UNIVERSIDAD DE CONCEPCIÓN
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE GEOFÍSICA

DETECCIÓN DE EVENTOS SÍSMICOS EN VOLCÁN COPAHUE MEDIANTE RED NEURONAL ARTIFICIAL

POR
MARTÍN ALEJANDRO SEPÚLVEDA VERGARA

Tesis presentada al Departamento de Geofísica de la Facultad de Ciencias Físicas y Matemáticas de la Universidad de Concepción para optar al título profesional de Geofísico.

Profesores Guía: Dr. Matthew Miller, Dr. Diego González

Marzo, 2021

Concepción, Chile

UNIVERSIDAD DE CONCEPCIÓN
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE GEOFÍSICA

DETECCIÓN DE EVENTOS SÍSMICOS EN VOLCÁN COPAHUE MEDIANTE RED NEURONAL ARTIFICIAL

POR
MARTÍN ALEJANDRO SEPÚLVEDA VERGARA

Tesis presentada al Departamento de Geofísica de la Facultad de Ciencias Físicas y Matemáticas de la Universidad de Concepción para optar al título profesional de Geofísico.

Profesores Guía: Dr. Matthew Miller, Dr. Diego González
Comisión de evaluación: Dr (c). Felipe Quiero

Marzo, 2021
Concepción, Chile

AGRADECIMIENTOS

Me gustaría agradecer a mis padres y hermano, por su preocupación, la constante motivación, el cariño y fe en mí a lo largo de este proceso. A Franchesca, por su amor infinito, su confianza, gran compañía y apoyo incondicional. A mis amigos, compañeros, cercanos y conocidos que formaron parte de este largo viaje.

Agradecer también a Matt y Diego, por su guía, paciencia y comprensión durante todo este tiempo. A Felipe, por aportar su conocimiento cuando lo necesité, y a todos los profesores, por sus enseñanzas e inspiración.

Índice general

Índice de tablas

Índice de figuras

Resumen

A medida que avanza el tiempo y la tecnología, y se instala un mayor número de estaciones, el volumen y variedad de los registros sísmicos aumenta considerablemente. Esto ha instaurado la necesidad de usar algoritmos automáticos de detección de sismos, los cuales son abundantes y diversos, y con distintos enfoques. Por otro lado, el *Machine-learning* y Redes neuronales se han desarrollado al punto de estar presente en muchas ramas de las ciencias, e incluso, nuestra vida cotidiana. Dada la especialidad de esta rama de la Inteligencia Artificial en el reconocimiento de patrones, y con la información necesaria para la discriminación de eventos, se puede desarrollar una red neuronal dedicada a la detección de sismos dentro de las trazas.

Así, en esta tesis se desarrolla una red neuronal, cuya función es la detección automática de eventos sísmicos, basado en datos de STA/LTA de los sismogramas. Para esta tarea, se utiliza una red de tipo *Multi-layered Perceptron*, con 50 neuronas de entrada, una capa oculta variable, y dos neuronas de salida. Para el entrenamiento, se usa *Mini-batch Gradient Descent* mientras que para la función de coste y función de salida, se recurre a *Cross-entropy* y *Softmax* respectivamente, utilizando, además, funciones de activación como ReLU, Sigmoide y Tangente Hiperbólica (Tanh). Tras la generación de múltiples modelos, variando las funciones de activación, estandarización de datos y número de neuronas ocultas, se encuentra que una de las mejores versiones consigue un 11.8 % de las detecciones y falsos positivos, comparado al método de STA/LTA, identificando igual número de sismos catalogados. Además, se muestra que un mayor número de neuronas y el uso de función Sigmoide entregan mejores resultados, mientras que para la estandarización de datos no se logra identificar mayor influencia. Dado lo anterior, se observa un impacto positivo al utilizar *Machine-learning* en una aplicación de Sismología como la detección de eventos en registros sísmicos, posibilitando y sugiriendo un más amplio y profundo uso en el futuro.

