos diezmados fue el más alto. Se obtuvo un 95.70° en el entrenamiento y 89.29 en la validación al finalizar los 26 segundos que tomó

su entrenamiento.
La diferencia radica en que en los otros dos tipos de redes neuronales el error de ambos conjuntos de datos no disminuyó, de hecho se mantuvo prácticamente igual. Esto se discute con más detalle en la siguiente sección.

Variación en los otros hiperparámetros

Épocas:

Dado que otro ninguno de los resultados de las distintas redes con datos diezmados parece estar sobre ajustados, es válido preguntarse si más bien se está en el otro extremo: *underfitted*. Por esta razón, se vuelven a entrenar las 3 redes neuronales bajo la misma configuración anterior, pero esta vez aumentando a 50 el número de épocas a entrenarlas, con el fin de determinar cuantas son necesarias para la red tenga un buen rendimiento en ambos conjuntos de entrenamiento.



(a) MLP

(b) CNN

(c) LSTM

Figura 4.33: Gráficas Error Absoluto Medio contra Épocas. Utilizando datos diezmados a 39.06Hz del canal 40 de la sesión 771 con ventanas de 179ms durante 50 épocas. Datos de entrenamiento en rojo y validación en azul.

En la figura 4.33 se encuentran las gráficas de error contra épocas. Se apre-

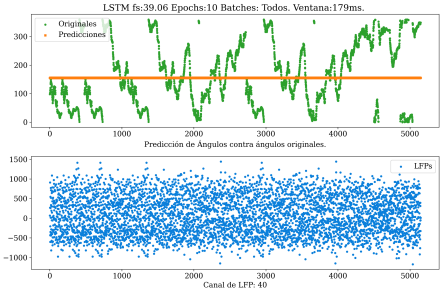
cia que para el caso de la CNN y el MLP se alcanza muy rápidamente el error más bajo para el entrenamiento (antes de las 5 épocas) y a partir de ahí el error en la validación comienza a fluctuar sin mejoría.

Por su parte la red LSTM tarda más en llegar al punto donde los errores dejan de disminuir, entre las 13 y 20 épocas. Lo cual indica que efectivamente el resultado de la LSTM con datos diezmados anterior aun no había llegado a su máximo aprendizaje. Tomando el dato exacto de los errores al finalizar las 50 épocas en la LSTM, se tuvo un error en el entrenamiento de 82.62° y 78.1° en la validación. Resultados que son ligeramente **mejores** a los obtenidos con la misma red neuronal LSTM, pero que se entrenó con datos interpolados a 1250Hz.

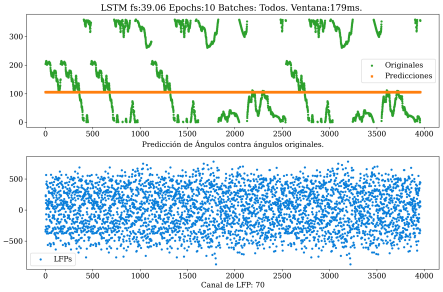
La enorme ventaja con respecto a los datos interpolados, es que con los datos diezmados la LSTM terminó su entrenamiento con 50 épocas en solamente 1 minuto y 42 segundos, contra la hora y 15 minutos de su contraparte. Obteniendo de todas formas un rendimiento incluso mejor que con datos interpolados.

Sesión, Canal y número de Ventanas:

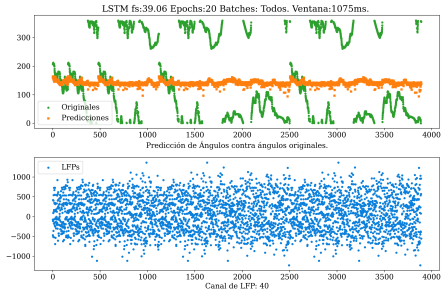
Para finalizar, se hacen tres variaciones más a la LSTM que se entrenó por 10 épocas con ventanas de 179ms del canal de LFP 40 de la sesión 771. Como se muestran en la figura 4.34.



(a) Cambio de sesión 771 por la sesión 765.



(b) Cambio del canal de LFP de la sesión 771 del 40 al 70.



(c) Incremento de ventana a 1.075 segundos y de épocas a 20.

Figura 4.34: Gráficas de todo el conjunto de prueba con la predicción de ángulos (naranja) contra ángulos esperados (verde) y LFPs (azul). Modelo: LSTM

El cambio de sesión resultó favorable en la disminución del error porque se obtuvo 79.78° en el entrenamiento y 75.96° en la validación, los cuales corresponden a los valores más bajos de todos las combinaciones probadas. Sin embargo, este resultado se podría atribuir a la coincidencia de que existe mucho giro de cabeza que atraviesa cerca de los 150° (como se observa en la figura 4.34a) en donde la red neuronal está prediciendo una línea recta que no se asemeja para nada al comportamiento real del ángulo del sujeto.

Con respecto al cambio de electrodo en la figura 4.34b, se obtiene errores de cerca de 90 grados que quedan bastante lejos de otros ya obtenidos.

En cuanto a la imagen 4.34c, al incrementar la ventana de 100ms a 1 segundo se obtuvo un error de entrenamiento de 82.43 grados y de 78.91 grados en la validación. Un rendimiento comparable a los otros 2 mejores. No obstante, como se aprecia en la figura 4.35, la LSTM justo llegaba a su máximo aprendizaje y por el contrario parece que comenzaría a deteriorarse. Por lo

que pensar en incrementar las épocas para obtener un mejor rendimiento no es viable en este caso.

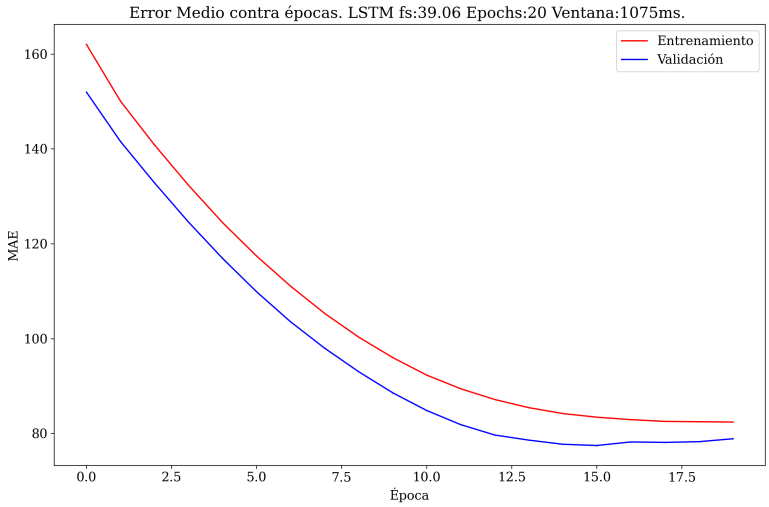


Figura 4.35: Gráficas Error Medio Absoluto contra Épocas al incrementar la ventana a 1s. Modelo: LSTM

5 Conclusiones y recomendaciones

Finalizada la investigación, se enlistan las siguientes conclusiones.

5.1 Conclusiones

- Se proporcionaron dos métodos de extracción de características, basados investigación detallada sobre el tipo de datos disponibles: por diezmado y por interpolación, que permitieron crear los datos de entrada a las distintas redes neuronales. Para esto se consideró: la región del cerebro con la mayor cantidad del tipo específico de neuronas que contienen información sobre orientación, los métodos de análisis de Potenciales de Campo Local, así como de series de tiempo y opciones de modificación de frecuencia de señales.

- Se implementaron dos tipos de redes neuronales, escogidas de manera que su estructura le permitiera reconocer patrones en series de tiempo y por cercanía espacial. Además de implementarse una red neuronal clásica para usarse como punto de referencia en cuanto a rendimiento y velocidad de aprendizaje de las redes propuestas.

- Se concluye que la mejor combinación de entradas y algoritmo de predicción es utilizar datos de LFPs diezmados bajo el procedimiento descrito y alimentarlos a una red LSTM que lleve a cabo la predicción. Esta solución fue la que obtuvo el menor error y con un tiempo de aprendizaje considerablemente rápido, capaz de ser entrenado sin costosos requerimientos de hardware.

A pesar de concluir la investigación cumpliendo con los objetivos planteados, no se obtiene un error entre las predicciones de los ángulos y los valores reales confiable y preciso. Por esta razón se hacen las siguientes recomendaciones:

5.2 Recomendaciones

- Investigar sobre otras alternativas de extracción de características a partir de señales de LFP. Contemplar la utilización de múltiples electrodos.

- Utilizar el ambiente de programación elaborado (disponible en https://github.com/PabloAvLo/LFP-Classification) como base de futuras investigaciones, expandiendo sus librerías y funcionalidades intentando mantenerla lo más general posible. Su documentación puede consultarse en: https://pabloavlo.github.io/LFP-Classification/.

- Enfocar los esfuerzos de entrenamiento de algoritmos de aprendizaje específicamente en redes neuronales recurrentes y derivadas (LSTM por ejemplo), las cuales consiguen resultados particularmente mejores ante problemas con datos del tipo de series de tiempo, como lo es este caso.

- Investigar sobre la posible expansión de la aplicación de este sistema a otros problemas relacionados a señales de Potenciales de Campo Local, por ejemplo en otras regiones del cerebro cuya funcionalidad podría relacionarse a eventos motores, entre otras.

Finalmente, se concluye con la principal meta de sentar un precedente en la extracción de información a partir de señales de LFPs utilizando algoritmos de aprendizaje automático, espacialmente en el contexto de Costa Rica y bajo el enfoque del sentido de orientación en el espacio del cerebro.

Bibliografía

Aguero, E. (2013). Implementation of active electrodes on a brain-computer interface and its application as p300 speller.

Arle, J. E. y Shils, J. L. (2011). *Essential Neuromodulation*. Academic Press, San Diego.

Barnett, Mark, y Larkman, P. (2007). The action potential. *Practical neurology*, 7:192–7.

Botchkarev, A. (2019). A new topology design of performance metrics to meassure errors in machine learning regression algorithms. *Interdisciplinary Journal of Information, Knowledge and Management*, 14.

Buzsaki, G. (2011). Hippocampus. *Scholarpedia*, 6(1):1468. revision #141265.

Calvo, D. (2015). Red neuronal convolucional cnn.

Cavelli, M. (2017). Actividad eléctrica cerebral durante el sueño.

Delgado, J. (2018). Implementación de un sistema de vocalización en un robot humanoide nao de palabras seleccionadas por medio de una interfaz cerebro computador, utilizando imaginiería motora.

Du, K.-L. y Swamy, M. (2014). *Recurrent Neural Networks*, páginas 337–353. ResearchGate GmbH.

Duda, R., Hart, P., y Stork, D. (2012). *Pattern Classification*. Wiley.

Gibson, S., Judy, J., y Markovíc, D. (2012). Spike sorting. *IEEE SIGNAL PROCESSING MAGAZINE*.

He, W., Zhao, Y., Tang, H., Sun, C., y Fu, W. (2016). A wireless bci and bmi system for wearable robots. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 46(7):936–946.

Islam, M., Mamun, K., y Deng, H. (2017). Decoding of human movements based on deep brain local field potentials using ensemble neural networks. *Hindawi, Computational Intelligence and Neuroscience*.

Jackson, A. y Hall, T. M. (2017). Decoding local field potentials for neural interfaces.

Jayabhavani, G. N., Raajan, N. R., y Abirami, J. (2013). A speller based mechanized messenger for smart phones using brain mobile interface (bmi). En *2013 International Conference on Emerging Trends in VLSI, Embedded System, Nano Electronics and Telecommunication System (ICEVENT)*, páginas 1–5.

Maling, N. y McIntyre, C. (2016). Chapter 5 - local field potential analysis for closed-loop neuromodulation. En El Hady, A., editor, *Closed Loop Neuroscience*, páginas 67–80. Academic Press, San Diego.

Mizuseki, K., Diba, K., y et al., E. P. (2014). Neurosharing: large-scale data sets (spike, lfp) recorded from the hippocampal-entorhinal system in behaving rats.

Mizuseki, K., Sirota, A., Pastalkova, E., Diba, K., y Buzsáki, G. (2013). Multiple single unit recordings from different rat hippocampal and entorhinal regions while the animals were performing multiple behavioral tasks.

Moroney, L. (2020). Sequences, time series and prediction lectures.

Naderi, M. A. y Mahdavi-Nasab, H. (2010). Analysis and classification of eeg signals using spectral analysis and recurrent neural networks. En *2010 17th Iranian Conference of Biomedical Engineering (ICBME)*, páginas 1–4.

NeuroNexus (2020). Buzsaki32.

Nielsen, M. (2015). *Neural Networks & Deep Learning*. Determination Press.

Oppenheim, A. V. y Schafer, R. W. (2009). *Discrete-Time Signal Processing*. Prentice Hall Press, USA, 3rd edición.

Perlmutter, Joel, y Mink, J. (2006). Deep brain stimulation. *Annual review of neuroscience*, 29:229–57.

Peyrache, A., Lacroix, M., Petersen, P., y Buzsáki, G. (2015). Internally-organized mechanisms of the head direction sense. *Nature Neuroscience*.

PRIS-Lab (2019). Publicaciones y antecedentes.

Python, S. F. (2017). 6. expressions: 6.6 binary arithmetic operations.

Ramos, N. (2019). Potencial de acción.

Rojas, J. C. (2019). Introduction to artificial intelligence lectures.

Salazar, A. (2018). Reconocimiento de patrones de activación de la corteza prefrontal durante la aritmética mental utilizando nirs en conjunto con eeg.

SciPy, C. (2021). Api reference: scipy.signal.decimate.

Shin, J., von Lühmanna, A., Blankertz, B., Kim, D.-W., Jeong, J., Hwang, H.-J., y Müller, K.-R. (2017). Open access dataset for eeg+nirs single-trial classification. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 25(10):1735–1745.

Sornmo, L. y Laguna, P. (2005). *Bioelectrical Signal Processing in Cardiac and Neurological Applications*. Academic Press series in biomedical engineering. Elsevier Science.

Souza, A. C., Koike, B. D., Pedrosa, R., Tort, A. B., Queiroz, C. M., y Ribeiro, S. (2018). Chapter 2 - recording day and night: Advice for new investigators in the sleep and memory field. En Manahan-Vaughan, D., editor, *Handbook of in Vivo Neural Plasticity Techniques*, volumen 28 of *Handbook of Behavioral Neuroscience*, páginas 43–62. Elsevier.

Taube, J. (2009). Head direction cells. *Scholarpedia*, 4(12):1787. revision #91349.

Teeters, J. y Sommer, F. (2019). Crcns - collaborative research in computational neuroscience - data sharing.

Tensorflow (2021). Tensorflow tutorial: Time series forecasting.

Vasilev, I., Slater, D., Spacagna, G., Roelants, P., y Zocca, V. (2019). *Python Deep Learning*, Capítulo 7. Packt Publishing Ltd.

A Documentación del código

UNIVERSITY OF COSTA RICA

ELECTRICAL ENGINEERING DEPARTMENT

GRADE THESIS

Local Field Potential (LFP) signals classifier's implementation using Neural Networks

CODE DOCUMENTATION

JOSÉ PABLO ÁVILA LÓPEZ (B30724)

JOSE.AVILALOPEZ@UCR.AC.CR

PROFESSOR: ESTEBAN ORTIZ

JULY, 2021

Contents

1 LFP Classification 1

2 Namespace Index 3

2.1 Packages . . . 3

3 File Index 5

3.1 File List . . . 5

4 Namespace Documentation 7

4.1 DataExploration Namespace Reference . . . 7

4.1.1 Function Documentation . . . 7

4.1.1.1 data_exploration() . . . 7

4.2 DatasetGenerator Namespace Reference . . . 8

4.2.1 Function Documentation . . . 8

4.2.1.1 data_set_generator() . . . 9

4.3 Environment Namespace Reference . . . 9

4.3.1 Function Documentation . . . 10

4.3.1.1 finish_test() . . . 10

4.3.1.2 init_environment() . . . 11

4.3.1.3 log_versions() . . . 11

4.3.1.4 print_box() . . . 11

4.3.1.5 print_header() . . . 12

4.3.1.6 print_parameters() . . . 12

4.3.1.7 print_text() . . . 12

4.3.1.8 step() . . . 13

4.3.2 Variable Documentation . . . 13

4.3.2.1 caller_file . . . 13

4.3.2.2 CAPTURES_FOLDER . . . 13

4.3.2.3 CURRENT_FOLDER . . . 13

4.3.2.4 debug . . . 13

4.3.2.5 LINE_LENGTH . . . 14

4.3.2.6 LOGS_FOLDER . . . 14

4.3.2.7 RESULTS_FOLDER . . . 14
4.3.2.8 START_TIME . . . 14
4.3.2.9 step_number . . . 14
4.3.2.10 versions_log . . . 14
4.3.2.11 VERSIONS_LOG_PATH . . . 14
4.3.2.12 yomchi_log . . . 14
4.3.2.13 YOMCHI_LOG_PATH . . . 15
4.4 models Namespace Reference . . . 15
4.4.1 Function Documentation . . . 15
4.4.1.1 cnn() . . . 15
4.4.1.2 compile_and_fit() . . . 16
4.4.1.3 lstm() . . . 16
4.4.1.4 mlp() . . . 17
4.5 Predictor Namespace Reference . . . 17
4.5.1 Function Documentation . . . 17
4.5.1.1 predictor() . . . 17
4.6 preprocessing Namespace Reference . . . 18
4.6.1 Function Documentation . . . 19
4.6.1.1 add_labels() . . . 19
4.6.1.2 angles_expansion() . . . 20
4.6.1.3 average_angles() . . . 20
4.6.1.4 channels_to_windows() . . . 20
4.6.1.5 clean_invalid_positional() . . . 21
4.6.1.6 downsample_lfps() . . . 22
4.6.1.7 interpolate_angles() . . . 22
4.6.1.8 load_angles_data() . . . 22
4.6.1.9 load_lfp_data() . . . 23
4.6.1.10 ndarray_to_dataframe() . . . 23
4.6.1.11 shortest_angle_interpolation() . . . 24
4.6.1.12 vectorized_sai() . . . 24
4.6.2 Variable Documentation . . . 24
4.6.2.1 ANGLES . . . 24
4.6.2.2 EC014_41_NUMBER_OF_CHANNELS . . . 25
4.6.2.3 LFP . . . 25
4.6.2.4 LFP_DATAMAX_SAMPLING_RATE . . . 25
4.6.2.5 LFP_NEURALYNX_SAMPLING_RATE . . . 25
4.6.2.6 PATH_TO_DATASETS . . . 25
4.6.2.7 POSITION_DATA_SAMPLING_RATE . . . 25
4.6.2.8 RAW_DATAMAX_SAMPLING_RATE . . . 25
4.6.2.9 RAW_NEURALYNX_SAMPLING_RATE . . . 26

4.7 visualization Namespace Reference . . . 26
4.7.1 Function Documentation . . . 26
4.7.1.1 store_figure() . . . 26
4.7.2 Variable Documentation . . . 26
4.7.2.1 FIG_FORMAT . . . 26

5 File Documentation **27**

5.1 /home/pabloav/Documents/Tesis_Lic/LFP-Classification/DataExploration/DataExploration.py File Reference . . . 27
5.1.1 Detailed Description . . . 27
5.2 DataExploration.py . . . 27
5.3 /home/pabloav/Documents/Tesis_Lic/LFP-Classification/DatasetGenerator/DatasetGenerator.py File Reference . . . 32
5.3.1 Detailed Description . . . 32
5.4 DatasetGenerator.py . . . 33
5.5 /home/pabloav/Documents/Tesis_Lic/LFP-Classification/Predictor/Predictor.py File Reference . . . 36
5.5.1 Detailed Description . . . 36
5.6 Predictor.py . . . 36
5.7 /home/pabloav/Documents/Tesis_Lic/LFP-Classification/README.md File Reference . . . 39
5.8 /home/pabloav/Documents/Tesis_Lic/LFP-Classification/README.md . . . 39
5.9 /home/pabloav/Documents/Tesis_Lic/LFP-Classification/Yomchi/Environment.py File Reference . . . 40
5.9.1 Detailed Description . . . 41
5.10 Environment.py . . . 42
5.11 /home/pabloav/Documents/Tesis_Lic/LFP-Classification/Yomchi/models.py File Reference . . . 45
5.11.1 Detailed Description . . . 45
5.12 models.py . . . 46
5.13 /home/pabloav/Documents/Tesis_Lic/LFP-Classification/Yomchi/preprocessing.py File Reference . . . 48
5.13.1 Detailed Description . . . 49
5.14 preprocessing.py . . . 49
5.15 /home/pabloav/Documents/Tesis_Lic/LFP-Classification/Yomchi/visualization.py File Reference . . . 54
5.15.1 Detailed Description . . . 55
5.16 visualization.py . . . 55

Index **57**

Chapter 1

LFP Classification

Classifier of Local Field Potential signals using Deep Neural Networks for head angular direction prediction

Prerequisites

The current project was developed using **Ubuntu 18.04.4 LTS**. The following are the required packages and python libraries. It is recommended to use the listed versions, nevertheless is worth to try the newest available version of them.