1. Introducción

Machine-learning, o Aprendizaje automático en español, es entendido como la rama de las ciencias computacionales que se evoca al desarrollo de técnicas o sistemas que aprenden automáticamente a través de los datos, esto mediante la identificación de patrones en ellos, para luego realizar predicciones, clasificaciones, entre otras aplicaciones (Mitchell et al., 1997). Actualmente es un área de gran importancia, de carácter transversal, que se utiliza en una variedad de campos científicos, así como en la vida cotidiana de la población ((Shinde and Shah, 2018), (Murphy, 2012), (Jordan and Mitchell, 2015)).

Uno de los subcampos del *Machine-learning* corresponde al de redes neuronales artificiales, las cuales se basan en la arquitectura de las neuronas biológicas, simulando su entramado e interconexión para lograr realizar tareas que son complejas de llevar a cabo utilizando unidades simples (neuronas individuales en este caso). Esta rama es una de las más destacadas y conocidas actualmente, debido a la facilidad de su implementación y su amplia variedad de tipos de redes, desarrollándose modelos cada vez más complejos para la solución de problemas de clasificación e identificación de patrones de mayor envergadura ((Bishop, 2006), (Géron, 2017)).

Por otro lado, la Sismología es la ciencia encargada del estudio de los terremotos y las ondas elásticas o señales sísmicas que estos generan, y que se propagan a través de la Tierra (Shearer, 2019). Uno de los objetivos fundamentales del estudio de señales sísmicas en Sismología es la detección de eventos, ya que esto produce datos básicos para investigaciones más complejas, como el desarrollo de catálogos, localización, tomografías sísmicas, etc. Además, la identificación y localización de sismos aborda el aspecto de riesgo sísmico directamente, al relacionarse con la cantidad, características, distribución y evolución espacio-temporal de los eventos detectados y réplicas, y así identificar zonas de enjambres, o monitorear procesos volcánicos (Dai and MacBeth, 1995).

Basado en lo anterior, y dada la creciente cantidad de datos, registros y estaciones instaladas, es que se utilizan métodos automáticos de detección de sismos, con el objetivo de procesar estos datos de forma eficiente y rápida, en apoyo a los sismólogos/as que analizan estos registros (Wang and Teng, 1995). De estos algoritmos automatizados existe una gran variedad, con algunos basándose en auto-correlación, auto-similaridad, comparación de plantillas o impulsividad, siendo este último el más utilizado, particularmente el método de STA/LTA. Poniendo mayor énfasis a este último caso, se observa que la metodología del STA/LTA ha entregado buenos resultados, acumulando una gran variedad de versiones con distintas especializaciones ((Sharma et al., 2010),(Withers et al., 1998)). Dentro de todo esto, se ve la posibilidad de generar una versión basada en la optimización de los parámetros que componen al método o la información que el algoritmo entrega, con el objetivo de reducir la cantidad de falsos positivos a la vez de mantener o aumentar su capacidad de detección y clasificación de eventos sísmicos.

Así, se explora la posibilidad de mejorar un sistema de detección como el STA/LTA, sabiendo que la identificación de eventos dentro de registros sísmicos es interpretable como un problema de reconocimiento de patrones y/o clasificación (Zhao and Takano, 1999). Junto con lo anterior, se ha encontrado en la literatura la aplicación de detección e incluso clasificación de señales sísmicas, mediante *Machine-learning* y redes neuronales ((Ibs-von Seht, 2008),(Curilem et al., 2009),(Ruano et al., 2013),(Kortström et al., 2016)). Dados estos antecedentes, se propone para este trabajo el desarrollo de una red neuronal que detecte eventos en los registros sismológicos, basándose particularmente en los datos de STA/LTA de la misma traza, para luego comparar el desempeño de estas metodologías y evaluar el uso de *Machine-learning* como una mejora por sobre el algoritmo STA/LTA.

Para lograr lo anteriormente mencionado, se utilizará el catálogo y datos sismológicos de la estación COPA del Centro de Monitoreo Volcánico de la Universidad de Concepción, los cuales serán seleccionados y separados en muestras de eventos y ruido sísmico, y así luego ser ingresados a la red neuronal, correspondiente al tipo *Multi-layered Perceptron* (Gardner and Dorling, 1998). Para su entrenamiento, se utilizará como muestras secciones de 2 segundos de STA/LTA obtenidos de las trazas, como también los métodos de *Mini-Batch*

Gradient Descent para el aprendizaje, *Cross-entropy* como función de coste y *Softmax* como función de salida (Géron, 2017). Debido a que aspectos de la arquitectura de la red quedarán a elección del usuario, como las funciones de activación y la capa oculta, es que se realizará un proceso de generación, selección y comparación de modelos variando estos distintos factores. Concretamente, la selección se basará en la comparación de cada modelo obtenido, apoyándose en métricas de error y exactitud, así como matrices de confusión y otras mediciones. Finalmente, los modelos que resten serán puestos a prueba frente a trazas continuas correspondientes a muestras de eventos, ruido y enjambre sísmico, obteniendo así un poco de estadística sobre el éxito de la red, como falsos positivos, frente a un detector basado en impulsividad o STA/LTA, y el catálogo proporcionado por el Centro de Monitoreo Volcánico.

Cabe notar que, los resultados de una aplicación de redes neuronales en este tipo de área, pueden tener un impacto significativo en el aspecto de la capacidad de detección de una red sismológica y falsos positivos, además de que se facilita la actualización del sistema, dado que ya no es necesario encontrar los parámetros ideales nuevamente, sino que basta con entrenar una vez más la red, con nuevos datos que proporcionen mejores características de lo que se va a identificar. Así, utilizar un sistema de estas características tiene otros aspectos positivos como su fácil implementación y versatilidad (Kong et al., 2019).

Dicho esto, la presente tesis proseguirá con la siguiente estructura. En primer lugar, en Marco Teórico y Metodología se presentarán los conceptos teóricos o bases sobre las que se trabajará, además de los pasos realizados en detalle para la obtención de los resultados, respectivamente. Por otro lado, en Resultados se mostrarán los productos de la metodología presentada, mientras que finalmente, en Discusión y Conclusiones se mencionarán y discutirán distintos aspectos de la investigación, sus resultados, métodos utilizados y comparación con otros trabajos en la literatura, planteando además si se cumplió la hipótesis y los objetivos relacionados a esta.

2. Hipótesis y objetivos

2.1. Hipótesis

Existe una variedad de estudios que han evidenciado e implementado la detección o clasificación automática de señales sísmicas (por ejemplo, (Ibs-von Seht, 2008), (Wang and Teng, 1995), (Curilem et al., 2009), (Tiira, 1999), (Dai and MacBeth, 1995)) mediante la extracción de distintos parámetros de las formas de onda (sobre dimensión temporal y/o frecuencial) y utilizando diversos tipos de redes neuronales, ya sea aplicándolos en redes sismológicas locales, regionales o instaladas en entornos volcánicos. Esto establece un precedente bastante sólido que abre un abanico de posibilidades a la hora de aplicar *Machine-learning* a un caso de estudio.

Entonces, se propone como hipótesis para este trabajo que, mediante la extracción de parámetros que puedan caracterizar y diferenciar eventos de ruido sísmico, específicamente el STA/LTA calculado de los registros sísmicos, es posible detectar/clasificar señales o sismos dentro de las trazas mediante la implementación de una red neuronal de tipo *Multi-layered Perceptron*.

2.2. Objetivos

2.2.1. Objetivo general

El objetivo principal para esta tesis corresponde a lograr la óptima detección/clasificación de las señales sísmicas que arriban a la estación instalada en las cercanías del Volcán Copahue, denominada COPA, mediante el desarrollo de un detector de eventos en base a la red neuronal *Multi-Layered Perceptron* (MLP) y a partir de la información proporcionada por las formas de onda, específicamente el STA/LTA obtenido de estas mismas.

2.2.2. Objetivos específicos

- Realizar una recopilación de eventos y ruido sísmico para el desarrollo de una base de datos inicial.
- Pre-procesar y obtener parámetros a partir de la base de datos, correspondientes a ambas clases de datos (eventos y ruido).
- Entrenar una variedad de versiones de la red neuronal a partir de la base de datos y los parámetros calculados.
- Evaluar y comparar el desempeño de las distintas versiones de la red neuronal.
- Seleccionar las mejores versiones de la red mediante métricas de desempeño.
- Comprobar mediante test con trazas continuas el desempeño de las versiones finales de la red.
- Calculo de estadísticas simples para mostrar el desempeño frente al algoritmo STA/LTA original.