Packages:

```
1. Python (3.6)
```

Optional:

```
1. Latex
2. Doxygen
3. Doxypy
4. Make
5. Git
6. Graphviz
```

Python libraries:

Package	Version
pip3	20.1
numpy	1.18.4
matplotlib	3.2.1
tensorflow	2.2.0
keras	2.3.1
sklearn	0.0
pandas	1.0.3
scikit-learn	0.23.0
seaborn	0.10.1

Install

Check the installed Linux version using: `lsb_release -a`. Check the installed version of the package using: `<package> --<version>` Check the installed python libraries and their version using: `pip list`

Packages:

```
sudo apt-get install python3.6 texlive-latex-extra doxygen doxygen-gui make git graphviz
```

Pip:

This is necessary to install the rest of python libraries

```
curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py
python3.6 get-pip.py
```

Python libraries:

If getting the latest available version:

```
pip3 install --upgrade <package>
```

To download an specific version:

```
pip3 install <package>==<version>
```

Documentation

- You can start reading the documentation in the web veiw here LFP-Classification-Documentation-Page.
- To read it as PDF run `make` under *docs/* and open the generated *latex/refman.pdf* file.

Author

Pablo Avila - PabloAvLo

License

This project is licensed under the MIT License - see the LICENSE.md file for details

Acknowledgments

- The present code is part of my grade thesis in Electrical Engineering at the University of Costa Rica.
- The main objective is to prove the hypothesis that based on Local Field Potential measurments from the Entorhinal Cortex of the Hippocampus of rats is possible to extract angular information of the head direction of the subject.
- The dataset used in this experiment is: BuszakiLab HC-3 Dataset.
- In order to do so, machine learning algorithms will be applied to the data, such as Feedforward Neural Network, RNNs and CNNs.

Chapter 2

Namespace Index

2.1 Packages

Here are the packages with brief descriptions (if available):

DataExploration 7
DatasetGenerator 8
Environment 9
models 15
Predictor 17
preprocessing 18
visualization 26

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

/home/pabloav/Documents/Tesis_Lic/LFP-Classification/DataExploration/DataExploration.py 27
/home/pabloav/Documents/Tesis_Lic/LFP-Classification/DatasetGenerator/DatasetGenerator.py 32
/home/pabloav/Documents/Tesis_Lic/LFP-Classification/Predictor/Predictor.py 36
/home/pabloav/Documents/Tesis_Lic/LFP-Classification/Yomchi/Environment.py 40
/home/pabloav/Documents/Tesis_Lic/LFP-Classification/Yomchi/models.py 45
/home/pabloav/Documents/Tesis_Lic/LFP-Classification/Yomchi/preprocessing.py 48
/home/pabloav/Documents/Tesis_Lic/LFP-Classification/Yomchi/visualization.py 54

Chapter 4

Namespace Documentation

4.1 DataExploration Namespace Reference

Functions

- def data_exploration ()

4.1.1 Function Documentation

4.1.1.1 data_exploration()

```
def DataExploration.data_exploration ( )
```

Data Exploration:

Setup

- Initialize Environment.
- Initialize Tensorflow session.
- Set seed for Numpy and Tensorflow
- Specify run configuration parameters.
- Specify dataset generation parameters.

Procedure

- **Step 1:**
  - Import LFP data.
  - Plot channels 0 and 97.
  - Plot the first 10 seconds of LFP data from selected channel at original sampling rate (1250Hz).
- **Step 2:**
  - Import angles data.

  - Plot angles data.
  - Plot 20 seconds at minute 6 of angles data at original sampling rate (39.06Hz).

- **Step 3:**
  - If the chosen synchronization method is to downsample the LFP data:
    - Downsample LFP data to match Angles sampling rate.
    - Plot channels 0 and 97 downsampled.
    - Plot the first 10 seconds of LFP data from selected channel at position's sampling rate (39.06Hz).
  - Else, the chosen synchronization method is to upsample the Angles data:
    - Fill angles gaps to match the LFP sampling rate.
    - Plot angles data in the new sampling rate.

- **Step 4:**
  - Interpolate angles data using a 'interpolation' approach.
  - Plot angles data after interpolation.
  - Plot 20 seconds at minute 6 of angles data at LFP's sampling rate (39.06Hz). Interpolated with the specified method.

- **Step 5:**
  - Label data by concatenating LFPs and interpolated Angles in a single 2D-array.
  - Plot LFP and angles data.
  - Print an angles data window where the first LED is unsynchronized and then the second LED is lost too.

- **Step 6:**
  - Clean the labeled dataset from invalid values.
  - Print Bar plot for labeled angles.

- **Step 7:**
  - Count number of NaNs at the beginning and at the end of the data.
  - Print some NaN counts in different dataset stages.

- **Step 8:** Plot clean LFP data from channels 0 and interpolated angles data.

- **Step 9:** Plot Boxplot of all channels in two figures, from 0-49 and from 50-98.

- **Step 10:**
  - Plot Boxplot of interpolated angles.
  - Plot Boxplot of interpolated angles vs channel 0

- **Finish Test and Exit:** Save logs, captures and results.

Definition at line 100 of file DataExploration.py.

4.2 DatasetGenerator Namespace Reference

Functions

- def data_set_generator ()

4.2.1 Function Documentation

4.2.1.1 data_set_generator()

```
def DatasetGenerator.data_set_generator ( )
```

Dataset Generator:

Setup

- Initialize Environment.
- Initialize Tensorflow session.
- Set seed for Numpy and Tensorflow
- Specify run configuration parameters.
- Specify dataset generation parameters.

Procedure

- **Step 1:** Import LFP data.
- **Step 2:** Import angles data.
- **Step 3:**
  - Downsample LFP data to match the Angles sampling rate or
  - Fill angles gaps to match the LFP sampling rate.
- **Step 4:** Label data by concatenating LFPs and Angles in a single 2D-array.
- **Step 5:** Clean the labeled dataframe from -1 values, which represent the wrongly acquired positional samples.
- **Step 6:** Interpolate angles data using an 'interpolation' approach.
- **Step 7:** Plotting Angles and the selected channel of the LFPs after cleaning and interpolation.
- **Step 8:**
  - Get preferred angle according to LFPs channel.
  - Plotting the sum of LFPs per angle from 0 to 359 degrees.
- **Step 9:** Get largest subset of data.
- **Step 10:** Split the subset into training, validation and testing set.
- **Step 11:** Save the dataset to a pickle file.
- **Finish Test and Exit:** Save logs, captures, results and the generated dataset into a .pickle file.

Definition at line 64 of file DatasetGenerator.py.

4.3 Environment Namespace Reference

Functions

- def init_environment (print_the_header=False, enable_debug=False)

  Initialize the directories and files to log the results.
- def print_text (text)

  Print a passed text and save it in the yomchi_log.txt.

- def print_box (text)

  Print a passed text in a box and save it in the yomchi_log.txt.
- def step (description=None, new_step_number=0)

  Print the current step number with a brief description.
- def print_parameters (params_dictionary)

  Print a list of parameters.
- def log_versions ()

  Stores the environment versions including packages, libraries and OS.
- def print_header ()

  Print a header in the console output and log files.
- def finish_test (rename_results_folder=None)

  Safely finish the test run and log some final information.

Variables

- START_TIME = datetime.datetime.now()
- CURRENT_FOLDER = os.getcwd()
- string RESULTS_FOLDER = CURRENT_FOLDER + "/Results-" + START_TIME.strftime("%Y-%m-%d_%H-%M") + "/"
- string LOGS_FOLDER = RESULTS_FOLDER + "Logs/"
- string CAPTURES_FOLDER = RESULTS_FOLDER + "Captures/"
- string VERSIONS_LOG_PATH = LOGS_FOLDER + "versions_log.txt"
- string YOMCHI_LOG_PATH = LOGS_FOLDER + "yomchi_log.txt"
- int LINE_LENGTH = 100
- int step_number = 1
- versions_log = None
- yomchi_log = None
- caller_file = None
- bool debug = False

4.3.1 Function Documentation

4.3.1.1 finish_test()

```
def Environment.finish_test (
              rename_results_folder = None )
```

Safely finish the test run and log some final information.

Parameters

rename_results_folder	Optional name for the results folder.

Returns

None

Definition at line 248 of file Environment.py.

References print_text().

4.3.1.2 init_environment()

```
def Environment.init_environment (
              print_the_header = False,
              enable_debug = False )
```

Initialize the directories and files to log the results.

Parameters

print_the_header	If True, print the header
enable_debug	If True, run some code to help debugging

Definition at line 49 of file Environment.py.

References log_versions(), and print_header().

4.3.1.3 log_versions()

```
def Environment.log_versions ( )
```

Stores the environment versions including packages, libraries and OS.

Returns

None

Definition at line 177 of file Environment.py.

Referenced by init_environment().

4.3.1.4 print_box()

```
def Environment.print_box (
              text )
```

Print a passed text in a box and save it in the yomchi_log.txt.

Parameters

text	Text to print

Returns

None

Definition at line 96 of file Environment.py.

References print_text().

Referenced by step().

4.3.1.5 print_header()

```
def Environment.print_header ( )
```

Print a header in the console output and log files.

Returns

None

Definition at line 223 of file Environment.py.

References print_text().

Referenced by init_environment().

4.3.1.6 print_parameters()

```
def Environment.print_parameters (
              params_dictionary )
```

Print a list of parameters.

Parameters

params_dictionary	Dictionary of parameters {name: value} to print.

Returns

None

Definition at line 149 of file Environment.py.

References print_text().

4.3.1.7 print_text()

```
def Environment.print_text (
              text )
```

Print a passed text and save it in the yomchi_log.txt.

Parameters

text	Text to print

Returns

None

Definition at line 83 of file Environment.py.

Referenced by finish_test(), print_box(), print_header(), and print_parameters().

4.3.1.8 step()

```
def Environment.step (
              description = None,
              new_step_number = 0 )
```

Print the current step number with a brief description.

Parameters

description	Description of the step.
new_step_number	If defined, reset the steps numeration to the passed value.

Returns

None

Definition at line 126 of file Environment.py.

References print_box().

4.3.2 Variable Documentation

4.3.2.1 caller_file

```
Environment.caller_file = None
```

Definition at line 39 of file Environment.py.

4.3.2.2 CAPTURES_FOLDER

```
string Environment.CAPTURES_FOLDER = RESULTS_FOLDER + "Captures/"
```

Definition at line 30 of file Environment.py.

4.3.2.3 CURRENT_FOLDER

```
Environment.CURRENT_FOLDER = os.getcwd()
```

Definition at line 27 of file Environment.py.

4.3.2.4 debug

```
bool Environment.debug = False
```

Definition at line 40 of file Environment.py.

4.3.2.5 LINE_LENGTH

```
int Environment.LINE_LENGTH = 100
```

Definition at line 33 of file Environment.py.

4.3.2.6 LOGS_FOLDER

```
string Environment.LOGS_FOLDER = RESULTS_FOLDER + "Logs/"
```

Definition at line 29 of file Environment.py.

4.3.2.7 RESULTS_FOLDER

```
string Environment.RESULTS_FOLDER = CURRENT_FOLDER + "/Results-" + START_TIME.strftime("%Y-%m-%d↩
_%H-%M") + "/"
```

Definition at line 28 of file Environment.py.

4.3.2.8 START_TIME

```
Environment.START_TIME = datetime.datetime.now()
```

Definition at line 26 of file Environment.py.

4.3.2.9 step_number

```
int Environment.step_number = 1
```

Definition at line 36 of file Environment.py.

4.3.2.10 versions_log

```
Environment.versions_log = None
```

Definition at line 37 of file Environment.py.

4.3.2.11 VERSIONS_LOG_PATH

```
string Environment.VERSIONS_LOG_PATH = LOGS_FOLDER + "versions_log.txt"
```

Definition at line 31 of file Environment.py.

4.3.2.12 yomchi_log

```
Environment.yomchi_log = None
```

Definition at line 38 of file Environment.py.

4.3.2.13 YOMCHI_LOG_PATH

```
string Environment.YOMCHI_LOG_PATH = LOGS_FOLDER + "yomchi_log.txt"
```

Definition at line 32 of file Environment.py.

4.4 models Namespace Reference

Functions

- def mlp (layers, units_per_layer, dropout=None)

  Defines a classical neural network sometimes called: Multi-Layer Perceptron.
- def cnn (inputs, units_per_layer, dropout)

  Creates a model of a Convolutional Neural Network with a Conv1D as the input layer, one dense as the only hidden layer and another dense with only 1 neuron as the output layer.
- def lstm (units_per_layer, dropout)

  Creates a model of a Long Short-Term Memory Neural Network as the input layer, one dense with only 1 neuron as the output layer.
- def compile_and_fit (model, train, val, epochs=20, patience=2)

  Fits the provided training data in the specified model and train's it.

4.4.1 Function Documentation

4.4.1.1 cnn()

```
def models.cnn (
              inputs,
              units_per_layer,
              dropout )
```

Creates a model of a Convolutional Neural Network with a Conv1D as the input layer, one dense as the only hidden layer and another dense with only 1 neuron as the output layer.

The first two layer has ReLU activation and the last one uses a linear activation function.S

Parameters

inputs	Number of inputs of the network. It is used as the kernel size with the intention that the 1D convolutional layer outputs a single value throughout the specified number of filters.
units_per_layer	Specified the number of neurons of the hidden dense layer, which has to match with the number of filters that will output a result in the 1D convolutional input layer.
dropout	A value between 0 and 1 of neuron's results to discard of the training. This regularization method will be added to the model as a layer the output layer.

Returns

conv_model: The model of the CNN created for later usage as the predictor.

Definition at line 66 of file models.py.

Referenced by Predictor.predictor().

4.4.1.2 compile_and_fit()

```
def models.compile_and_fit (
              model,
              train,
              val,
              epochs = 20,
              patience = 2 )
```

Fits the provided training data in the specified model and train's it.

The validation data is used to compare the performance of the model against unknown data. MSE is used as the cost function to train the model and MAE as the performance evaluation metric.

Parameters

model	Model to train. It can be a MLP, CNN or LSTM, among others.
train	The dataset for training the model. Must have the shape: (batch, time, features)
val	The dataset for validating the model. Must have the shape: (batch, time, features)
epochs	Number of iteration over the entire set of data (all the batches).
patience	Number of epochs to wait for improvement in the metrics. If there is no notorious improvement in the performance of the validation set after the 'patience' epochs, the training will stop at this point.

Returns

history: The results of the training.

Definition at line 116 of file models.py.

Referenced by Predictor.predictor().

4.4.1.3 lstm()

```
def models.lstm (
              units_per_layer,
              dropout )
```

Creates a model of a Long Short-Term Memory Neural Network as the input layer, one dense with only 1 neuron as the output layer.

The activation functions of the LSTM layer are the regular ones for each of it's gates.

Parameters

units_per_layer	Number of inputs of the network.
dropout	A value between 0 and 1 of neuron's results to discard of the training. If provided,this regularization method will be to the LSTM Layer.

Returns

lstm_model: The model of the LSTM created for later usage as the predictor.

Definition at line 89 of file models.py.

Referenced by Predictor.predictor().

4.4.1.4 mlp()

```
def models.mlp (
              layers,
              units_per_layer,
              dropout = None )
```

Defines a classical neural network sometimes called: Multi-Layer Perceptron.

It is Feedforward and fully-connected, with the specified parameters. The activation function of the hidden layers is ReLU, and the activation of the output is linear.

Parameters

layers	Number of hidden layers (besides the input and outputs ones).
units_per_layer	Number of neurons per layer. Applies to all layers except for the last one which has only 1: the predicted angle.
dropout	A value between 0 and 1 of neuron's results to discard of the training. This regularization method will be added to the model. One after the input layer and one before the output layer.

Returns

model: The model of the MLP created for later usage as the predictor.

Definition at line 32 of file models.py.

Referenced by Predictor.predictor().

4.5 Predictor Namespace Reference

Functions

- def predictor ()

4.5.1 Function Documentation

4.5.1.1 predictor()

```
def Predictor.predictor ( )
```

Predictor

Experiment Setup

- Initialize Environment
- Initialize Tensorflow session.
- Set seed for Numpy and Tensorflow
- Specify run configuration parameters.
- Specify run configuration parameters.
- Specify dataset generation parameters.
- Specify model's training parameters.

Procedure

- **Step 1:** Import training, validation and test datasets of LFPs as inputs and Angles as labels.
- **Step 2:** Convert data to windowed series.
- **Step 3:** Create the model with the specified parameters
- **Step 4:** Train the model with the training and validation data.
- **Step 5:** Plotting the original angular data vs the predictions for different batch sizes.
- **Step 6:** Plotting the original angular data vs the predictions of all testing data.
- **Step 7:** Plotting the historic of the training vs the validation error values.
- **Finish Test and Exit:** Save logs, captures and results.

Definition at line 55 of file Predictor.py.