3. Marco Teórico

Para este estudio se trabajó principalmente con redes neuronales, particularmente con *Multi-layered Perceptron*, lo cual forma parte de la amplia área de conocimiento llamada *Machine-learning*. Por otro lado, se desarrolló y manejó una base de datos para la obtención de parámetros, lo cual abarca ciertos aspectos teóricos que serán definidos en esta sección de la investigación, en conjunto a aquellos relacionados con *Machine-learning*.

3.1. Base de datos

El desarrollo de una base de datos que fuese útil y efectiva para este trabajo constó de dos principales aspectos: el manejo y procesamiento de los datos, lo cual fue logrado con la librería ObsPy de Python, y por otro lado la obtención de un producto final que sirviera de parámetro de entrada para la red, esto mediante el cálculo de STA/LTA sobre los datos de señales registradas.

3.1.1. ObsPy

ObsPy corresponde a un software de código abierto desarrollado en Python para el procesamiento de datos sismológicos. Esta librería tiene como objetivo proporcionar una plataforma versátil, rápida y accesible para el desarrollo de aplicaciones y herramientas a científicos/as y sismólogos/as, y que gracias a la integración de otras librerías del ambiente de Python como NumPy o SciPy, permite trabajar fácilmente con todo tipo de formatos y herramientas ((Beyreuther et al., 2010), (Megies et al., 2011), (Krischer et al., 2015)).

Para el caso de estudio, la utilidad de ObsPy recaerá en su función de manejo y procesamiento de los datos, permitiendo leer, cortar y guardar distintas series de tiempo para la generación de bases de datos, además del uso de funcionalidades propias de la librería como *Triggering*, filtrado, STA/LTA, etc. De estos últimos, el más importante corresponde al método de STA/LTA, que será definido en la siguiente sección.

3.1.2. STA/LTA

El STA/LTA es un algoritmo ampliamente utilizado en sismología dada su simple implementación y eficiencia computacional, su nombre proviene del inglés *short-term average/long-term average* y corresponde a un algoritmo que calcula la razón entre el promedio de amplitud sobre una ventana móvil corta y una ventana móvil larga de datos sísmicos, con estas ventanas propagándose simultáneamente a través de los datos en el dominio temporal. La ventana móvil corta es sensible a rápidas variaciones de amplitud o impulsos que pueden corresponder a eventos sísmicos, mientras que la ventana larga es una medida representativa del ruido sísmico de fondo, por lo que la razón entre ambas ventanas muestra una medida de la impulsividad de la señal contrastada con el variante ruido de fondo ((Sharma et al., 2010); (Withers et al., 1998); (Trnkoczy, 2009))

Al ser este un método de detección de eventos muy utilizado de hace varias décadas, se han desarrollado distintas variaciones. En particular, para este trabajo se utilizará la versión denominada *STA/LTA recursiva*, la cual ser más eficiente y suave que la versión clásica del algoritmo al utilizar una respuesta exponencial (Withers et al., 1998), además de que es usada por defecto en una variedad de rutinas y librerías especializadas en detección de eventos, como también es utilizada por (Wang and Teng, 1995) en su investigación.

Para el cálculo de STA/LTA, se definen las ventanas móviles de la siguiente manera:

$$STA_i = Cx_i + (1 - C)\mathrm{STA}_{i-1} \tag{3.1.1}$$

$$LTA_i = Cx_i + (1 - C)\mathrm{LTA}_{i-1} \tag{3.1.2}$$

$$C = 1 - e^{-S/T} \tag{3.1.3}$$

En donde x_i corresponde al cuadrado de la muestra de dato, con i variando desde 1 hasta el largo de la señal, T es el tiempo de decaimiento característico, o sea, el tiempo requerido para que la respuesta del impulso decaiga a $1/e$ de su valor original (Evans and Allen, 1983) mientras que S son los segundos por

muestra. Cabe destacar que usualmente (y particularmente en ObsPy) en lugar de la ecuación 3.1.3 se usa:

$$C = 1/N_{STA} \qquad o \qquad C = 1/N_{LTA} \tag{3.1.4}$$

Con N_{STA} y N_{LTA} siendo el largo (en muestras) de la ventana móvil de STA y LTA respectivamente. Finalmente, la función característica resultante de 3.1.1 y 3.1.2 (o como denominaremos ahora, CFT) sería:

$$CFT = \frac{STA}{LTA} \tag{3.1.5}$$

Ahora, con el STA/LTA ya definido, se deben ajustar ciertos parámetros para que el algoritmo funcione correctamente aparte de los ya mostrados como N_{STA} y N_{LTA}, y que además son utilizados dentro de la rutina de ObsPy (Trnkoczy, 2009). Estos son:

- Parámetros de filtrado: Con el objetivo de mejorar la calidad de la señal y quitar ruido sísmico que puede contaminar los datos, se fijan estos parámetros que corresponden a los límites de un filtro pasabanda (Frecuencia esquina inferior y superior), los cuales dependen directamente de lo que el usuario quiera destacar y/o atenuar de la señal.
- Valor umbral: También denominado *Trigger On* o *Threshold On*, es el valor que debe ser superado por la función característica CFT para declarar un posible evento sísmico. Esta declaración se mantiene hasta que la CFT disminuye a un valor definido.
- Valor *Trigger Off*: O también *Threshold Off*, es el valor umbral que representa un comportamiento normal de la CFT. Cuando se alcanza este punto luego de haberse declarado un posible evento, se demarca el termino de este.

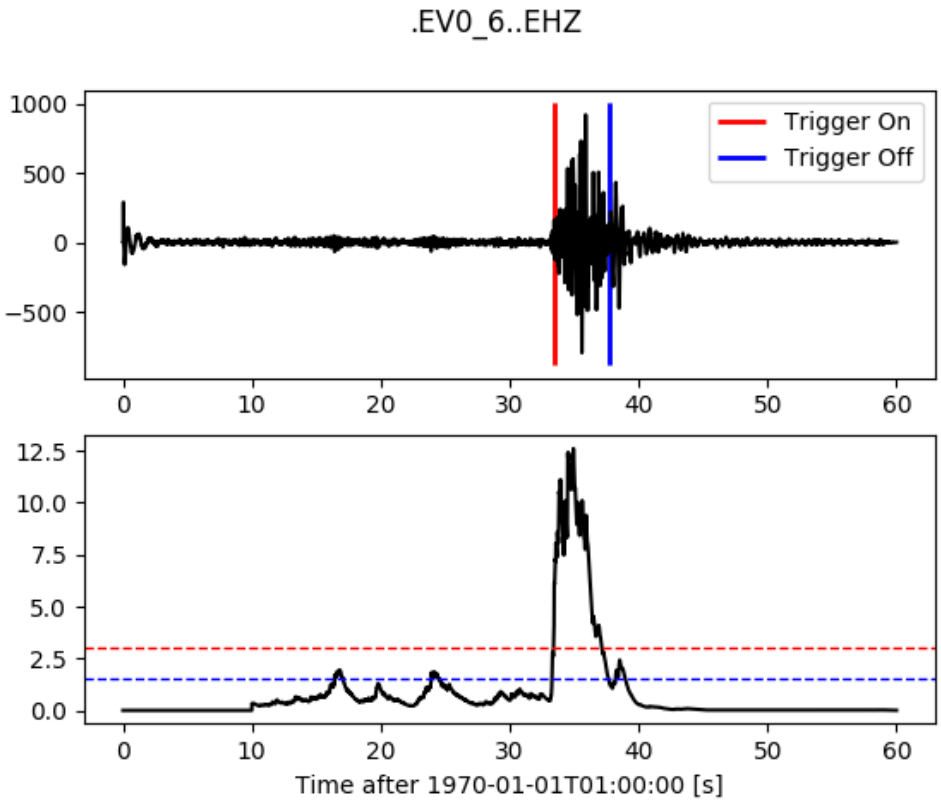


Figura 3.1.1: Ejemplo con traza obtenida de la base de ObsPy, con los parámetros personalizados. Sin filtrado, $STA = 0{,}5$, $LTA = 10$, $Trigger\_On = 3$ y $Trigger\_Off = 1{,}5$. Imagen superior presenta la traza en cuentas, con las respectivas marcas de *Trigger On* y *Trigger Off*. Imagen inferior muestra el STA/LTA recursivo calculado para la traza de la imagen superior.