References models.cnn(), models.compile_and_fit(), models.lstm(), and models.mlp().

4.6 preprocessing Namespace Reference

Functions

- def load_lfp_data (file=LFP[771], channels=EC014_41_NUMBER_OF_CHANNELS)

  Loads the LFP signals from a .eeg file (LFP Data only f < 625Hz) or a .dat file (LFP Data + Spikes).
- def load_angles_data (file=ANGLES[771], degrees=True)

  Loads the animal position data from a .whl file which contain 2 (x, y) pairs, one for each LED.
- def downsample_lfps (lfp_data, orig_rate, new_rate)

  Downsample the LFP signal data after applying an anti-aliasing filter.
- def angles_expansion (angles_data, orig_rate, new_rate)

  Fill angular data with 'NaN' values to match an expected sampling rate.
- def shortest_angle_interpolation (start, end, amount)

  This interpolation method considers the 'start' and 'end' as angles in a circumference where the objective is to find the smallest arch between two angles.
- def vectorized_sai (angles_data)

  Replace 'NaN' values between valid values (interpolation) in angular data with an interpolated value using the shortest angle interpolation.
- def interpolate_angles (angles_data, method="Shortest Angle")

  Replace 'NaN' values in angular data with an interpolated value using a given method.
- def add_labels (lfps, angles, round_labels, start=0, offset=30)

Add an additional column to the LFP signals matrix with the angular data used as the labels.

- def clean_invalid_positional (labeled_dataset, is_padded=True)

  Clean the data rows which have '-1' values as labels (angles) from the the data and their LFPs associated in each channel.

- def ndarray_to_dataframe (dataset, rate)

  Converts an n-D Numpy array to a Pandas Dataframe.

- def channels_to_windows (series, channel, window_size, batch_size, shuffle_buffer=None, offset=1)

  Receives a numpy array containing the time series of LFP signals of n channels and returns the same data, separated in windows.

- def average_angles (angles, window_size)

  Receives a numpy array containing the time series of the angles and returns the an set of windows with the average of the 'window_size' angles in each window.

Variables

- PATH_TO_DATASETS = os.path.join(Env.CURRENT_FOLDER, "../Datasets/")
- dictionary LFP
- dictionary ANGLES
- int RAW_DATAMAX_SAMPLING_RATE = 20000
- int RAW_NEURALYNX_SAMPLING_RATE = 32552
- int LFP_DATAMAX_SAMPLING_RATE = 1250
- int LFP_NEURALYNX_SAMPLING_RATE = 1252
- float POSITION_DATA_SAMPLING_RATE = 39.06
- int EC014_41_NUMBER_OF_CHANNELS = 99

4.6.1 Function Documentation

4.6.1.1 add_labels()

```
def preprocessing.add_labels (
              lfps,
              angles,
              round_labels,
              start = 0,
              offset = 30 )
```

Add an additional column to the LFP signals matrix with the angular data used as the labels.

Parameters

lfps	Matrix [n x numChannels] with the LFP signals used as the preliminary features of the data.
angles	Array with the angles data extracted from the positions used as the labels of the data.
round_labels	Boolean, if true the labels are rounded to angles multiples of 'offset' starting from 'start'
start	Angle in [0°, 360°[used as first label.
offset	Offset in [1°, 360°[between labels starting from 'start' angle.

Returns

labeled_data: Matrix with the labeled data [n x lfps[numChannels], angles].

Definition at line 239 of file preprocessing.py.

4.6.1.2 angles_expansion()

```
def preprocessing.angles_expansion (
              angles_data,
              orig_rate,
              new_rate )
```

Fill angular data with 'NaN' values to match an expected sampling rate.

Usually the position data is acquired at a lower sampling rate than the LFP signals.

Assuming that the acquisition of data started and stopped at the same time, then no data has to be added after the last sample.

Parameters

angles_data	Array with the angles data extracted from the animal positions.
orig_rate	Sampling rate originally used to acquire the data.
new_rate	New sampling rate of the data. The gaps are filled with 'NaN'.

Returns

upsampled_data: Original data filled with 'NaN' to match the new sampling rate.

Definition at line 135 of file preprocessing.py.

4.6.1.3 average_angles()

```
def preprocessing.average_angles (
              angles,
              window_size )
```

Receives a numpy array containing the time series of the angles and returns the an set of windows with the average of the 'window_size' angles in each window.

Each window is 1 element shifted from the previous window.

Parameters

angles	Numpy Array with the Angles data to use as the labels.
window_size	Size of the windows in which the data are being split.

Returns

average_angles: Averaged Angles data separated in windows.

Definition at line 423 of file preprocessing.py.

4.6.1.4 channels_to_windows()

```
def preprocessing.channels_to_windows (
              series,
```

```
                channel,
                window_size,
                batch_size,
                shuffle_buffer = None,
                offset = 1 )
```

Receives a numpy array containing the time series of LFP signals of n channels and returns the same data, separated in windows.

Parameters

series	Numpy Array with the LFP data of the n channels.
channel	Channel to use.
window_size	Size of the windows in which the data are being split.
batch_size	Number of pairs data-labels to group as a batch
shuffle_buffer	Number of windows to shuffle at the same time.
offset	Number of samples to shift between windows.

Returns

windowed_ds: LFP data of the selected channel separated in windows.

Definition at line 378 of file preprocessing.py.

4.6.1.5 clean_invalid_positional()

```
def preprocessing.clean_invalid_positional (
                labeled_dataset,
                is_padded = True )
```

Clean the data rows which have '-1' values as labels (angles) from the the data and their LFPs associated in each channel.

Plus the following 31 rows with NaN as angle value in case of padded data.

The positional data taken from the LEDs placed in the rat have discontinuities where one or both LEDs are lost, making them invalid. Hence '-1' values are used instead to denote invalid position data and are meant to be removed from the data since they are not representative labels.

Parameters

labeled_dataset	Matrix [n x (numChannels +1)] with the LFP signals used as the preliminary features of the data and the angles data extracted from the positions used as the labels of the data.
is_padded	If True, manage the input labeled dataset as a padded array of angles, or a downsampled LFP set otherwise.

Returns

clean_dataset: Input data without invalid positional values.

Definition at line 287 of file preprocessing.py.

4.6.1.6 downsample_lfps()

```
def preprocessing.downsample_lfps (
              lfp_data,
              orig_rate,
              new_rate )
```

Downsample the LFP signal data after applying an anti-aliasing filter.

An order 8 Chebyshev type I filter is used. Usually the LFP signals are acquired at a higher sampling rate than the position data.

Note

: This method assumes that the reason of frequencies is 32 to compute the decimation.

Parameters

lfp_data	Matrix [n x numChannels] with the LFP signals
orig_rate	Sampling rate originally used to acquire the data.
new_rate	New sampling rate of the data.

Returns

resampled_data: Original data downsampled to the new rate.

Definition at line 108 of file preprocessing.py.

4.6.1.7 interpolate_angles()

```
def preprocessing.interpolate_angles (
              angles_data,
              method = "Shortest Angle" )
```

Replace 'NaN' values in angular data with an interpolated value using a given method.

Parameters

angles_data	Array with the angles data extracted from the animal positions.
method	Interpolation method to fill the gaps in the data. The optional methods available are the supported by pandas.DataFrame.interpolate function, which are: 'linear', 'quadratic', 'cubic', 'polynomial', among others.

Returns

interpolated_angles: Array with the angles data interpolated using the given method.

Definition at line 211 of file preprocessing.py.

References vectorized_sai().

4.6.1.8 load_angles_data()

```
def preprocessing.load_angles_data (
```

```
                file = ANGLES[771],
                degrees = True )
```

Loads the animal position data from a .whl file which contain 2 (x, y) pairs, one for each LED.

If any position value equals '-1' then it's replaced with 'NaN' instead.

Parameters

file	Path to the file containing the animal LED's position information
degrees	If this flag is set, then the angles are returned in degrees from [0, 360[, or radians otherwise.

Returns

angles: Array with the angles in radians extracted from the positions. The angles are given as float16 values calculated as arctan(y2 - y1 /x2 - x1). Unless the denominator is 0, in that case '0' is returned for that element.

Definition at line 71 of file preprocessing.py.

4.6.1.9 load_lfp_data()

```
def preprocessing.load_lfp_data (
                file = LFP[771],
                channels = EC014_41_NUMBER_OF_CHANNELS )
```

Loads the LFP signals from a .eeg file (LFP Data only f $<$ 625Hz) or a .dat file (LFP Data + Spikes).

Parameters

file	Path to the file containing the animal LFP data.
channels	Number of recorded channels in LFP signals file.

Returns

lfp: Array (n x channels) with the data. With the columns being the channels and the rows the a different time step.

Definition at line 48 of file preprocessing.py.

4.6.1.10 ndarray_to_dataframe()

```
def preprocessing.ndarray_to_dataframe (
                dataset,
                rate )
```

Converts an n-D Numpy array to a Pandas Dataframe.

Parameters

dataset	Matrix [n x (numChannels +1)] with the LFP signals used as the preliminary features of the data and the angles data extracted from the positions used as the labels of the data.
rate	

Returns

dataframe: Pandas data frame with Channels 0-99 and Angles as columns names, and the timestamp as indexes calculated as 1/rate ∗ 1e6 to get the time step of the acquisition in microseconds.

Definition at line 352 of file preprocessing.py.

4.6.1.11 shortest_angle_interpolation()

```
def preprocessing.shortest_angle_interpolation (
              start,
              end,
              amount )
```

This interpolation method considers the 'start' and 'end' as angles in a circumference where the objective is to find the smallest arch between two angles.

param start: Start angle with values in the range of [0 to 360[param end: Final angle with values in the range of [0 to 360[param amount: Value between [0, 1] which determines how close the interpolated angle will be placed from the Start angle (0) or from the Final angle (1), being 0.5 the middle. return interpolated_angle: Interpolated angle between 'start' and 'end'.

Definition at line 160 of file preprocessing.py.

Referenced by vectorized_sai().

4.6.1.12 vectorized_sai()

```
def preprocessing.vectorized_sai (
              angles_data )
```

Replace 'NaN' values between valid values (interpolation) in angular data with an interpolated value using the shortest angle interpolation.

param angles_data: Array with the angles data extracted from the animal positions. return interpolated_angles: Array with the angles data interpolated using the Shortest Angle Interpolation

Definition at line 173 of file preprocessing.py.

References shortest_angle_interpolation().

Referenced by interpolate_angles().

4.6.2 Variable Documentation

4.6.2.1 ANGLES

```
dictionary preprocessing.ANGLES
```

Initial value:

```
00001 =  {765: PATH_TO_DATASETS + "ec014.765.whl",
00002           771: PATH_TO_DATASETS + "ec014.771.whl"}
```

Definition at line 28 of file preprocessing.py.

4.6.2.2 EC014_41_NUMBER_OF_CHANNELS

```
int preprocessing.EC014_41_NUMBER_OF_CHANNELS = 99
```

Definition at line 37 of file preprocessing.py.

4.6.2.3 LFP

```
dictionary preprocessing.LFP
```

Initial value:

```
00001 =  {765: PATH_TO_DATASETS + "ec014.765.eeg",
00002        771: PATH_TO_DATASETS + "ec014.771.eeg"}
```

Definition at line 26 of file preprocessing.py.

4.6.2.4 LFP_DATAMAX_SAMPLING_RATE

```
int preprocessing.LFP_DATAMAX_SAMPLING_RATE = 1250
```

Definition at line 34 of file preprocessing.py.

4.6.2.5 LFP_NEURALYNX_SAMPLING_RATE

```
int preprocessing.LFP_NEURALYNX_SAMPLING_RATE = 1252
```

Definition at line 35 of file preprocessing.py.

4.6.2.6 PATH_TO_DATASETS

```
preprocessing.PATH_TO_DATASETS = os.path.join(Env.CURRENT_FOLDER, "../Datasets/")
```

Definition at line 25 of file preprocessing.py.

4.6.2.7 POSITION_DATA_SAMPLING_RATE

```
float preprocessing.POSITION_DATA_SAMPLING_RATE = 39.06
```

Definition at line 36 of file preprocessing.py.

4.6.2.8 RAW_DATAMAX_SAMPLING_RATE

```
int preprocessing.RAW_DATAMAX_SAMPLING_RATE = 20000
```

Definition at line 32 of file preprocessing.py.

4.6.2.9 RAW_NEURALYNX_SAMPLING_RATE

```
int preprocessing.RAW_NEURALYNX_SAMPLING_RATE = 32552
```

Definition at line 33 of file preprocessing.py.

4.7 visualization Namespace Reference

Functions

- def store_figure (fig_name, show=False)

  Stores a figure.

Variables

- string FIG_FORMAT = 'png'

4.7.1 Function Documentation

4.7.1.1 store_figure()

```
def visualization.store_figure (
              fig_name,
              show = False )
```

Stores a figure.

Parameters

fig_name	Name of the figure to store.
show	Displays the figure when ready. Warning: Stalls execution until closing it.

Returns

None

Definition at line 40 of file visualization.py.

4.7.2 Variable Documentation

4.7.2.1 FIG_FORMAT

```
string visualization.FIG_FORMAT = 'png'
```

Definition at line 21 of file visualization.py.

Chapter 5

File Documentation

5.1 /home/pabloav/Documents/Tesis_Lic/LFP-Classification/DataExploration/Data↩ Exploration.py File Reference

Namespaces

- DataExploration

Functions

- def DataExploration.data_exploration ()

5.1.1 Detailed Description

Author

Pablo Avila [B30724] jose.avilalopez@ucr.ac.cr

Copyright

MIT License

Date

July, 2021

Script for explore the input data by manipulating, printing and Plot the information in different ways.

Definition in file DataExploration.py.