Se puede observar en la figura 3.1.1 la personalización de los parámetros y el cálculo del STA/LTA a partir de la traza. Finalmente, el ajuste y optimización los parámetros dentro del algoritmo es fundamental para mantener un equilibrio entre una alta detección de eventos sísmicos y una baja cantidad de falsos positivos.

3.2. Machine-learning

Machine-learning corresponde al área de estudio o ciencia de programación computacional que busca que las computadoras *aprendan a través de los datos*, o sea, se le entrega al equipo implícitamente mediante algoritmos la habilidad de aprender patrones, tomar decisiones, clasificar, etc. sin ser programado u ordenado de forma directa, mejorando su desempeño a través del proceso de aprendizaje establecido por el usuario (Géron, 2017)

Una conocida definición de *Machine-learning* fue dada por (Mitchell et al., 1997): Se dice que un programa computacional *aprende* de una experiencia E, con respecto a un tipo de tarea T y medida de desempeño P, si su desempeño en la tarea T, medido por P, mejora con la experiencia E.

Dentro de los distintos tipos de modelos dentro de *Machine-learning*, se utilizará para este trabajo redes neuronales, los cuales se procede a definir a continuación.

3.2.1. Redes neuronales

Las redes neuronales artificiales (RNA de aquí en adelante) están inspiradas en sistemas biológicos donde un gran número de neuronas, que individualmente funcionan de forma lenta e imperfecta, colectivamente realizan funciones que grandes computadores no pueden igualar. Este campo ha sido y es uno de los más requeridos actualmente por la comunidad científica, esto debido a que es un campo interdisciplinario por naturaleza.

Tal como en un sistema nervioso biológico, el elemento fundamental de un RNA es la neurona artificial. La función de las neuronas artificiales son idénticas a las que realizan las neuronas reales: éstas deben integrar señales de entrada provenientes de otras neuronas y comunicar la señal integrada a un centro que tome las decisiones. Existen dos funciones principales para las RNA, la primera es la extracción de características o parámetros, mientras que la segunda es la asociación de patrones o generalización. La extracción de parámetros es estimada o aprendida por la RNA con una muestra representativa de patrones de entrada y salida (proceso de entrenamiento). La generalización del RNA es un patrón de salida en respuesta a un patrón de entrada, basado en la memoria del RNA que funciona como un cerebro humano (Wang and Teng, 1995).

Con objetivos similares a esta tesis existen algunos trabajos realizados (por ejemplo, (Ibs-von Seht, 2008)), sea para redes sísmicas regionales o redes de monitoreo sobre volcanes, siendo este último de especial interés. De hecho, existen ya casos de estudio en volcanes de Chile para la clasificación de señales, como es el del Volcán Llaima y Villarrica ((Curilem et al., 2014); (Curilem et al., 2009)

respectivamente), donde utilizan distintos métodos de parametrización de señales sísmicas, tipos de redes neuronales y bases de datos.

3.2.1.1. Multi-layered perceptron

Dentro de las redes neuronales existen distintos tipos de modelos, algunos especialmente desarrollados para ciertas tareas mientras que otros pueden ser bastante versátiles. Uno que es muy comúnmente usado corresponde al *Multi-layered perceptron* o Perceptrón multicapa (MLP de ahora en adelante) el cual funciona mediante tres capas fundamentales de neuronas interconectadas entre sí, con cada conexión incluyendo respectivos pesos. Una capa corresponde a las neuronas de entrada que transmiten la información, luego esta información es integrada a una capa oculta y posteriormente a una última capa de salida, con la característica capa oculta pudiendo ser compuesta por una o mas capas de neuronas, añadiendo así mayor complejidad a la red (Gardner and Dorling, 1998). Notar que cada neurona integra la información de las neuronas anteriores, ponderadas por los correspondientes pesos, para luego ser sujetas a una toma de decisión mediante las funciones de activación asociadas a cada capa.

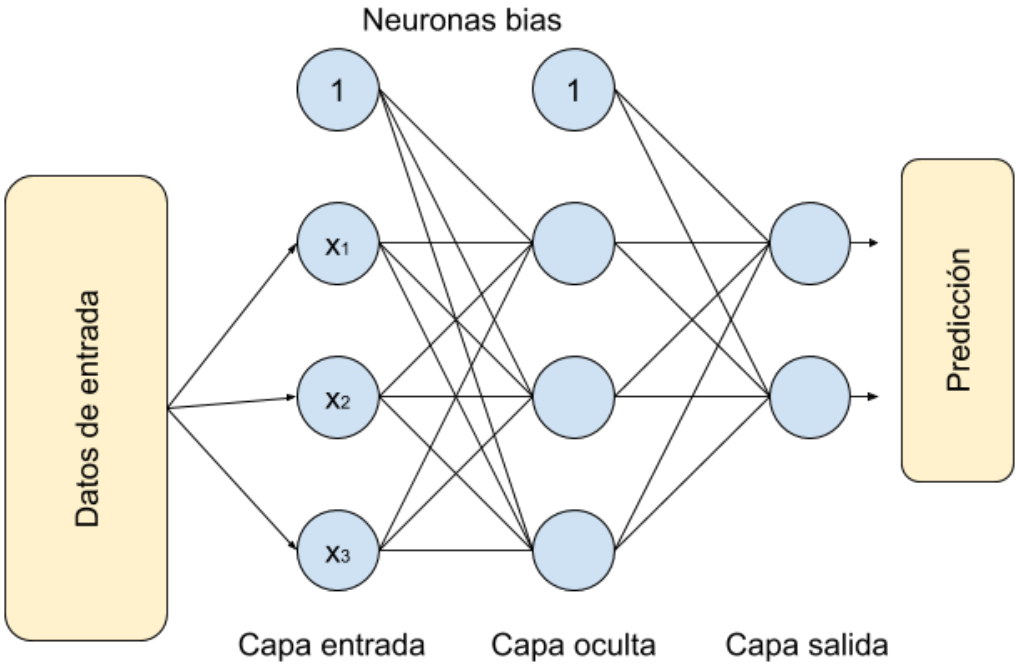


Figura 3.2.1: Ejemplo de la estructura de un Perceptrón multicapa, con 4 neuronas de entrada, una sola capa oculta con 4 neuronas y 2 neuronas de salida.

Se puede observar de la figura 3.2.1 la estructura de un MLP, con la característica capa oculta. Frente a este tipo de arquitectura de red, se desarrolló un método de entrenamiento en general, el cual es llamado *Backpropagation* (Ver A.1). Para este caso, la optimización del error de la red MLP se logrará a través del Gradiente descendiente, explicado a continuación.

3.2.2. Gradiente descendiente

El gradiente descendiente (*Gradient descent* en inglés, o GD) corresponde a un algoritmo de optimización que, en lugar de obtener los mejores parámetros o pesos para un modelo o red neuronal a través de una ecuación directa (como es el caso de la ecuación normal en regresión lineal), los obtiene a través de un proceso iterativo, paso a paso, en el cual el modelo converge a los valores óptimos de los parámetros que minimizan a la función de coste [1].

[1]Función de coste es una medida del error de la red sobre el set de entrenamiento, y el objetivo es minimizar este error para que la toma de decisiones de la red sea correcta.

Para lograr esto, el algoritmo inicia con un set de parámetros aleatorios, luego calcula el gradiente local de la función de coste y se mueve en la dirección del mayor gradiente negativo con paso definido por la tasa de aprendizaje η, modificando en cada iteración los parámetros iniciales, hasta alcanzar un gradiente cero, que correspondería al mínimo del error (local o absoluto), y es en este punto que se dice que los parámetros son óptimos.

Sin embargo, el uso de este algoritmo presenta ciertas problemáticas, como por ejemplo, la definición de una tasa de aprendizaje no adecuada (Ver figura 3.2.2). Si la tasa es muy baja, el algoritmo va a converger de forma segura al mínimo error pero de forma más lenta, por lo que será necesario realizar mayor número de iteraciones lo que significa un mayor costo computacional, inclusive con la posibilidad de no alcanzar los parámetros óptimos por falta de iteraciones. Por otro lado, si la tasa es muy alta, convergerá rápidamente a una solución pero puede saltarse el mínimo error, ya que los pasos que dé el algoritmo durante las iteraciones pueden ser muy largos. También se pueden encontrar otros problemas como converger a mínimos locales en lugar del mínimo absoluto, o encontrarse con una zona de gradiente cero, que no necesariamente sea un mínimo sino un plano por ejemplo. Para enfrentar esas problemáticas resulta muy importante definir una buena tasa de aprendizaje, además de tomar en consideración la forma del gradiente de la función de costo, por ejemplo, en el caso de MSE (*Mean Squared Error* o Error Medio Cuadrado), su gradiente tiende a no tener cambios abruptos y suele ser un valle, por lo tanto se puede afirmar con seguridad que el algoritmo llegará a un buen resultado (Géron, 2017).