5.2 DataExploration.py

```
00001 # ##############################################################################
00002 #                   University of Costa Rica
00003 #               Electrical Engineering Department
00004 #                         Grade Thesis
00005 # ##############################################################################
00006
00007 """
00008 @file DataExploration.py
00009 @author Pablo Avila [B30724] jose.avilalopez@ucr.ac.cr
00010 @copyright MIT License
```

```
00011 @date July, 2021
00012 @details Script for explore the input data by manipulating, printing and Plot the information in different
       ways.
00013 """
00014
00015 import Yomchi.Environment as Env
00016 import Yomchi.preprocessing as data
00017 import Yomchi.visualization as ui
00018
00019 import matplotlib.pyplot as plt
00020 import numpy as np
00021 import seaborn as sns
00022 import pandas as pd
00023 import tensorflow as tf
00024
00025
00026 def data_exploration():
00027     """
00028     @details
00029     <h1> Data Exploration:</h1>
00030     <h2> Setup </h2>
00031     <ul>
00032         <li> Initialize Environment.
00033         <li> Initialize Tensorflow session.
00034         <li> Set seed for Numpy and Tensorflow
00035         <li> Specify run configuration parameters.
00036         <li> Specify dataset generation parameters.
00037     </ul>
00038     <h2> Procedure </h2>
00039     <ul>
00040     <li> <b>Step 1:</b>
00041         <ul>
00042         <li> Import LFP data.
00043         <li> Plot channels 0 and 97.
00044         <li> Plot the first 10 seconds of LFP data from selected channel at original sampling rate
       (1250Hz).
00045         </ul>
00046     <li> <b>Step 2:</b>
00047         <ul>
00048         <li> Import angles data.
00049         <li> Plot angles data.
00050         <li> Plot 20 seconds at minute 6 of angles data at original sampling rate (39.06Hz).
00051         </ul>
00052     <li> <b>Step 3:</b>
00053         <ul>
00054         <li> If the chosen synchronization method is to downsample the LFP data:
00055             <ul>
00056             <li> Downsample LFP data to match Angles sampling rate.
00057             <li> Plot channels 0 and 97 downsampled.
00058             <li> Plot the first 10 seconds of LFP data from selected channel at position's sampling rate
       (39.06Hz).
00059             </ul>
00060         <li> Else, the chosen synchronization method is to upsample the Angles data:
00061             <ul>
00062             <li> Fill angles gaps to match the LFP sampling rate.
00063             <li> Plot angles data in the new sampling rate.
00064             </ul>
00065         </ul>
00066     <li> <b>Step 4:</b>
00067         <ul>
00068         <li> Interpolate angles data using a 'interpolation' approach.
00069         <li> Plot angles data after interpolation.
00070         <li> Plot 20 seconds at minute 6 of angles data at LFP's sampling rate (39.06Hz). Interpolated with
       the
00071         specified method.
00072         </ul>
00073     <li> <b>Step 5:</b>
00074         <ul>
00075         <li> Label data by concatenating LFPs and interpolated Angles in a single 2D-array.
00076         <li> Plot LFP and angles data.
00077         <li> Print an angles data window where the first LED is unsynchronized and then the second LED is
       lost too.
00078         </ul>
00079     <li> <b>Step 6:</b>
00080         <ul>
00081         <li> Clean the labeled dataset from invalid values.
00082         <li> Print Bar plot for labeled angles.
00083         </ul>
00084     <li> <b>Step 7:</b>
00085         <ul>
00086         <li> Count number of NaNs at the beginning and at the end of the data.
00087         <li> Print some NaN counts in different dataset stages.
00088         </ul>
00089     <li> <b>Step 8:</b> Plot clean LFP data from channels 0 and interpolated angles data.
00090     <li> <b>Step 9:</b> Plot Boxplot of all channels in two figures, from 0-49 and from 50-98.
00091     <li> <b>Step 10:</b>
00092         <ul>
```

```
00093         <li> Plot Boxplot of interpolated angles.
00094         <li> Plot Boxplot of interpolated angles vs channel 0
00095         </ul>
00096     <li> <b>Finish Test and Exit:</b> Save logs, captures and results.
00097     </ul>
00098     """
00099
00100     # ------- SETUP -------
00101     Env.init_environment(True)
00102
00103     tf.keras.backend.clear_session()
00104     tf.random.set_seed(51)
00105     np.random.seed(51)
00106
00107     # Run configuration parameters
00108     PLOT = True
00109
00110     # Session and methods Parameters
00111     session = 765  # 771 or 765
00112
00113     interpolation = "quadratic"  # "linear" "quadratic" "cubic" "nearest" "Shortest Angle"
00114     sync_method = "Downsample LFPs"  # "Upsample Angles" "Downsample LFPs"
00115
00116     # Data Properties
00117     num_channels = data.EC014_41_NUMBER_OF_CHANNELS
00118     rate_used = data.POSITION_DATA_SAMPLING_RATE
00119     if sync_method == "Upsample Angles":
00120         rate_used = data.LFP_DATAMAX_SAMPLING_RATE
00121
00122     # Windowing properties
00123     lfp_channel = 70
00124     round_angles = False
00125     base_angle = 0  # Unused if round_angles = False
00126     offset_between_angles = 30  # Unused if round_angles = False
00127
00128     extra = {"Round angles to get discrete label": round_angles}
00129     if round_angles:
00130         extra.update({"Angle labels starting from": str(base_angle) + "°",
00131                       "until 360° in steps of": str(offset_between_angles) + "°"})
00132
00133     parameters_dictionary = {"Recording Session Number": str(session),
00134                              "Interpolation Method": interpolation.title(),
00135                              "Synchronization Method": sync_method,
00136                              "Number of Recorded Channels": str(num_channels),
00137                              "Sampling Rate to Use": str(rate_used) + " Hz",
00138                              "LFP Signal channel to use": lfp_channel}
00139
00140     parameters_dictionary.update(extra)
00141     Env.print_parameters(parameters_dictionary)
00142
00143     # ------- STEP 1 -------
00144     Env.step(f"Importing LFP data from session: {session}")
00145     lfp_data = data.load_lfp_data(data.LFP[session])
00146
00147     if PLOT:
00148         Env.print_text(f"Plot LFP data from channels 0 and 97 at {data.LFP_DATAMAX_SAMPLING_RATE}Hz")
00149         figname = f"{session}_LFP_C0_and_C97_{data.LFP_DATAMAX_SAMPLING_RATE}Hz"
00150         plt.figure(figname)
00151         plt.subplot(211)
00152         plt.plot(lfp_data[:, 0], "xr")
00153         plt.title(f"Señal LFP. Sesión: {session} del Canal 0 a {data.LFP_DATAMAX_SAMPLING_RATE}Hz")
00154
00155         plt.subplot(212)
00156         plt.plot(lfp_data[:, 97], "xb")
00157         plt.title(f"Señal LFP. Sesión: {session} del Canal 97 a {data.LFP_DATAMAX_SAMPLING_RATE}Hz")
00158         ui.store_figure(figname)
00159
00160         Env.print_text(f"Plot the first 10 seconds of "
00161                        f"LFP data from channel {lfp_channel} at {data.LFP_DATAMAX_SAMPLING_RATE}Hz")
00162         figname = f"{session}_LFP-window_C{lfp_channel}_{data.LFP_DATAMAX_SAMPLING_RATE}Hz"
00163         plt.figure(figname)
00164         plt.plot(lfp_data[0: round(10 * data.LFP_DATAMAX_SAMPLING_RATE), lfp_channel], ".b", markersize=3)
00165         plt.title(f"Primeros 10s de Señal LFP, sesión: {session}, canal: {lfp_channel} a "
00166                   f"{data.LFP_DATAMAX_SAMPLING_RATE}Hz")
00167         ui.store_figure(figname)
00168
00169     # ------- STEP 2 -------
00170     Env.step(f"Importing angles data from session: {session}")
00171
00172     angles_data = data.load_angles_data(data.ANGLES[session])
00173
00174     Env.print_text("Min angle: " + str(np.nanmin(angles_data)) + ", Max angle: " + str(np.nanmax(
      angles_data)))
00175
00176     if PLOT:
00177         Env.print_text(f"Plot Angles data [°] at {data.POSITION_DATA_SAMPLING_RATE}Hz")
00178         figname = f"{session}_Angles_degrees_{data.POSITION_DATA_SAMPLING_RATE}Hz"
```

```
00179         plt.figure(figname)
00180         plt.plot(angles_data[:], "xr")
00181         plt.title(f"Información de ángulos [°]. Sesión: {session} a {data.POSITION_DATA_SAMPLING_RATE}Hz")
00182         ui.store_figure(figname)
00183 
00184         Env.print_text(f"Plot 20 seconds at minute 6 of Angles data at {data.POSITION_DATA_SAMPLING_RATE}Hz
      ")
00185         figname = f"{session}_Angles_degrees-window_{data.POSITION_DATA_SAMPLING_RATE}Hz"
00186         plt.figure(figname)
00187         plt.plot(angles_data[round(6 * 60 * data.POSITION_DATA_SAMPLING_RATE):
00188                              round((6 * 60 + 20) * data.POSITION_DATA_SAMPLING_RATE)], ".r", markersize=4)
00189         plt.title(f"20s al minuto 6 de los ángulos, sesión: {session} a
       {data.POSITION_DATA_SAMPLING_RATE}Hz")
00190         ui.store_figure(figname)
00191 
00192     # ------- STEP 3 -------
00193     if sync_method == "Downsample LFPs":
00194         Env.step("Downsample LFP data to match Angles sampling rate.")
00195 
00196         lfp_data = data.downsample_lfps(lfp_data, data.LFP_DATAMAX_SAMPLING_RATE,
      data.POSITION_DATA_SAMPLING_RATE)
00197 
00198         if PLOT:
00199             Env.print_text(f"Plot LFP downsampled data to {rate_used} Hz from channels 0 and 97.")
00200             figname = f"{session}_LFP_downsampled_0_and_97_{rate_used}Hz"
00201             plt.figure(figname)
00202             plt.subplot(211)
00203             plt.plot(lfp_data[:, 0], "xr")
00204             plt.title(f"Señal LFP del Canal 0 a {rate_used}Hz. Sesión: {session}")
00205 
00206             plt.subplot(212)
00207             plt.plot(lfp_data[:, 97], "xb")
00208             plt.title(f"Señal LFP del Canal 97 a {rate_used}Hz. Sesión: {session}")
00209             ui.store_figure(figname)
00210 
00211             Env.print_text(f"Plot the first 10 seconds of LFP downsampled data from channel {lfp_channel} "
00212                            f"at {rate_used}Hz")
00213             figname = f"{session}_LFP-window-down_C{lfp_channel}_{rate_used}Hz"
00214             plt.figure(figname)
00215             plt.plot(lfp_data[0: round(10 * rate_used), lfp_channel], ".b", markersize=3)
00216             plt.title(f"Primeros 10s de Señal LFP, sesión: {session}, canal: {lfp_channel} a "
00217                       f"{rate_used}Hz")
00218             ui.store_figure(figname)
00219 
00220     elif sync_method == "Upsample Angles":
00221         Env.step("Upsample Angles data to reach a higher sampling rate")
00222 
00223         angles_data = data.angles_expansion(angles_data, data.POSITION_DATA_SAMPLING_RATE,
00224                                             data.LFP_DATAMAX_SAMPLING_RATE)
00225 
00226         if PLOT:
00227             Env.print_text("Plot Angles data expanded with 'NaN' [°]")
00228             figname = f"{session}Expanded_Angles_degrees_{rate_used}Hz"
00229             plt.figure(figname)
00230             plt.plot(angles_data[:], "xb")
00231             plt.title(f"Información de ángulos [°] expandida a {rate_used}Hz. Sesión: {session}")
00232             ui.store_figure(figname)
00233 
00234     # ------- STEP 4 -------
00235     Env.step(f"Interpolate angles data using a {interpolation} approach.", 4)
00236     angles_data_interpolated = data.interpolate_angles(angles_data, interpolation)
00237 
00238     interpolation_ = ""
00239     if sync_method == "Upsample Angles":
00240         interpolation_ = f".\n Interpolada con: {interpolation}"
00241 
00242     Env.print_text(f"Number of NaNs in the Angles Data before interpolation:
       {np.count_nonzero(np.isnan(angles_data))}")
00243     Env.print_text(f"Number of -1s in the Angles Data before interpolation: {np.count_nonzero(angles_data
       == -1)}")
00244 
00245     Env.print_text(f"Number of NaNs in the Angles Data after interpolation: "
00246                    f"{np.count_nonzero(np.isnan(angles_data_interpolated))}")
00247     Env.print_text(f"Number of -1s in the Angles Data after interpolation: "
00248                    f"{np.count_nonzero(angles_data_interpolated == -1)}")
00249 
00250     if PLOT and sync_method == sync_method == "Upsample Angles":
00251         Env.print_text(f"Plot Angles data after {interpolation} interpolation [°]")
00252         figname = f"{session}_Angles_{interpolation} _degrees_{rate_used}Hz"
00253         plt.figure(figname)
00254         plt.plot(angles_data_interpolated[:], "xr")
00255         plt.title(f"Información de ángulos [°]. Sesión: {session}{interpolation_} a {rate_used}Hz")
00256         ui.store_figure(figname)
00257 
00258         Env.print_text(f"Plot 20 seconds at minute 6 of Angles data interpolated with {interpolation} at
       {rate_used}Hz")
00259         figname = f"{session}_Angles_degrees-window_up_{rate_used}Hz"
```

```
00260         plt.figure(figname)
00261         plt.plot(angles_data_interpolated[round(6*60*rate_used): round((6*60+20) * rate_used)], ".r",
    markersize=4)
00262         plt.title(f"20s al minuto 6 de los ángulos, sesión: {session} a {rate_used}Hz."
00263                   f"\n Interpolada con {interpolation}")
00264         ui.store_figure(figname)
00265
00266     # ------- STEP 5 -------
00267     Env.step("Label data by concatenating LFPs and interpolated Angles in a single 2D-array.", 5)
00268
00269     # IF round_angles: Rounding the angles to be discrete labels, starting from 'base_angle' until 360 on
     steps of
00270     # 'offset_between_angles'. Else Not rounding the angles to be discrete labels
00271     labeled_data = data.add_labels(lfp_data, np.expand_dims(angles_data_interpolated, axis=1), round_angles
    ,
00272                                    base_angle, offset_between_angles)
00273
00274     if PLOT:
00275         Env.print_text(f"Plot LFP data from channels 0 and interpolated angles data at {rate_used}Hz. [°]")
00276         figname = f"{session}_LFP_C0_and_angles_{interpolation}_{rate_used}Hz"
00277         plt.figure(figname)
00278         plt.subplot(211)
00279         plt.plot(labeled_data[:-1, 0], "xr")
00280         plt.title(f"Señal LFP del Canal 0. Muestreada a {rate_used}Hz. Sesión: {session}")
00281
00282         plt.subplot(212)
00283         plt.plot(labeled_data[:, -1], "xb")
00284         plt.title(f"Información de ángulos [°]. Sesión: {session}{interpolation_} a {rate_used}Hz")
00285         ui.store_figure(figname)
00286
00287     # ------- STEP 6 -------
00288     Env.step("Clean the labeled dataset from invalid values.")
00289
00290     clean_dataset = data.clean_invalid_positional(labeled_data, sync_method == "Upsample Angles")
00291
00292     if round_angles:
00293         labels = np.arange(base_angle, 360, offset_between_angles)
00294         percentages = []
00295         for u in range(base_angle, 360, offset_between_angles):
00296             percentages.append(round(np.sum(clean_dataset[:, -1] == u) * 100 / len(clean_dataset[:, -1])))
00297
00298         labels_percent = np.concatenate((np.expand_dims(labels, axis=1), np.expand_dims(percentages, axis=1
    )), axis=1)
00299         dataframe_labels = pd.DataFrame(data=labels_percent, columns=["Angulos", "Porcentaje"])
00300
00301         if PLOT:
00302             Env.print_text(f"Plot Barplot of Labels after {interpolation} interpolation")
00303             figname = f"{session}_BarPLotAngles_{interpolation}_{rate_used}Hz"
00304             plt.figure(figname)
00305             sns.barplot(x="Angulos", y="Porcentaje", data=dataframe_labels)
00306             plt.title(f"Gráfico de Barras de las etiquetas. Sesión: {session}{interpolation_} a "
00307                       f"{rate_used}Hz")
00308             ui.store_figure(figname)
00309
00310     # ------- STEP 7 -------
00311     Env.step("Count and print number of NaNs in the Dataset.")
00312
00313     invalid_begin = 0
00314     invalid_end = 0
00315     length = len(angles_data_interpolated)
00316     for i in range(length):
00317         if angles_data_interpolated[i] != -1:
00318             invalid_begin = i
00319             break
00320     for i in range(length - 1, 0, -1):
00321         if angles_data_interpolated[i] != -1:
00322             invalid_end = i
00323             break
00324     invalid_end = length - (invalid_end + 1)
00325
00326     Env.print_text(
00327         f"Number of invalid in Angles Data without interpolation: {np.count_nonzero(angles_data == -1)}")
00328     Env.print_text("Number of invalid in Labeled Dataset with interpolated Angles Data: "
00329                    + str(np.count_nonzero(angles_data_interpolated == -1)))
00330     Env.print_text("Number of invalid in Labeled and Clean Dataset with interpolated Angles Data: "
00331                    + str(np.count_nonzero(clean_dataset[0][:, -1] == -1)))
00332
00333     if PLOT:
00334         # ------- STEP 8 -------
00335         Env.step(f"Plot clean LFP data from channels 0 and interpolated angles data at {rate_used}Hz. [°]")
00336
00337         figname = f"{session}_LFP_C{lfp_channel}_clean_and_angles_{interpolation}_{rate_used}Hz"
00338         plt.figure(figname)
00339         plt.subplot(211)
00340         plt.plot(clean_dataset[0][:, lfp_channel], "xr")
00341         plt.title(f"Señal LFP del Canal {lfp_channel}. Muestreada a {rate_used}Hz. Sesión: {session}")
00342         plt.subplot(212)
```

```
00343         plt.plot(clean_dataset[0][:, -1], "xb")
00344         plt.title(f"Información de ángulos [°]. Sesión: {session}{interpolation_} a "
00345                   f"{rate_used}Hz.")
00346         ui.store_figure(figname)
00347 
00348     if PLOT:
00349         # ------- STEP 9 -------
00350         Env.step("Plot Boxplot of all channels in two figures, from 0-49 and from 50-98.")
00351 
00352         figname = f"{session}_BoxPLot0-49_{interpolation}_{rate_used}Hz"
00353         plt.figure(figname, figsize=[12, 16])
00354         sns.boxplot(data=clean_dataset[0][:, 0:50], orient="h")
00355         plt.ylabel("Canales")
00356         plt.xlabel("Voltaje")
00357         plt.title(f"Diagrama de caja de los canales 0-49.\nSesión: {session}{interpolation_}"
00358                   f" a {rate_used}Hz")
00359         ui.store_figure(figname)
00360 
00361         figname = f"{session}_BoxPLot50-98_{interpolation}_{rate_used}Hz"
00362         plt.figure(figname, figsize=[12, 16])
00363         sns.boxplot(data=clean_dataset[0][:, 50:99], orient="h")
00364         plt.ylabel("Canales")
00365         plt.xlabel("Voltaje")
00366         plt.title(f"Diagrama de caja de los canales 50-98.\nSesión: {session}{interpolation_}"
00367                   f" a {rate_used}Hz")
00368         ui.store_figure(figname)
00369 
00370         # ------- STEP 10 -------
00371         Env.step("Plot Boxplot of interpolated angles.")
00372 
00373         figname = f"{session}_BoxPLotAngles_{interpolation}_{rate_used}Hz"
00374         plt.figure(figname)
00375         sns.boxplot(x=clean_dataset[0][:, -1])
00376         plt.ylabel("Muestras")
00377         plt.xlabel("Ángulos")
00378         plt.title(f"Diagrama de caja de los ángulos.\nSesión: {session}{interpolation_} "
00379                   f"a {rate_used}Hz")
00380         ui.store_figure(figname)
00381 
00382     # ------- FINISH TEST AND EXIT -------
00383     #Env.finish_test()
00384     Env.finish_test(f"{session}_{interpolation}_" + sync_method.replace(" ", ""))
00385 
00386 
00387 # ------- Execute Dataset Exploration -------
00388 data_exploration()
```

5.3 /home/pabloav/Documents/Tesis_Lic/LFP-Classification/DatasetGenerator/Dataset↩ Generator.py File Reference

Namespaces

- DatasetGenerator

Functions

- def DatasetGenerator.data_set_generator ()

5.3.1 Detailed Description

Author

Pablo Avila [B30724] jose.avilalopez@ucr.ac.cr

Copyright

MIT License

Date

July, 2021

Properly loads the input data and labels and prepare a clean dataset to finally export it as a pickle file.

Definition in file DatasetGenerator.py.

5.4 DatasetGenerator.py

```
00001 # ##########################################################################
00002 #                  University of Costa Rica
00003 #               Electrical Engineering Department
00004 #                         Grade Thesis
00005 # ##########################################################################
00006
00007 """
00008 @file DatasetGenerator.py
00009 @author Pablo Avila [B30724] jose.avilalopez@ucr.ac.cr
00010 @copyright MIT License
00011 @date July, 2021
00012 @details Properly loads the input data and labels and prepare a clean dataset to finally export it as a
       pickle file.
00013 """
00014
00015 from Yomchi import \
00016     Environment as Env, \
00017     preprocessing as data, \
00018     visualization as ui
00019
00020 import matplotlib.pyplot as plt
00021 import numpy as np
00022 import tensorflow as tf
00023 import pickle
00024
00025
00026 def data_set_generator():
00027     """
00028     @details
00029     <h1> Dataset Generator:</h1>
00030     <h2> Setup </h2>
00031     <ul>
00032         <li> Initialize Environment.
00033         <li> Initialize Tensorflow session.
00034         <li> Set seed for Numpy and Tensorflow
00035         <li> Specify run configuration parameters.
00036         <li> Specify dataset generation parameters.
00037     </ul>
00038     <h2> Procedure </h2>
00039     <ul>
00040         <li> <b>Step 1:</b> Import LFP data.
00041         <li> <b>Step 2:</b> Import angles data.
00042         <li> <b>Step 3:</b>
00043         <ul>
00044             <li> Downsample LFP data to match the Angles sampling rate or
00045             <li> Fill angles gaps to match the LFP sampling rate.
00046         </ul>
00047         <li> <b>Step 4:</b> Label data by concatenating LFPs and Angles in a single 2D-array.
00048         <li> <b>Step 5:</b> Clean the labeled dataframe from -1 values, which represent the wrongly
       acquired
00049         positional samples.
00050         <li> <b>Step 6:</b> Interpolate angles data using an 'interpolation' approach.
00051         <li> <b>Step 7:</b> Plotting Angles and the selected channel of the LFPs after cleaning and
      interpolation.
00052         <li> <b>Step 8:</b>
00053         <ul>
00054             <li> Get preferred angle according to LFPs channel.
00055             <li> Plotting the sum of LFPs per angle from 0 to 359 degrees.
00056         </ul>
00057         <li> <b>Step 9:</b> Get largest subset of data.
00058         <li> <b>Step 10:</b> Split the subset into training, validation and testing set.
00059         <li> <b>Step 11:</b> Save the dataset to a pickle file.
00060         <li> <b>Finish Test and Exit: </b> Save logs, captures, results and the generated dataset into a
       .pickle file.
00061     </ul>
00062     """
00063
00064     # ------- SETUP -------
00065     Env.init_environment(True)
00066
00067     tf.keras.backend.clear_session()
00068     tf.random.set_seed(51)
```

```
00069     np.random.seed(51)
00070
00071     # Dataset Generation Parameters
00072     # The recording session 771 has a 23.81% of invalid positions, while the session 765 has only 6.98%
00073     session = 771  # 771 or 765
00074     interpolation = "linear"  # "linear" "quadratic" "cubic" "nearest" "Shortest Angle"
00075     sync_method = "Upsample Angles"  # "Upsample Angles" "Downsample LFPs"
00076
00077     # Data Properties
00078     num_channels = data.EC014_41_NUMBER_OF_CHANNELS
00079     rate_used = data.POSITION_DATA_SAMPLING_RATE
00080     if sync_method == "Upsample Angles":
00081         rate_used = data.LFP_DATAMAX_SAMPLING_RATE
00082
00083     # Windowing properties
00084     lfp_channel = 70
00085     round_angles = False
00086     base_angle = 0  # Unused if round_angles = False
00087     offset_between_angles = 30  # Unused if round_angles = False
00088
00089     extra = {"Round angles to get discrete label": round_angles}
00090     if round_angles:
00091         extra.update({"Angle labels starting from": str(base_angle) + "°",
00092                       "until 360° in steps of": str(offset_between_angles) + "°"})
00093
00094     o_pickle_file_name = f"S-{session}_I-" + interpolation.replace(" ", "") + f"_F-{int(rate_used)}.pickle"
00095
00096     parameters_dictionary = {"Recording Session Number": str(session),
00097                              "Interpolation Method": interpolation.title(),
00098                              "Synchronization Method": sync_method,
00099                              "Number of Recorded Channels": str(num_channels),
00100                              "Sampling Rate to Use": str(rate_used) + " Hz",
00101                              "LFP Signal channel to use": lfp_channel}
00102
00103     parameters_dictionary.update(extra)
00104     Env.print_parameters(parameters_dictionary)
00105
00106     # ------- STEP 1 -------
00107     Env.step(f"Importing LFP data from session: {session}")
00108     lfp_data = data.load_lfp_data(data.LFP[session])
00109
00110     # ------- STEP 2 -------
00111     Env.step(f"Importing angles data from session: {session}")
00112     angles_data = data.load_angles_data(data.ANGLES[session])
00113
00114     # ------- STEP 3 -------
00115     interpolation_ = ""
00116     if sync_method == "Downsample LFPs":
00117         Env.step("Downsample LFP data to match Angles sampling rate.")
00118         lfp_data = data.downsample_lfps(lfp_data, data.LFP_DATAMAX_SAMPLING_RATE,
      data.POSITION_DATA_SAMPLING_RATE)
00119
00120     elif sync_method == "Upsample Angles":
00121         Env.step("Upsample Angles data to reach a higher sampling rate")
00122         angles_data = data.angles_expansion(angles_data, data.POSITION_DATA_SAMPLING_RATE,
00123                                             data.LFP_DATAMAX_SAMPLING_RATE)
00124         interpolation_ = f"Interpolados con: {interpolation}"
00125
00126     # ------- STEP 4 -------
00127     Env.step("Label data by concatenating LFPs and interpolated Angles in a single 2D-array.")
00128     # IF round_angles: Rounding the angles to be discrete labels, starting from 'base_angle' until 360 on
       steps of
00129     # 'offset_between_angles'. Else, the angles are not rounded to discrete labels.
00130     labeled_data = data.add_labels(lfp_data, np.expand_dims(angles_data, axis=1), round_angles,
00131                                    base_angle, offset_between_angles)
00132
00133     # ------- STEP 5 -------
00134     Env.step("Clean the labeled dataset from discontinuities in positional data (angles).")
00135
00136     clean_datasets = data.clean_invalid_positional(labeled_data, is_padded=(sync_method == "Upsample Angles
      "))
00137
00138     # ------- STEP 6 -------
00139     Env.step(f"Interpolate angles data using a {interpolation} approach.")
00140
00141     # Get all LFP channels, excluding the angles.
00142     clean_interpolated_data = [s[:, :-1] for s in clean_datasets]
00143
00144     # Interpolate angles and add them to the dataset
00145     clean_interpolated_angles = [data.interpolate_angles(s[:, -1], interpolation) for s in clean_datasets]
00146
00147     for i in range(len(clean_interpolated_data)):
00148         clean_interpolated_data[i] = \
00149             np.concatenate((clean_interpolated_data[i], np.expand_dims(clean_interpolated_angles[i], axis=1
      )), axis=1)
00150
00151     # ------- STEP 7 -------
```

```
00152     Env.step("Plotting Angles and one channel of the LFPs after cleaning and interpolation.")
00153
00154     Env.print_text(f"Plotting Angles data [°] at {rate_used}Hz")
00155     figname = f"{session}_Angles_degrees_{int(rate_used)}Hz"
00156     plt.figure(figname)
00157     plt.plot(clean_interpolated_data[0][0:round(3*60*rate_used), -1], "xr")
00158     plt.title(f"3 minutos de Ángulos [°] limpios.\n {interpolation_}. Sesión: {session} a {rate_used}Hz")
00159     ui.store_figure(figname, show=Env.debug)
00160
00161     Env.print_text(f"Plotting LFPs Channel {lfp_channel} at {rate_used}Hz")
00162
00163     figname = f"{session}_LFP_c{lfp_channel}_{int(rate_used)}Hz"
00164     plt.figure(figname)
00165     plt.plot(clean_interpolated_data[4][:, lfp_channel], "x-r")
00166     plt.title(f"LFPs del Canal {lfp_channel}. Sesión: {session} a {rate_used}Hz")
00167     ui.store_figure(figname, show=Env.debug)
00168
00169     Env.print_text(f"Plot clean LFP data from the chosen channel and interpolated angles data at
      {rate_used}Hz. [°]")
00170
00171     figname = f"{session}_LFP_C{lfp_channel}_clean_and_angles_{interpolation}_{rate_used}Hz"
00172     fig, axs = plt.subplots(2, 1, num=figname, constrained_layout=True)
00173     fig.suptitle(f"5 minutos limpios de la sesión: {session} a {rate_used}Hz.")
00174     axs[0].plot(clean_interpolated_data[0][0:round(5*60*rate_used), lfp_channel], "xr")
00175     axs[0].set_title(f"Señal LFP del Canal {lfp_channel}.")
00176     axs[1].plot(clean_interpolated_data[0][0:round(5*60*rate_used), -1], "xb")
00177     axs[1].set_title(f"Ángulos [°]. {interpolation_}.")
00178     ui.store_figure(figname, show=Env.debug)
00179
00180
00181     # ------- STEP 8 -------
00182     Env.step("Get Preferred angle.")
00183
00184     angles = [0] * 361
00185
00186     for subset in clean_interpolated_data:
00187         for index in range(len(subset)):
00188             angles[int(round(subset[index, -1]))] += abs(subset[index, lfp_channel])
00189
00190     angles[0] += angles[360]
00191     angles = angles[:-1]
00192     preferred_angle = angles.index(max(angles))
00193
00194     Env.print_text(f" Preferred angle according to LFP Channel {lfp_channel}: {preferred_angle}°")
00195
00196     Env.print_text(f"Plotting Preferred angle according to LFP Channel {lfp_channel}")
00197     figname = f"{session}_preferred_angle_LFP_c{lfp_channel}_{int(rate_used)}Hz"
00198     plt.figure(figname)
00199     plt.plot(range(360), angles, "x-r")
00200     plt.title(f"Suma de LFPs del Canal {lfp_channel} por ángulo. Sesión: {session}.")
00201     plt.xlabel(f"Ángulo en grados")
00202     ui.store_figure(figname, show=Env.debug)
00203
00204     # ------- STEP 9 -------
00205     Env.step("Get largest subset of data.")
00206
00207     max_length = 0
00208     max_subset_index = 0
00209     for index, subset in enumerate(clean_interpolated_data):
00210         if max_length < len(subset):
00211             max_length = len(subset)
00212             max_subset_index = index
00213
00214     largest_subset = clean_interpolated_data[max_subset_index]
00215     Env.print_text(f"Shape of the largest subset of data : [{len(largest_subset)},{len(largest_subset[0])}]
    ")
00216
00217     # ------- STEP 10 -------
00218     Env.step()
00219
00220     n = len(largest_subset)
00221     Env.print_text(f"Largest subset: {max_subset_index} = {n} samples")
00222     train_array = largest_subset[0:int(n * 0.7), :]
00223     valid_array = largest_subset[int(n * 0.7):int(n * 0.9), :]
00224     test_array = largest_subset[int(n * 0.9):, :]
00225
00226     Env.print_text(f"Training data shape: [{len(train_array)},{len(train_array[0])}]")
00227     Env.print_text(f"Validation data shape: [{len(valid_array)},{len(valid_array[0])}]")
00228     Env.print_text(f"Test data shape: [{len(test_array)},{len(test_array[0])}]")
00229
00230     # ------- STEP 11 -------
00231     Env.step(f"Save the dataset to pickle file: {o_pickle_file_name}.")
00232
00233     with open(f"{Env.RESULTS_FOLDER}/{o_pickle_file_name}", 'wb') as f:
00234         pickle.dump([train_array, valid_array, test_array], f)
00235
00236     # ------- FINISH TEST AND EXIT -------
```

```
00237     #Env.finish_test()
00238     Env.finish_test(str(session) + "_" + interpolation.replace(" ", "") + "_" + sync_method.replace(" ", ""
    ))
00239
00240
00241 # ------- Execute Dataset generation -------
00242 data_set_generator()
```

5.5 /home/pabloav/Documents/Tesis_Lic/LFP-Classification/Predictor/Predictor.py File Reference

Namespaces

- Predictor

Functions

- def Predictor.predictor ()

5.5.1 Detailed Description

Author

Pablo Avila [B30724] jose.avilalopez@ucr.ac.cr

Copyright

MIT License

Date

July, 2021

The predictor loads a dataset of LFP with Angles as labels previously generated and feed it to a Neural Network in order to train it to predict new angles based on LFP data windows.

Definition in file Predictor.py.

5.6 Predictor.py

```
00001 # ##########################################################################
00002 #                  University of Costa Rica
00003 #               Electrical Engineering Department
00004 #                          Grade Thesis
00005 # ##########################################################################
00006
00007 """
00008 @file Predictor.py
00009 @author Pablo Avila [B30724] jose.avilalopez@ucr.ac.cr
00010 @copyright MIT License
00011 @date July, 2021
00012 @details The predictor loads a dataset of LFP with Angles as labels previously generated and feed it to a
       Neural
00013 Network in order to train it to predict new angles based on LFP data windows.
00014 """
00015
00016 from Yomchi import \
00017     Environment as Env, \
00018     preprocessing as data, \
00019     models, \
00020     visualization as ui
00021
00022 import numpy as np
```

```
00023 import tensorflow as tf
00024 import matplotlib.pyplot as plt
00025 import pickle
00026
00027
00028 def predictor():
00029     """
00030     @details
00031     <h1> Predictor </h1>
00032     <h2> Experiment Setup </h2>
00033     <ul>
00034         <li> Initialize Environment
00035         <li> Initialize Tensorflow session.
00036         <li> Set seed for Numpy and Tensorflow
00037         <li> Specify run configuration parameters.
00038         <li> Specify run configuration parameters.
00039         <li> Specify dataset generation parameters.
00040         <li> Specify model's training parameters.
00041     </ul>
00042     <h2> Procedure </h2>
00043     <ul>
00044         <li> <b>Step 1:</b> Import training, validation and test datasets of LFPs as inputs and Angles as
      labels.
00045         <li> <b>Step 2:</b> Convert data to windowed series.
00046         <li> <b>Step 3:</b> Create the model with the specified parameters
00047         <li> <b>Step 4:</b> Train the model with the training and validation data.
00048         <li> <b>Step 5:</b> Plotting the original angular data vs the predictions for different batch
      sizes.
00049         <li> <b>Step 6:</b> Plotting the original angular data vs the predictions of all testing data.
00050         <li> <b>Step 7:</b> Plotting the historic of the training vs the validation error values.
00051         <li> <b>Finish Test and Exit:</b> Save logs, captures and results.
00052     </ul>
00053     """
00054
00055     # ------- SETUP -------
00056     Env.init_environment(True)
00057
00058     tf.keras.backend.clear_session()
00059     tf.keras.backend.set_floatx('float64')
00060     tf.random.set_seed(51)
00061     np.random.seed(51)
00062
00063     # Session, data and methods Parameters
00064     session = 771  # 765 or 771
00065     interpolation = "Shortest Angle"  # or "linear", "quadratic", "cubic", "nearest", "Shortest Angle"
00066     rate_used = data.POSITION_DATA_SAMPLING_RATE  # data.LFP_DATAMAX_SAMPLING_RATE or
      data.POSITION_DATA_SAMPLING_RATE
00067
00068     # Windowing properties
00069     window_size = int(rate_used * 0.1 * 2)  # Frequency_used * 0.1 = 100ms.
00070     batch_size = 32
00071     shuffle_buffer = 1000
00072     lfp_channel = 40
00073     batches_to_plot = [5, 10, 20, 50]
00074
00075     # Parameters
00076     units_per_layer = 32
00077     dropout = 0.2
00078     layers = 3
00079     epochs = 50
00080     model_name = "CNN"  # "LSTM" "MLP" or "CNN"
00081
00082     # Input pickle file.
00083     i_pickle_file_name = f"S-{session}_I-" + interpolation.replace(" ", "") + f"_F-{int(rate_used)}.pickle"
00084     parameters_dictionary = {"Recording Session Number": str(session),
00085                              "Interpolation Method": interpolation.title(),
00086                              "Sampling Rate to Use": str(rate_used) + "Hz",
00087                              "LFP Signal channel to use": lfp_channel,
00088                              "Shuffle buffer size": str(shuffle_buffer),
00089                              "Pickle file used": i_pickle_file_name,
00090                              "Batches to Plot": str(batches_to_plot),
00091                              "Batch size (# of windows)": batch_size,
00092                              "Window Size": f"{window_size} samples ({int(window_size * 1e3 /
      rate_used)}ms)",
00093                              "Model Name": model_name,
00094                              "Layers (for MLP model only)": str(layers),
00095                              "Epochs": str(epochs),
00096                              "Units per Layer": str(units_per_layer),
00097                              "Dropout Regularization %": str(dropout)
00098                              }
00099
00100     Env.print_parameters(parameters_dictionary)
00101
00102     # ------- STEP 1 -------
00103     Env.step("Import training, validation and test datasets of LFPs as inputs and Angles as labels.")
00104
00105     with open(f"{data.PATH_TO_DATASETS}Pickles/{i_pickle_file_name}", "rb") as f:
```

```
00106         train_array, valid_array, test_array = pickle.load(f)
00107
00108     # ------- STEP 2 -------
00109     Env.step("Convert data to windowed series.")
00110
00111     # The train data is shuffled so the network is trained with a variety of examples.
00112     train_data = data.channels_to_windows(train_array, lfp_channel, window_size, batch_size, shuffle_buffer
    )
00113
00114     # The validation and testing data are not shuffled because the intention is to predict the series in
     order.
00115     val_data = data.channels_to_windows(valid_array, lfp_channel, window_size, batch_size)
00116     test_data = data.channels_to_windows(test_array, lfp_channel, window_size, batch_size)
00117
00118     for example_inputs, example_labels in train_data.take(1):
00119         Env.print_text(f'Inputs shape (batch, time, channels): {example_inputs.shape}')
00120         Env.print_text(f'Labels shape (batch, time, labels): {example_labels.shape}')
00121
00122     # ------- STEP 3 -------
00123     Env.step("Create the model with the specified parameters ")
00124
00125     if model_name == "LSTM":
00126         model = models.lstm(units_per_layer, dropout)
00127     elif model_name == "CNN":
00128         model = models.cnn(window_size, units_per_layer, dropout)
00129     else:
00130         model = models.mlp(layers, units_per_layer, dropout)
00131
00132     for example_inputs, example_labels in train_data.take(1):
00133         Env.print_text(f'Input shape: {example_inputs.shape}')
00134         Env.print_text(f'Output shape: {model(example_inputs).shape}')
00135
00136     # ------- STEP 4 -------
00137     Env.step("Train the model with the training and validation data.")
00138
00139     history = models.compile_and_fit(model, train_data, val_data, epochs=epochs,
    patience=epochs)
00140
00141     Env.print_text(f"\nTraining metrics history:")
00142     Env.print_text(f"Loss: {history.history['loss']}")
00143     Env.print_text(f"Mean Absolute Error: {history.history['mean_absolute_error']}")
00144     Env.print_text(f"\nValidation metrics history:")
00145     Env.print_text(f"Loss: {history.history['val_loss']}")
00146     Env.print_text(f"Mean Absolute Error: {history.history['val_mean_absolute_error']}")
00147
00148     # ------- STEP 5 -------
00149     Env.step(f"Plotting the original angular data vs the predictions of {str(batches_to_plot)} batches")
00150
00151     lfps = []
00152     real = []
00153     pred = []
00154
00155     interpolated = ""
00156     if rate_used == data.LFP_DATAMAX_SAMPLING_RATE:
00157         interpolated = f"Interpolación: {interpolation}"
00158         if interpolation == "Shortest Angle":
00159             interpolated = "Interpolación: Por ángulo corto."
00160
00161     for batches in batches_to_plot:
00162         for inputs, label in test_data.take(batches):
00163             # Getting original data
00164             real = np.append(real, label[:, 0, 0].numpy())
00165             lfps = np.append(lfps, inputs[:, 0, 0].numpy())
00166             # Getting predicted data
00167             pred = np.append(pred, model.predict(inputs))
00168
00169         # Plotting
00170         figname = f"{model_name}_S-{session}_W-{int(window_size * 1e3 /
     rate_used)}_F-{int(rate_used)}_E-{epochs}_C-" \
00171                   f"{lfp_channel}_B{batches}"
00172
00173         fig, axs = plt.subplots(2, 1, num=figname)
00174         axs[0].set_title(
00175             f"{model_name} fs:{rate_used} Epochs:{epochs} Batches:{batches} Ventana:"
00176             f"{int(window_size * 1e3 / rate_used)}ms."
00177             f" {interpolated}")
00178
00179         axs[0].set_xlabel(f"Predicción de Ángulos contra ángulos originales.")
00180         axs[0].scatter(range(len(real)), real, label='Originales', c='#2ca02c', s=10)
00181         axs[0].scatter(range(len(pred)), pred, marker='X', label='Predicciones', c='#ff7f0e', s=16)
00182         axs[0].legend()
00183
00184         axs[1].set_xlabel(f'Canal de LFP: {lfp_channel}')
00185         axs[1].scatter(range(len(lfps)), lfps, label='LFPs', c='#0a7fdb', s=10)
00186         axs[1].legend()
00187         ui.store_figure(figname, show=Env.debug)
00188
```

```
00189     for inputs, label in test_data:
00190         # Getting original data
00191         real = np.append(real, label[:, 0, 0].numpy())
00192         lfps = np.append(lfps, inputs[:, 0, 0].numpy())
00193         # Getting predicted data
00194         pred = np.append(pred, model.predict(inputs))
00195 
00196     # ------- STEP 6 -------
00197     Env.step(f"Plotting the original angular data vs the predictions of all testing data")
00198 
00199     figname = f"{model_name}_S-{session}_W-{int(window_size * 1e3 /
      rate_used)}_F-{int(rate_used)}_E-{epochs}_C-" \
00200               f"{lfp_channel}_BAll"
00201 
00202     fig, axs = plt.subplots(2, 1, num=figname)
00203     axs[0].set_title(
00204         f"{model_name} fs:{rate_used} Epochs:{epochs} Batches: Todos. Ventana:"
00205         f"{int(window_size * 1e3 / rate_used)}ms."
00206         f" {interpolated}")
00207 
00208     axs[0].set_xlabel(f"Predicción de Ángulos contra ángulos originales.")
00209     axs[0].scatter(range(len(real)), real, label='Originales', c='#2ca02c', s=10)
00210     axs[0].scatter(range(len(pred)), pred, marker='X', label='Predicciones', c='#ff7f0e', s=16)
00211     axs[0].legend()
00212 
00213     axs[1].set_xlabel(f'Canal de LFP: {lfp_channel}')
00214     axs[1].scatter(range(len(lfps)), lfps, label='LFPs', c='#0a7fdb', s=10)
00215     axs[1].legend()
00216     ui.store_figure(figname, show=Env.debug)
00217 
00218     # ------- STEP 7 -------
00219     Env.step(f"Plotting the historic of the training vs the validation error values.")
00220 
00221     figname = f"{model_name}_S-{session}_W-{int(window_size * 1e3 /
      rate_used)}_F-{int(rate_used)}_E-{epochs}_C-" \
00222               f"{lfp_channel}_history"
00223     plt.figure(figname)
00224     plt.plot(history.history['mean_absolute_error'], "x-r", label="Entrenamiento")
00225     plt.plot(history.history['val_mean_absolute_error'], "x-b", label="Validación")
00226     plt.xlabel("Época")
00227     plt.ylabel("MAE")
00228     plt.legend(loc='best')
00229     plt.title(f"Error Medio contra épocas. {model_name} fs:{rate_used} Epochs:{epochs} Ventana:"
00230               f"{int(window_size * 1e3 / rate_used)}ms. {interpolated}")
00231     ui.store_figure(figname)
00232 
00233     # ------- FINISH TEST AND EXIT -------
00234     # Env.finish_test()
00235     Env.finish_test(f"M-{model_name}_S-{session}_W-{int(window_size * 1e3 / rate_used)}_F-{int(rate_used)}"
00236                     f"_E-{epochs}_C-{lfp_channel}")
00237 
00238 
00239 # ------- Execute Predictor -------
00240 predictor()
```

5.7 /home/pabloav/Documents/Tesis_Lic/LFP-Classification/README.md File Reference

5.8 /home/pabloav/Documents/Tesis_Lic/LFP-Classification/README.md

```
00001 # LFP Classification
00002 Classifier of Local Field Potential signals using Deep Neural Networks for head angular direction
       prediction
00003 
00004 ## Prerequisites
00005 The current project was developed using **Ubuntu 18.04.4 LTS**.
00006 The following are the required packages and python libraries.
00007 It is recommended to use the listed versions, nevertheless is worth to try the newest available
       version of them.
00008 
00009 ### Packages:
00010 ```
00011 1. Python (3.6)
00012 ```
00013 
00014 **Optional:**
00015 ```
00016 1. Latex
00017 2. Doxygen
00018 3. Doxypy
00019 4. Make
00020 5. Git
```

```
00021 6. Graphviz
00022 ```
00023
00024 ### Python libraries:
00025 Package | Version
00026 ------- | --------
00027 pip3 | 20.1
00028 numpy | 1.18.4
00029 matplotlib | 3.2.1
00030 tensorflow | 2.2.0
00031 keras | 2.3.1
00032 sklearn | 0.0
00033 pandas | 1.0.3
00034 scikit-learn | 0.23.0
00035 seaborn | 0.10.1
00036
00037
00038 ## Install
00039
00040 Check the installed Linux version using: `lsb_release -a`.
00041 Check the installed version of the package using: `<package> --<version>`
00042 Check the installed python libraries and their version using: `pip list`
00043
00044 ### Packages:
00045 ```
00046 sudo apt-get install python3.6 texlive-latex-extra doxygen doxygen-gui make git graphviz
00047 ```
00048
00049 ### Pip:
00050
00051 This is necessary to install the rest of python libraries
00052
00053 ```
00054 curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py
00055 python3.6 get-pip.py
00056 ```
00057
00058 ### Python libraries:
00059
00060 If getting the latest available version:
00061 ```
00062 pip3 install --upgrade <package>
00063 ```
00064
00065 To download an specific version:
00066 ```
00067 pip3 install <package>==<version>
00068 ```
00069
00070 ## Documentation
00071  - You can start reading the documentation in the web veiw here
       [LFP-Classification-Documentation-Page](https://pabloavlo.github.io/LFP-Classification/html/index.html).
00072  - To read it as PDF run ```make``` under *docs/* and open the generated *latex/refman.pdf* file.
00073
00074 ## Author
00075 **Pablo Avila** - [PabloAvLo](https://github.com/PabloAvLo)
00076
00077 ## License
00078 This project is licensed under the MIT License - see the LICENSE.md file for details
00079
00080 ## Acknowledgments
00081
00082 * The present code is part of my grade thesis in Electrical Engineering at the University of Costa
       Rica.
00083
00084 * The main objective is to prove the hypothesis that based on Local Field Potential measurments from
       the Entorhinal Cortex of the Hippocampus of rats is possible to extract angular information of the head
       direction of the subject.
00085
00086 * The dataset used in this experiment is: [BuszakiLab HC-3
       Dataset](https://crcns.org/data-sets/hc/hc-3/about-hc-3/?searchterm=LFP).
00087
00088 * In order to do so, machine learning algorithms will be applied to the data, such as Feedforward
       Neural Network, RNNs and CNNs.
```

5.9 /home/pabloav/Documents/Tesis_Lic/LFP-Classification/Yomchi/Environment.py File Reference

Namespaces

- Environment

Functions

- def Environment.init_environment (print_the_header=False, enable_debug=False)

  Initialize the directories and files to log the results.
- def Environment.print_text (text)

  Print a passed text and save it in the yomchi_log.txt.
- def Environment.print_box (text)

  Print a passed text in a box and save it in the yomchi_log.txt.
- def Environment.step (description=None, new_step_number=0)

  Print the current step number with a brief description.
- def Environment.print_parameters (params_dictionary)

  Print a list of parameters.
- def Environment.log_versions ()

  Stores the environment versions including packages, libraries and OS.
- def Environment.print_header ()

  Print a header in the console output and log files.
- def Environment.finish_test (rename_results_folder=None)

  Safely finish the test run and log some final information.

Variables

- Environment.START_TIME = datetime.datetime.now()
- Environment.CURRENT_FOLDER = os.getcwd()
- string Environment.RESULTS_FOLDER = CURRENT_FOLDER + "/Results-" + START_TIME.strftime("%Y-%m-%d_%H-%M") + "/"
- string Environment.LOGS_FOLDER = RESULTS_FOLDER + "Logs/"
- string Environment.CAPTURES_FOLDER = RESULTS_FOLDER + "Captures/"
- string Environment.VERSIONS_LOG_PATH = LOGS_FOLDER + "versions_log.txt"
- string Environment.YOMCHI_LOG_PATH = LOGS_FOLDER + "yomchi_log.txt"
- int Environment.LINE_LENGTH = 100
- int Environment.step_number = 1
- Environment.versions_log = None
- Environment.yomchi_log = None
- Environment.caller_file = None
- bool Environment.debug = False

5.9.1 Detailed Description

Author

Pablo Avila [B30724] jose.avilalopez@ucr.ac.cr

Copyright

MIT License

Date

July, 2021

Python environment which integrates data pre-processing, ML models implementation, data visualization and metrics. This module contains a set of functions to successfully document every test run by creating Results folders with console logs, generated images, dependencies versions, etc.

Definition in file Environment.py.

5.10 Environment.py

```
00001 # ##############################################################################
00002 #                   University of Costa Rica
00003 #               Electrical Engineering Department
00004 #                         Grade Thesis
00005 # ##############################################################################
00006
00007 """
00008 @file Environment.py
00009 @author Pablo Avila [B30724] jose.avilalopez@ucr.ac.cr
00010 @copyright MIT License
00011 @date July, 2021
00012 @details Python environment which integrates data pre-processing, ML models implementation, data
       visualization and
00013 metrics. This module contains a set of functions to successfully document every test run by creating
       Results folders
00014 with console logs, generated images, dependencies versions, etc.
00015 """
00016
00017 import os
00018 import sys
00019 import inspect
00020 import platform
00021 import datetime
00022 import numpy as np
00023
00024 # Module Constants
00025 START_TIME = datetime.datetime.now()
00026 CURRENT_FOLDER = os.getcwd()
00027 RESULTS_FOLDER = CURRENT_FOLDER + "/Results-" + START_TIME.strftime("%Y-%m-%d_%H-%M") + "/"
00028 LOGS_FOLDER = RESULTS_FOLDER + "Logs/"
00029 CAPTURES_FOLDER = RESULTS_FOLDER + "Captures/"
00030 VERSIONS_LOG_PATH = LOGS_FOLDER + "versions_log.txt"
00031 YOMCHI_LOG_PATH = LOGS_FOLDER + "yomchi_log.txt"
00032 LINE_LENGTH = 100
00033
00034 # Module Variables
00035 step_number = 1
00036 versions_log = None
00037 yomchi_log = None
00038 caller_file = None
00039 debug = False
00040
00041
00042 def init_environment(print_the_header=False, enable_debug=False):
00043     """
00044     Initialize the directories and files to log the results.
00045     @param print_the_header: If True, print the header
00046     @param enable_debug: If True, run some code to help debugging
00047     """
00048     global versions_log
00049     global yomchi_log
00050     global caller_file
00051     global debug
00052
00053     debug = enable_debug
00054     os.makedirs(RESULTS_FOLDER)
00055     os.makedirs(LOGS_FOLDER)
00056     os.makedirs(CAPTURES_FOLDER)
00057
00058     # First get the full filename (including path and file extension)
00059     caller_frame = inspect.stack()[1]
00060     caller_filename = caller_frame.filename
00061
00062     # Now get rid of the directory and extension
00063     caller_file = os.path.basename(caller_filename)
00064
00065     yomchi_log = open(YOMCHI_LOG_PATH, "w+")
00066
00067     versions_log = open(VERSIONS_LOG_PATH, "w+")
```

```
00068
00069     if print_the_header:
00070         print_header()
00071
00072     log_versions()
00073
00074
00075 def print_text(text):
00076     """
00077     Print a passed text and save it in the yomchi_log.txt.
00078     @param text: Text to print
00079     @return None
00080     """
00081     global yomchi_log
00082
00083     print(text)
00084     yomchi_log.write(text + "\n")
00085
00086
00087 def print_box(text):
00088     """
00089     Print a passed text in a box and save it in the yomchi_log.txt.
00090     @param text: Text to print
00091     @return None
00092     """
00093     text_length = len(text)
00094     counter = 0
00095     for i in range(0, text_length):
00096         if text[i] == '\n':
00097             counter = 0
00098         else:
00099             counter += 1
00100
00101         if counter == LINE_LENGTH:
00102             if text[i] in ['a', 'e', 'i', 'o', 'u']:
00103                 text = text[:i - 1] + '-' + '\n' + text[i - 1:]
00104             else:
00105                 text = text[:i] + '\n' + text[i:]
00106             counter = 0
00107
00108     text_array = text.splitlines()
00109     print_text("\n|" + "-" * (LINE_LENGTH + 2) + "|")
00110     for line in text_array:
00111         print_text("| " + line.ljust(LINE_LENGTH, ' ') + " |")
00112     print_text("|" + "-" * (LINE_LENGTH + 2) + "|")
00113
00114
00115 def step(description=None, new_step_number=0):
00116     """
00117     Print the current step number with a brief description.
00118     @param description: Description of the step.
00119     @param new_step_number: If defined, reset the steps numeration to the passed value.
00120     @return None
00121     """
00122     global step_number
00123
00124     larger_string = 7
00125     pad = round((LINE_LENGTH - larger_string) / 2)
00126
00127     if new_step_number != 0:
00128         step_number = new_step_number
00129
00130     text = " " * pad + "Step " + str(step_number)
00131     if description is not None:
00132         text = text + "\n" + description
00133
00134     print_box(text)
00135     step_number += 1
00136
00137
00138 def print_parameters(params_dictionary):
00139     """
00140     Print a list of parameters.
00141     @param params_dictionary: Dictionary of parameters {name: value} to print.
00142     @return None
00143     """
00144
00145     pad = round((LINE_LENGTH - 18) / 2)
00146     title = " " * pad + "*** Parameters ***"
00147
00148     names = list(params_dictionary.keys())
00149     values = list(params_dictionary.values())
00150     longest_name = max(names, key=len)
00151
00152     new_names = []
00153     for name in names:
00154         new_names.append(name + " " * (len(longest_name) - len(name)) + " : ")
```

```
00155
00156     new_dictionary = np.char.add(new_names, values)
00157
00158     print_text("\n|" + "-" * (LINE_LENGTH + 2) + "|")
00159     print_text("| " + title.ljust(LINE_LENGTH, ' ') + " |")
00160     print_text("| " + ''.ljust(LINE_LENGTH, ' ') + " |")
00161     for param in new_dictionary:
00162         print_text("| " + param.ljust(LINE_LENGTH, ' ') + " |")
00163     print_text("|" + "-" * (LINE_LENGTH + 2) + "|\n")
00164
00165
00166 def log_versions():
00167     """
00168     Stores the environment versions including packages, libraries and OS.
00169     @return None
00170     """
00171     global versions_log
00172     required_packages = ["Keras", "numpy", "pandas", "matplotlib", "scikit-learn", "seaborn",
00173                          "sklearn", "tensorflow", "pip"]
00174
00175     try:
00176         from pip._internal.operations import freeze
00177     except ImportError:  # pip < 10.0
00178         from pip.operations import freeze
00179
00180     try:
00181         distro = platform.linux_distribution()
00182     except:
00183         distro = ["N/A", ""]
00184
00185     versions_log.write("\n\nOperation System: ")
00186     versions_log.write("\n  Kernel: " + platform.system() + " " + platform.release())
00187     versions_log.write("\n  Distribution: " + distro[0] + " " + distro[1])
00188
00189     versions_log.write("\n\nPython version: " + str(sys.version_info[0]) + "."
00190                        + str(sys.version_info[1]) + "." + str(sys.version_info[2]))
00191     versions_log.write("\n\nPackages versions:\n")
00192
00193     list = freeze.freeze()
00194
00195     versions_log.write("  Required Packages:")
00196     for package in list:
00197         for required in required_packages:
00198             if package.find(required) != -1:
00199                 index = package.find("==")
00200                 package = package.replace("==", " " * (25 - index) + "=  ")
00201                 versions_log.write("\n    " + package)
00202
00203     list = freeze.freeze()
00204     versions_log.write("\n\n  Complete List:")
00205     for package in list:
00206         index = package.find("==")
00207         package = package.replace("==", " " * (25 - index) + "=  ")
00208         versions_log.write("\n    " + package)
00209
00210
00211 def print_header():
00212     """
00213     Print a header in the console output and log files.
00214     @return None
00215     """
00216     larger_string = 33
00217     pad = round((LINE_LENGTH - larger_string) / 2)
00218     header = "#" * LINE_LENGTH + "#" * 3 \
00219              + "\n#" + " " * pad + "                                  " + " " * pad + "#" \
00220              + "\n#" + " " * pad + "    University of Costa Rica     " + " " * pad + "#" \
00221              + "\n#" + " " * pad + "Electrical Engineering Department" + " " * pad + "#" \
00222              + "\n#" + " " * pad + "          Grade Thesis           " + " " * pad + "#" \
00223              + "\n#" + " " * pad + "      Pablo Avila [B30724]       " + " " * pad + "#" \
00224              + "\n#" + " " * pad + "    jose.avilalopez@ucr.ac.cr    " + " " * pad + "#" \
00225              + "\n#" + " " * pad + "                                  " + " " * pad + "#" \
00226              + "\n" + "#" * LINE_LENGTH + "#" * 3 \
00227              + "\n" + " " * pad + "     *** START OF TEST ***  "\
00228              + "\n# File      : " + caller_file \
00229              + "\n# Date      : " + START_TIME.strftime("%m-%d-%Y") \
00230              + "\n# Start Time: " + START_TIME.strftime("%H:%M:%S") + "\n"
00231     print_text(header)
00232     versions_log.write(header)
00233
00234 def finish_test(rename_results_folder=None):
00235     """
00236     Safely finish the test run and log some final information.
00237     @param rename_results_folder: Optional name for the results folder.
00238     @return None
00239     """
00240     global yomchi_log
00241     global versions_log
```

```
00242
00243     larger_string = 33
00244     pad = round((LINE_LENGTH - larger_string) / 2)
00245
00246     finish_time = datetime.datetime.now()
00247     elapsed_time = finish_time - START_TIME
00248
00249     days = elapsed_time.days
00250     hours, rem = divmod(elapsed_time.seconds, 3600)
00251     minutes, seconds = divmod(rem, 60)
00252
00253     footer = "\n" + "#" * LINE_LENGTH + "#" * 3 \
00254              + "\n" + " " * pad + "     *** END OF TEST ***  " \
00255              + "\n# Test Duration : " + str(days) + str(hours) + ":" + str(minutes) + ":" + str(seconds) \
00256              + "\n# Finish Time   : " + finish_time.strftime("%H:%M:%S") \
00257              + "\n" + "#" * LINE_LENGTH + "#" * 3
00258
00259     print_text(footer)
00260
00261     yomchi_log.close()
00262     versions_log.close()
00263
00264     if rename_results_folder is not None:
00265         os.rename(RESULTS_FOLDER, CURRENT_FOLDER + "/" + rename_results_folder + "/")
00266
00267     sys.exit()
```

5.11 /home/pabloav/Documents/Tesis_Lic/LFP-Classification/Yomchi/models.py File Reference

Namespaces

- models

Functions

- def models.mlp (layers, units_per_layer, dropout=None)

  Defines a classical neural network sometimes called: Multi-Layer Perceptron.
- def models.cnn (inputs, units_per_layer, dropout)

  Creates a model of a Convolutional Neural Network with a Conv1D as the input layer, one dense as the only hidden layer and another dense with only 1 neuron as the output layer.
- def models.lstm (units_per_layer, dropout)

  Creates a model of a Long Short-Term Memory Neural Network as the input layer, one dense with only 1 neuron as the output layer.
- def models.compile_and_fit (model, train, val, epochs=20, patience=2)

  Fits the provided training data in the specified model and train's it.

5.11.1 Detailed Description

Author

Pablo Avila [B30724] jose.avilalopez@ucr.ac.cr

Copyright

MIT License

Date

July, 2021

This module contains a set of functions to generate Machine Learning models such as Feed-forward Neural Networks, Long Short-Term Memory Neural Network (LSTM), Convolutional Neural Networks (CNN) and more.

Definition in file models.py.

5.12 models.py

```
00001 # ##########################################################################
00002 #                    University of Costa Rica
00003 #                Electrical Engineering Department
00004 #                           Grade Thesis
00005 # ##########################################################################
00006
00007 """
00008 @file models.py
00009 @author Pablo Avila [B30724] jose.avilalopez@ucr.ac.cr
00010 @copyright MIT License
00011 @date July, 2021
00012 @details This module contains a set of functions to generate Machine Learning models such as Feed-forward
       Neural
00013 Networks, Long Short-Term Memory Neural Network (LSTM), Convolutional Neural Networks (CNN) and more.
00014 """
00015 import Yomchi.Environment as Env
00016 import tensorflow as tf
00017
00018
00019 def mlp(layers, units_per_layer, dropout=None):
00020     """
00021     Defines a classical neural network sometimes called: Multi-Layer Perceptron. It is Feedforward and
       fully-connected,
00022     with the specified parameters. The activation function of the hidden layers is ReLU, and the activation
       of the
00023     output is linear.
00024     @param layers: Number of hidden layers (besides the input and outputs ones).
00025     @param units_per_layer: Number of neurons per layer. Applies to all layers except for the last one
       which has only 1:
00026     the predicted angle.
00027     @param dropout: A value between 0 and 1 of neuron's results to discard of the training. This
       regularization method
00028     will be added to the model. One after the input layer and one before the output layer.
00029     @return model: The model of the MLP created for later usage as the predictor.
00030     """
00031
00032     Env.print_text(f"Creating a fully-connected feed-forward Neural Network model with {layers} layers,
       using "
00033                    f"{units_per_layer} units per layer. \nThe activation function in the hidden layers is '
      ReLu' and "
00034                    f"the output layer has only 1 unit (the predicted angle).")
00035
00036     model = tf.keras.Sequential()
00037     model.add(tf.keras.layers.Flatten())  # Shape: (time, features) => (time*features)
00038     model.add(tf.keras.layers.Dense(units=units_per_layer, activation=tf.nn.relu))  # Input layer
00039     model.add(tf.keras.layers.Dropout(dropout))
00040
00041     for n in range(1, layers):  # Other hidden layers
00042         model.add(tf.keras.layers.Dense(units=units_per_layer, activation=tf.nn.relu))
00043
00044     model.add(tf.keras.layers.Dropout(dropout))
00045     model.add(tf.keras.layers.Dense(units=1))  # Output layer
00046     model.add(tf.keras.layers.Reshape([1, -1]))  # Add back the time dimension. Shape: (outputs) => (1,
       outputs)
00047
00048     return model
00049
00050
00051 def cnn(inputs, units_per_layer, dropout):
00052     """
00053     Creates a model of a Convolutional Neural Network with a Conv1D as the input layer, one dense as the
       only hidden
00054     layer and another dense with only 1 neuron as the output layer. The first two layer has ReLU activation
       and the last
00055     one uses a linear activation function.S
00056     @param inputs: Number of inputs of the network. It is used as the kernel size with the intention that
       the 1D
00057     convolutional layer outputs a single value throughout the specified number of filters.
00058     @param units_per_layer: Specified the number of neurons of the hidden dense layer, which has to match
       with the
```

```
00059     number of filters that will output a result in the 1D convolutional input layer.
00060     @param dropout: A value between 0 and 1 of neuron's results to discard of the training. This
      regularization method
00061     will be added to the model as a layer the output layer.
00062     @return conv_model: The model of the CNN created for later usage as the predictor.
00063     """
00064     Env.print_text(f"Creating a Convolutional Neural Network model with one 1D convolutional layer, using "
00065                    f"{units_per_layer} filters and units in the following dense layer. \nThe activation
      function in "
00066                    f"the dense layer is 'ReLU' and the output dense layer has only 1 unit (the predicted
      angle).")
00067
00068     conv_model = tf.keras.Sequential()
00069     conv_model.add(tf.keras.layers.Conv1D(filters=units_per_layer, kernel_size=(inputs,), activation='relu'
     ))
00070     conv_model.add(tf.keras.layers.Dense(units=units_per_layer, activation='relu'))
00071     conv_model.add(tf.keras.layers.Dropout(dropout))
00072     conv_model.add(tf.keras.layers.Dense(units=1))
00073
00074     return conv_model
00075
00076
00077 def lstm(units_per_layer, dropout):
00078     """
00079     Creates a model of a Long Short-Term Memory Neural Network as the input layer, one dense with only 1
      neuron as the
00080     output layer. The activation functions of the LSTM layer are the regular ones for each of it's gates.
00081     @param units_per_layer: Number of inputs of the network.
00082     @param dropout: A value between 0 and 1 of neuron's results to discard of the training. If
      provided,this
00083     regularization method will be to the LSTM Layer.
00084     @return lstm_model: The model of the LSTM created for later usage as the predictor.
00085     """
00086
00087     Env.print_text(f"Creating a Long Short-term Memory (LSTM) Neural Network using {units_per_layer} units
      per layer."
00088                    f"\nThis network only outputs the final timestamp, giving the model time to warm up its
      internal  "
00089                    f"state before making a single prediction.")
00090
00091     # Shape [batch, time, features] => [batch, time, lstm_units]
00092     lstm_model = tf.keras.models.Sequential()
00093     lstm_model.add(tf.keras.layers.LSTM(units_per_layer, return_sequences=False, dropout=dropout))
00094     lstm_model.add(tf.keras.layers.Dense(units=1))  # Shape => [batch, time, features]
00095
00096     return lstm_model
00097
00098
00099 def compile_and_fit(model, train, val, epochs=20, patience=2):
00100     """
00101     Fits the provided training data in the specified model and train's it. The validation data is used to
      compare the
00102     performance of the model against unknown data. MSE is used as the cost function to train the model and
      MAE as the
00103     performance evaluation metric.
00104     @param model: Model to train. It can be a MLP, CNN or LSTM, among others.
00105     @param train: The dataset for training the model. Must have the shape: (batch, time, features)
00106     @param val: The dataset for validating the model. Must have the shape: (batch, time, features)
00107     @param epochs: Number of iteration over the entire set of data (all the batches).
00108     @param patience: Number of epochs to wait for improvement in the metrics. If there is no notorious
      improvement in
00109     the performance of the validation set after the 'patience' epochs, the training will stop at this
      point.
00110     @return history: The results of the training.
00111     """
00112
00113     Env.print_text(f"Compiling the input model {model.name} with {epochs} epochs. The "
00114                    f"loss function is the MSE and metric to evaluate improvement is the MAE.")
00115
00116     early_stopping = tf.keras.callbacks.EarlyStopping(monitor='val_loss',
00117                                                       patience=patience,
00118                                                       mode='min')
00119
00120     model.compile(loss=tf.losses.MeanSquaredError(),  # tf.losses.mean_absolute_percentage_error()
00121                   metrics=[tf.metrics.MeanAbsoluteError()])
00122
00123     history = model.fit(train, epochs=epochs,
00124                         validation_data=val,
00125                         callbacks=[early_stopping])
00126
00127     return history
```

5.13 /home/pabloav/Documents/Tesis_Lic/LFP-Classification/Yomchi/preprocessing.py File Reference

Namespaces

- preprocessing

Functions

- def preprocessing.load_lfp_data (file=LFP[771], channels=EC014_41_NUMBER_OF_CHANNELS)

  Loads the LFP signals from a .eeg file (LFP Data only f < 625Hz) or a .dat file (LFP Data + Spikes).
- def preprocessing.load_angles_data (file=ANGLES[771], degrees=True)

  Loads the animal position data from a .whl file which contain 2 (x, y) pairs, one for each LED.
- def preprocessing.downsample_lfps (lfp_data, orig_rate, new_rate)

  Downsample the LFP signal data after applying an anti-aliasing filter.
- def preprocessing.angles_expansion (angles_data, orig_rate, new_rate)

  Fill angular data with 'NaN' values to match an expected sampling rate.
- def preprocessing.shortest_angle_interpolation (start, end, amount)

  This interpolation method considers the 'start' and 'end' as angles in a circumference where the objective is to find the smallest arch between two angles.
- def preprocessing.vectorized_sai (angles_data)

  Replace 'NaN' values between valid values (interpolation) in angular data with an interpolated value using the shortest angle interpolation.
- def preprocessing.interpolate_angles (angles_data, method="Shortest Angle")

  Replace 'NaN' values in angular data with an interpolated value using a given method.
- def preprocessing.add_labels (lfps, angles, round_labels, start=0, offset=30)

  Add an additional column to the LFP signals matrix with the angular data used as the labels.
- def preprocessing.clean_invalid_positional (labeled_dataset, is_padded=True)

  Clean the data rows which have '-1' values as labels (angles) from the the data and their LFPs associated in each channel.
- def preprocessing.ndarray_to_dataframe (dataset, rate)

  Converts an n-D Numpy array to a Pandas Dataframe.
- def preprocessing.channels_to_windows (series, channel, window_size, batch_size, shuffle_buffer=None, offset=1)

  Receives a numpy array containing the time series of LFP signals of n channels and returns the same data, separated in windows.
- def preprocessing.average_angles (angles, window_size)

  Receives a numpy array containing the time series of the angles and returns the an set of windows with the average of the 'window_size' angles in each window.

Variables

- preprocessing.PATH_TO_DATASETS = os.path.join(Env.CURRENT_FOLDER, "../Datasets/")
- dictionary preprocessing.LFP
- dictionary preprocessing.ANGLES
- int preprocessing.RAW_DATAMAX_SAMPLING_RATE = 20000
- int preprocessing.RAW_NEURALYNX_SAMPLING_RATE = 32552
- int preprocessing.LFP_DATAMAX_SAMPLING_RATE = 1250
- int preprocessing.LFP_NEURALYNX_SAMPLING_RATE = 1252
- float preprocessing.POSITION_DATA_SAMPLING_RATE = 39.06
- int preprocessing.EC014_41_NUMBER_OF_CHANNELS = 99

5.13.1 Detailed Description

Author

Pablo Avila [B30724] jose.avilalopez@ucr.ac.cr

Copyright

MIT License

Date

July, 2021

This module contains a set of functions to import clean, parse and reshape input data.

Definition in file preprocessing.py.

5.14 preprocessing.py

```
00001 # ##########################################################################
00002 #                    University of Costa Rica
00003 #                Electrical Engineering Department
00004 #                           Grade Thesis
00005 # ##########################################################################
00006
00007 """
00008 @file preprocessing.py
00009 @author Pablo Avila [B30724] jose.avilalopez@ucr.ac.cr
00010 @copyright MIT License
00011 @date July, 2021
00012 @details This module contains a set of functions to import clean, parse and reshape input data.
00013 """
00014
00015 import os
00016 import pandas as pd
00017 import pickle
00018 import scipy.signal
00019 import numpy as np
00020 import tensorflow as tf
00021 import Yomchi.Environment as Env
00022
00023 # Dataset Paths
00024 PATH_TO_DATASETS = os.path.join(Env.CURRENT_FOLDER, "../Datasets/")
00025 LFP = {765: PATH_TO_DATASETS + "ec014.765.eeg",
00026        771: PATH_TO_DATASETS + "ec014.771.eeg"}
00027 ANGLES = {765: PATH_TO_DATASETS + "ec014.765.whl",
00028           771: PATH_TO_DATASETS + "ec014.771.whl"}
00029
00030 # Sampling Rates
00031 RAW_DATAMAX_SAMPLING_RATE   = 20000
00032 RAW_NEURALYNX_SAMPLING_RATE = 32552
00033 LFP_DATAMAX_SAMPLING_RATE   = 1250
00034 LFP_NEURALYNX_SAMPLING_RATE = 1252
00035 POSITION_DATA_SAMPLING_RATE = 39.06
00036 EC014_41_NUMBER_OF_CHANNELS = 99
00037
00038
00039 def load_lfp_data(file=LFP[771], channels=EC014_41_NUMBER_OF_CHANNELS):
00040     """
00041     Loads the LFP signals from a .eeg file (LFP Data only f < 625Hz) or a .dat file (LFP Data + Spikes).
00042     @param file: Path to the file containing the animal LFP data.
00043     @param channels: Number of recorded channels in LFP signals file.
00044     @return lfp: Array (n x channels) with the data. With the columns being the channels and the rows the
00045     a different time step.
00046     """
00047     Env.print_text("Loading LFP Data of " + str(channels) + " channels from: " + os.path.basename(file) + "
      .")
00048
00049     signals = open(file, "rb")
00050     signalsArray = np.fromfile(file=signals, dtype=np.int16)
00051     lfp = np.reshape(signalsArray, (-1, channels))
00052     signals.close()
00053
00054     Env.print_text("LFP Data shape: " + str(np.shape(lfp)))
00055
```

```
00056     return lfp
00057
00058
00059 def load_angles_data(file=ANGLES[771], degrees=True):
00060     """
00061     Loads the animal position data from a .whl file which contain 2 (x, y) pairs, one for each LED. If any
      position
00062     value equals '-1' then it's replaced with 'NaN' instead.
00063     @param file: Path to the file containing the animal LED's position information
00064     @param degrees: If this flag is set, then the angles are returned in degrees from [0, 360[, or radians
      otherwise.
00065     @return angles: Array with the angles in radians extracted from the positions. The angles are given as
00066     float16 values calculated as arctan(y2 - y1 /x2 - x1). Unless the denominator is 0, in that case '0' is
      returned
00067     for that element.
00068     """
00069     Env.print_text("Loading the animal position data from: " + os.path.basename(file) + ".")
00070
00071     positions_file = open(file, "rb")
00072     positions = np.genfromtxt(fname=positions_file, dtype=np.float16, delimiter='\t')
00073     positions_file.close()
00074
00075     positions[positions == -1] = np.NAN
00076     #TODO: of x_2 - x_1 = 0, angles should be NaN.
00077
00078     angles = np.arctan2((np.subtract(positions[:, 3], positions[:, 1])),
00079                         (np.subtract(positions[:, 2], positions[:, 0])))
00080
00081     if degrees:
00082         angles = np.degrees(angles)
00083         angles += 180
00084
00085     angles[np.isnan(angles)] = -1
00086     invalid = np.count_nonzero(angles == -1)
00087
00088     Env.print_text("Head angle data shape: " + str(np.shape(angles)))
00089     Env.print_text(f"Head angle invalid data: {invalid:d} ({invalid/len(angles) * 100:.2f}%)")
00090
00091     return angles
00092
00093
00094 def downsample_lfps(lfp_data, orig_rate, new_rate):
00095     """
00096     Downsample the LFP signal data after applying an anti-aliasing filter.
00097     An order 8 Chebyshev type I filter is used. Usually the LFP signals are acquired at a higher sampling
      rate
00098     than the position data.
00099     @note: This method assumes that the reason of frequencies is 32 to compute the decimation.
00100     @param lfp_data: Matrix [n x numChannels] with the LFP signals
00101     @param orig_rate: Sampling rate originally used to acquire the data.
00102     @param new_rate: New sampling rate of the data.
00103     @return resampled_data: Original data downsampled to the new rate.
00104     """
00105     Env.print_text("Downsampling LFP data to match the new sampling rate: " + str(new_rate)
00106                    + "Hz. Original was: " + str(orig_rate) + "Hz.")
00107
00108     resampled_data = []
00109     for channel_i in np.transpose(lfp_data):
00110         channel_i = scipy.signal.decimate(channel_i, 8)
00111         channel_i = scipy.signal.decimate(channel_i, 4)
00112         resampled_data.append(channel_i)
00113
00114     resampled_data = np.transpose(resampled_data)
00115     Env.print_text("LFP data decimated shape: " + str(np.shape(resampled_data)))
00116
00117     return resampled_data
00118
00119
00120 def angles_expansion(angles_data, orig_rate, new_rate):
00121     """
00122     Fill angular data with 'NaN' values to match an expected sampling rate. Usually the position data is
      acquired at a
00123     lower sampling rate than the LFP signals.
00124     @details Assuming that the acquisition of data started and stopped at the same time, then no data has
      to be added
00125     after the last sample.
00126     @param angles_data: Array with the angles data extracted from the animal positions.
00127     @param orig_rate: Sampling rate originally used to acquire the data.
00128     @param new_rate: New sampling rate of the data. The gaps are filled with 'NaN'.
00129     @return upsampled_data: Original data filled with 'NaN' to match the new sampling rate.
00130     """
00131     Env.print_text("Expanding angles data with 'NaN' values to match the new sampling rate: " + str(
      new_rate) + "Hz. "
00132                    + "Original was: " + str(orig_rate) + "Hz.")
00133
00134     expansion_factor = round(new_rate/orig_rate)
00135     padding = np.full((len(angles_data), expansion_factor - 1), np.NaN)
```

```
00136     upsampled_data = np.concatenate((np.transpose(np.array([angles_data])), padding), axis=1)
00137     upsampled_data = upsampled_data[:-1, :]
00138     upsampled_data = upsampled_data.flatten()
00139 
00140     Env.print_text("Angles data upsampled shape: " + str(np.shape(upsampled_data)))
00141 
00142     return upsampled_data
00143 
00144 
00145 def shortest_angle_interpolation(start, end, amount):
00146     """
00147     This interpolation method considers the 'start' and 'end' as angles in a circumference where the objective is to
00148     find the smallest arch between two angles.
00149     param start: Start angle with values in the range of [0 to 360[
00150     param end: Final angle with values in the range of [0 to 360[
00151     param amount: Value between [0, 1] which determines how close the interpolated angle will be placed from the Start
00152     angle (0) or from the Final angle (1), being 0.5 the middle.
00153     return interpolated_angle: Interpolated angle between 'start' and 'end'.
00154     """
00155     shortest_angle = ((end-start) + 180) % 360 - 180
00156     interpolated_angle = (start + shortest_angle * amount) % 360
00157     return interpolated_angle
00158 
00159 
00160 def vectorized_sai(angles_data):
00161     """
00162     Replace 'NaN' values between valid values (interpolation) in angular data with an interpolated value using
00163     the shortest angle interpolation.
00164     param angles_data: Array with the angles data extracted from the animal positions.
00165     return interpolated_angles: Array with the angles data interpolated using the Shortest Angle Interpolation
00166     """
00167     start_angle = np.nan
00168     no_nans = 0
00169     first_nan_index = 0
00170     interpolated_angles = angles_data
00171 
00172     for i in range(1, len(interpolated_angles)):
00173         # If a valid value followed by NaN: this is the first NaN, start counting
00174         if np.isnan(interpolated_angles[i]) and not np.isnan(interpolated_angles[i-1]):
00175             start_angle = interpolated_angles[i-1]
00176             no_nans = 1
00177             first_nan_index = i
00178 
00179         # If a NaN followed by another NaN: Increment counter 1+.
00180         elif np.isnan(interpolated_angles[i]) and np.isnan(interpolated_angles[i - 1]):
00181             no_nans += 1
00182 
00183         # If a NaN followed by a valid value: This is the last NaN, interpolate.
00184         elif not np.isnan(interpolated_angles[i]) and np.isnan(interpolated_angles[i - 1]):
00185             if no_nans > 0 and not np.isnan(start_angle):
00186                 amount = 0
00187                 end_angle = interpolated_angles[i]
00188                 for j in range(first_nan_index, first_nan_index + no_nans):
00189                     amount += 1/(no_nans + 1)
00190                     interpolated_angles[j] = shortest_angle_interpolation(start_angle, end_angle, amount)
00191                 start_angle = np.nan
00192 
00193     return interpolated_angles
00194 
00195 
00196 def interpolate_angles(angles_data, method="Shortest Angle"):
00197     """
00198     Replace 'NaN' values in angular data with an interpolated value using a given method.
00199     @param angles_data: Array with the angles data extracted from the animal positions.
00200     @param method: Interpolation method to fill the gaps in the data. The optional methods available are the supported
00201     by pandas.DataFrame.interpolate function, which are: 'linear', 'quadratic', 'cubic', 'polynomial', among others.
00202     @return interpolated_angles: Array with the angles data interpolated using the given method.
00203     """
00204 
00205     if np.count_nonzero(np.isnan(angles_data)) > 0:
00206         Env.print_text("Interpolate angles data using " + method + " method.")
00207 
00208         if method == "Shortest Angle":
00209             interpolated_angles = vectorized_sai(angles_data)
00210 
00211         else:
00212             angles_series = pd.Series(angles_data)
00213             interpolated_angles = angles_series.interpolate(method)
00214             interpolated_angles = interpolated_angles.to_numpy()
00215     else:
```

```
00216         Env.print_text("Skipping Interpolation of angles since the angles weren't padded.")
00217         interpolated_angles = angles_data
00218     return interpolated_angles
00219
00220
00221 def add_labels(lfps, angles, round_labels, start=0, offset=30):
00222     """
00223     Add an additional column to the LFP signals matrix with the angular data used as the labels.
00224     @param lfps: Matrix [n x numChannels] with the LFP signals used as the preliminary features of the
      data.
00225     @param angles: Array with the angles data extracted from the positions used as the labels of the data.
00226     @param round_labels: Boolean, if true the labels are rounded to angles multiples of 'offset' starting
      from 'start'
00227     @param start: Angle in [0°, 360°[ used as first label.
00228     @param offset: Offset in [1°, 360°[ between labels starting from 'start' angle.
00229     @return labeled_data: Matrix with the labeled data [n x lfps[numChannels], angles].
00230     """
00231
00232     len_lfps = len(lfps)
00233     len_angles = len(angles)
00234
00235     if len_lfps > len_angles:
00236         lfps = lfps[0: len_angles]
00237     elif len_lfps < len_angles:
00238         angles = angles[0: len_lfps]
00239
00240     Env.print_text("Adding labels to the data by concatenating the [" + str(len(lfps)) + " x " + str(len(
      lfps[0])) +
00241                    "] LFP data matrix with the [" + str(len(angles)) + "] Angles vector.")
00242     Env.print_text("Rounding Labels = " + str(round_labels))
00243
00244     if round_labels:
00245         if (0 <= start < 360) and (1 <= offset < 360):
00246             labels = np.arange(start, 360, offset)
00247
00248         for i in range(0, len(angles)):
00249             if not np.isnan(angles[i]):
00250                 minval = np.inf
00251                 label = start
00252                 for tag in labels:
00253                     diff = abs(angles[i] - tag)
00254                     if diff < minval:
00255                         minval = diff
00256                         label = tag
00257                 angles[i] = label
00258
00259     labeled_data = np.concatenate((lfps, angles), axis=1)
00260
00261     return labeled_data
00262
00263
00264 def clean_invalid_positional(labeled_dataset, is_padded=True):
00265     """
00266     Clean the data rows which have '-1' values as labels (angles) from the the data and their LFPs
      associated in each
00267     channel. Plus the following 31 rows with NaN as angle value in case of padded data.
00268     @details The positional data taken from the LEDs placed in the rat have discontinuities where one or
      both LEDs
00269     are lost, making them invalid.
00270     Hence '-1' values are used instead to denote invalid position data and are meant to be removed from the
      data since
00271     they are not representative labels.
00272     @param labeled_dataset: Matrix [n x (numChannels +1)] with the LFP signals used as the preliminary
      features of the
00273     data and the angles data extracted from the positions used as the labels of the data.
00274     @param is_padded: If True, manage the input labeled dataset as a padded array of angles, or a
      downsampled LFP set
00275      otherwise.
00276     @return clean_dataset: Input data without invalid positional values.
00277     """
00278     if is_padded:
00279         # Get the indexes of all invalid values (-1) plus the 31 following padded values (NaN).
00280         # and save that range as an element of 'invalid_indexes' array.
00281         invalid_indexes = [np.arange(i, i + 32).tolist() for i, v in enumerate(labeled_dataset) if v[-1] ==
      -1]
00282         Env.print_text(f"Amount of invalid indexes: {len(invalid_indexes)}")
00283
00284         # Merge all sublists as a single consecutive array of invalid indexes.
00285         invalid_indexes = [item for sublist in invalid_indexes for item in sublist]
00286         Env.print_text(f"Amount of invalid indexes + associated expanded indexes: {len(invalid_indexes)}")
00287         Env.print_text(f"Number of valid samples: {len(labeled_dataset) - len(invalid_indexes)} (including
      the 31 NaN "
00288                        f"values padded for the last real positional data.)")
00289     else:
00290         # Get the indexes of all invalid values (-1) and save that range as an element of 'invalid_indexes'
      array.
00291         invalid_indexes = [i for i, v in enumerate(labeled_dataset) if v[-1] == -1]
```

```
00292         Env.print_text(f"Amount of invalid indexes: {len(invalid_indexes)}")
00293         Env.print_text(f"Number of valid samples: {len(labeled_dataset) - len(invalid_indexes)}")
00294 
00295     # Get the indexes where the discontinuities start and end.
00296     discontinuities_starts = [invalid_indexes[0]]
00297     discontinuities_ends = []
00298     for i in range(1, len(invalid_indexes)):
00299         if invalid_indexes[i] - 1 != invalid_indexes[i-1]:
00300             discontinuities_starts.append(invalid_indexes[i])
00301             discontinuities_ends.append(invalid_indexes[i-1])
00302     discontinuities_ends.append(invalid_indexes[-1])
00303 
00304     # Stores sub-arrays of valid indexes:
00305     clean_datasets = []
00306 
00307     # From the first item to the start of the first discontinuity
00308     if discontinuities_starts[0] != 0:
00309         clean_datasets.append(labeled_dataset[0:discontinuities_starts[0], :])
00310 
00311     # From the end+1 of the ith discontinuity to the start (not included) of the ith + 1 discontinuity
00312     for i in range(len(discontinuities_starts)-1):
00313         clean_datasets.append(labeled_dataset[discontinuities_ends[i]+1:discontinuities_starts[i+1], :])
00314 
00315     # From the end of the last discontinuity to the last item.
00316     if discontinuities_ends[-1] != len(labeled_dataset)-1:
00317         clean_datasets.append(labeled_dataset[discontinuities_ends[-1] + 1:-1, :])
00318 
00319     if is_padded:
00320         # Delete subsets with only 1 valid sample (i.e its length is <=32)
00321         # From the resulting subsets, delete the last 31 rows with NaN angles
00322         clean_datasets = [v[:-31, :] for v in clean_datasets if len(v) > 32]
00323     else:
00324         clean_datasets = [v[:, :] for v in clean_datasets if len(v) > 2]
00325 
00326     Env.print_text(f"Total number of valid subsets: {len(clean_datasets)}")
00327     for s in range(0, len(clean_datasets)):
00328         Env.print_text(f"Total number of samples and channels + angles in subsets {s}:
    {clean_datasets[s].shape}")
00329 
00330     return clean_datasets
00331 
00332 
00333 def ndarray_to_dataframe(dataset, rate):
00334     """
00335     Converts an n-D Numpy array to a Pandas Dataframe
00336     @param dataset: Matrix [n x (numChannels +1)] with the LFP signals used as the preliminary features
00337     of the data and the angles data extracted from the positions used as the labels of the data.
00338     @param rate:
00339     @return dataframe: Pandas data frame with Channels 0-99 and Angles as columns names, and the timestamp
     as indexes
00340     calculated as 1/rate * 1e6 to get the time step of the acquisition in microseconds.
00341     """
00342 
00343     columns = []
00344     for i in range(0, 99):
00345         columns.append(f"Channel {i}")
00346     columns.append("Angles")
00347 
00348     time_step = round((1/rate) * 1e6)   # 1/f [us]: 1250Hz => 800us, 39.06Hz => 2560.164us
00349     indeces = np.arange(0, len(dataset) * time_step, time_step)
00350     dataframe = pd.DataFrame(data=dataset, columns=columns, index=indeces)
00351 
00352     return dataframe
00353 
00354 
00355 def channels_to_windows(series, channel, window_size, batch_size, shuffle_buffer=None,
    offset=1):
00356     """
00357     Receives a numpy array containing the time series of LFP signals of n channels and returns the same
     data, separated
00358     in windows.
00359     @param series: Numpy Array with the LFP data of the n channels.
00360     @param channel: Channel to use.
00361     @param window_size: Size of the windows in which the data are being split.
00362     @param batch_size: Number of pairs data-labels to group as a batch
00363     @param shuffle_buffer: Number of windows to shuffle at the same time.
00364     @param offset: Number of samples to shift between windows.
00365     @return windowed_ds: LFP data of the selected channel separated in windows.
00366     """
00367 
00368     labels = np.expand_dims(series[window_size::, -1], 1)
00369     data = series[:, channel]
00370 
00371     # Get the average angle for each window.
00372     #labels_ds = average_angles(labels, window_size)
00373 
00374     labels_ds = tf.data.Dataset.from_tensor_slices(np.expand_dims(labels, 1))
```

```
00375
00376     # Creates a dataset from the input
00377     windowed_data = tf.data.Dataset.from_tensor_slices(np.expand_dims(data, 1))
00378
00379     # Split the data set in windows shifting each window by 1 and forcing them to the same size
      (window_size + 1)
00380     windowed_data = windowed_data.window(window_size, shift=offset, drop_remainder=True)
00381
00382     # Make each window a numpy array row.
00383     windowed_data = windowed_data.flat_map(lambda window: window.batch(window_size))
00384
00385     # Add labels to the data
00386     windowed_ds = tf.data.Dataset.zip((windowed_data, labels_ds))
00387
00388     if Env.debug:
00389         for x, y in windowed_ds.take(1):
00390             print("x = ", x.numpy())
00391             print("y = ", y.numpy())
00392
00393     if shuffle_buffer != None:
00394         # Shuffle the data in groups of shuffle_buffer to accelerate. Instead of shuffle it all at once.
00395         windowed_ds = windowed_ds.shuffle(shuffle_buffer)
00396
00397     # Batch the data into sets of 'batch_size'.
00398     windowed_ds = windowed_ds.batch(batch_size).prefetch(1)
00399
00400     return windowed_ds
00401
00402
00403 def average_angles(angles, window_size):
00404     """
00405     Receives a numpy array containing the time series of the angles and returns the an set of windows with
      the average
00406     of the 'window_size' angles in each window. Each window is 1 element shifted from the previous window.
00407     @param angles: Numpy Array with the Angles data to use as the labels.
00408     @param window_size: Size of the windows in which the data are being split.
00409     @return average_angles: Averaged Angles data separated in windows.
00410     """
00411
00412     # Get an array of labels from the series
00413     windowed_angles = tf.data.Dataset.from_tensor_slices(angles)
00414
00415     # Split the data set in windows shifting each window by 1 and forcing them to the same size
      (window_size + 1)
00416     windowed_angles = windowed_angles.window(window_size, shift=1, drop_remainder=True)
00417
00418     # Make each window a numpy array row.
00419     windowed_angles = windowed_angles.flat_map(lambda window: window.batch(window_size))
00420     averaged_angles = windowed_angles.map(lambda window: tf.math.reduce_mean(window))
00421
00422     if Env.debug:
00423         print("Angles:")
00424         for window in windowed_angles.take(1):
00425             print(window.numpy())
00426         for window in averaged_angles.take(1):
00427             print(window.numpy())
00428
00429     return averaged_angles
00430
```

5.15 /home/pabloav/Documents/Tesis_Lic/LFP-Classification/Yomchi/visualization.py File Reference

Namespaces

- visualization

Functions

- def visualization.store_figure (fig_name, show=False)

  Stores a figure.

Variables

- string visualization.FIG_FORMAT = 'png'

5.15.1 Detailed Description

Author

Pablo Avila [B30724] jose.avilalopez@ucr.ac.cr

Copyright

MIT License

Date

July, 2021

This module contains a set of functions to plot and in general terms, visualize data.

Definition in file visualization.py.

5.16 visualization.py

```
00001 # ###########################################################################
00002 #                   University of Costa Rica
00003 #               Electrical Engineering Department
00004 #                          Grade Thesis
00005 # ###########################################################################
00006
00007 """
00008 @file visualization.py
00009 @author Pablo Avila [B30724] jose.avilalopez@ucr.ac.cr
00010 @copyright MIT License
00011 @date July, 2021
00012 @details This module contains a set of functions to plot and in general terms, visualize data.
00013 """
00014
00015 import matplotlib.pyplot as plt
00016 import Yomchi.Environment as Env
00017
00018
00019 # Matplotlib figures parameters
00020 FIG_FORMAT = 'png'
00021 plt.rcParams['figure.dpi'] = 200
00022 plt.rcParams['figure.figsize'] = [12, 8]
00023 plt.rcParams['savefig.format'] = FIG_FORMAT
00024 plt.rcParams['lines.markersize'] = 1
00025 plt.rcParams['font.family'] = 'serif'
00026 plt.rcParams['font.size'] = 14
00027 plt.rcParams['figure.autolayout'] = True
00028 plt.rcParams['axes.titlepad'] = 7
00029 plt.rcParams['axes.grid'] = False
00030
00031
00032 def store_figure(fig_name, show=False):
00033     """
00034     Stores a figure.
00035     @param fig_name: Name of the figure to store.
00036     @param show: Displays the figure when ready. Warning: Stalls execution until closing it.
00037     @return None
00038     """
00039
00040     plt.figure(fig_name)
00041     plt.savefig(fname=Env.CAPTURES_FOLDER + fig_name + "." + FIG_FORMAT)
00042     if show:
00043         plt.show()
00044     plt.close(fig_name)
```

Index

/home/pabloav/Documents/Tesis_Lic/LFP-Classification/↩debug
DataExploration/DataExploration.py, 27
/home/pabloav/Documents/Tesis_Lic/LFP-Classification/↩downsample_lfps
DatasetGenerator/DatasetGenerator.py, 32, 33
/home/pabloav/Documents/Tesis_Lic/LFP-Classification/↩EC014_41_NUMBER_OF_CHANNELS
Predictor/Predictor.py, 36
/home/pabloav/Documents/Tesis_Lic/LFP-Classification/↩Environment, 9
README.md, 39
/home/pabloav/Documents/Tesis_Lic/LFP-Classification/↩
Yomchi/Environment.py, 40, 42
/home/pabloav/Documents/Tesis_Lic/LFP-Classification/↩
Yomchi/models.py, 45, 46
/home/pabloav/Documents/Tesis_Lic/LFP-Classification/↩
Yomchi/preprocessing.py, 48, 49
/home/pabloav/Documents/Tesis_Lic/LFP-Classification/↩
Yomchi/visualization.py, 54, 55

ANGLES
preprocessing, 24
add_labels
preprocessing, 19
angles_expansion
preprocessing, 20
average_angles
preprocessing, 20

CAPTURES_FOLDER
Environment, 13
CURRENT_FOLDER
Environment, 13
caller_file
Environment, 13
channels_to_windows
preprocessing, 20
clean_invalid_positional
preprocessing, 21
cnn
models, 15
compile_and_fit
models, 16

data_exploration
DataExploration, 7
data_set_generator
DatasetGenerator, 8
DataExploration, 7
data_exploration, 7
DatasetGenerator, 8
data_set_generator, 8
debug
Environment, 13
downsample_lfps
preprocessing, 21
EC014_41_NUMBER_OF_CHANNELS
preprocessing, 24
Environment, 9
CAPTURES_FOLDER, 13
CURRENT_FOLDER, 13
caller_file, 13
debug, 13
finish_test, 10
init_environment, 10
LINE_LENGTH, 13
LOGS_FOLDER, 14
log_versions, 11
print_box, 11
print_header, 11
print_parameters, 12
print_text, 12
RESULTS_FOLDER, 14
START_TIME, 14
step, 12
step_number, 14
VERSIONS_LOG_PATH, 14
versions_log, 14
YOMCHI_LOG_PATH, 15
yomchi_log, 14

FIG_FORMAT
visualization, 26
finish_test
Environment, 10

init_environment
Environment, 10
interpolate_angles
preprocessing, 22

LFP_DATAMAX_SAMPLING_RATE
preprocessing, 25
LFP_NEURALYNX_SAMPLING_RATE
preprocessing, 25
LFP
preprocessing, 25
LINE_LENGTH
Environment, 13
LOGS_FOLDER
Environment, 14

load_angles_data
    preprocessing, 22
load_lfp_data
    preprocessing, 23
log_versions
    Environment, 11
lstm
    models, 16

mlp
    models, 17
models, 15
    cnn, 15
    compile_and_fit, 16
    lstm, 16
    mlp, 17

ndarray_to_dataframe
    preprocessing, 23

PATH_TO_DATASETS
    preprocessing, 25
POSITION_DATA_SAMPLING_RATE
    preprocessing, 25
Predictor, 17
    predictor, 17
predictor
    Predictor, 17
preprocessing, 18
    ANGLES, 24
    add_labels, 19
    angles_expansion, 20
    average_angles, 20
    channels_to_windows, 20
    clean_invalid_positional, 21
    downsample_lfps, 21
    EC014_41_NUMBER_OF_CHANNELS, 24
    interpolate_angles, 22
    LFP_DATAMAX_SAMPLING_RATE, 25
    LFP_NEURALYNX_SAMPLING_RATE, 25
    LFP, 25
    load_angles_data, 22
    load_lfp_data, 23
    ndarray_to_dataframe, 23
    PATH_TO_DATASETS, 25
    POSITION_DATA_SAMPLING_RATE, 25
    RAW_DATAMAX_SAMPLING_RATE, 25
    RAW_NEURALYNX_SAMPLING_RATE, 25
    shortest_angle_interpolation, 24
    vectorized_sai, 24
print_box
    Environment, 11
print_header
    Environment, 11
print_parameters
    Environment, 12
print_text
    Environment, 12

RAW_DATAMAX_SAMPLING_RATE
    preprocessing, 25
RAW_NEURALYNX_SAMPLING_RATE
    preprocessing, 25
RESULTS_FOLDER
    Environment, 14

START_TIME
    Environment, 14
shortest_angle_interpolation
    preprocessing, 24
step
    Environment, 12
step_number
    Environment, 14
store_figure
    visualization, 26

VERSIONS_LOG_PATH
    Environment, 14
vectorized_sai
    preprocessing, 24
versions_log
    Environment, 14
visualization, 26
    FIG_FORMAT, 26
    store_figure, 26

YOMCHI_LOG_PATH
    Environment, 15
yomchi_log
    Environment, 14