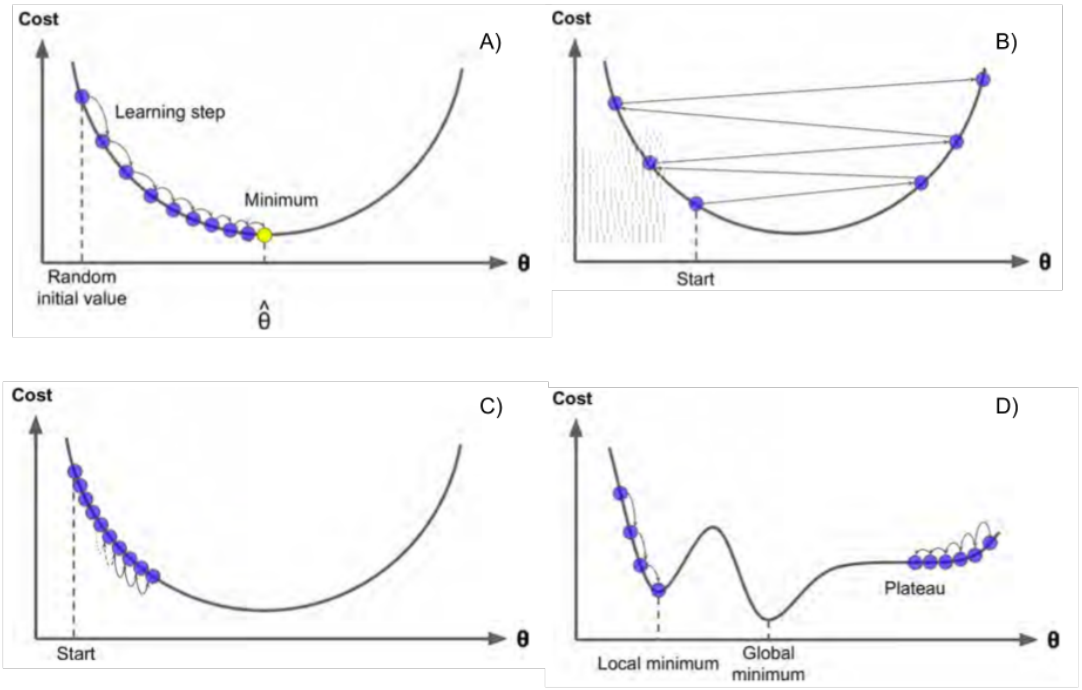


Figura 3.2.2: Funcionamiento y problemáticas del gradiente descendente. A) Ejemplo simple del proceso de optimización del vector de parámetros θ. B) Ejemplo de una tasa de aprendizaje η muy grande. C) Ejemplo de una tasa de aprendizaje η muy baja. D) Ejemplo de una función de coste irregular y encuentro de mínimos tempranos (local o plano). Imágenes compiladas de (Géron, 2017).

Para definir matemáticamente el proceso de optimización, tomaremos el caso de la función de coste MSE, que se define de la siguiente forma:

$$MSE(\theta) = \frac{1}{m}\sum_{i=1}^{m}(\hat{y}^{(i)} - y^{(i)})^2 \tag{3.2.1}$$

Con $\hat{y}^{(i)} = \theta^T \cdot \mathbf{x}^{(i)}$ el valor predicho por el modelo para la instancia de entrenamiento i, θ^T la transpuesta del vector de parámetros, $x^{(i)}$ el vector de datos de entrenamiento para la instancia i, m el número de instancias de entrenamiento y $y^{(i)}$ el valor objetivo de la instancia i. En este punto, es donde implementamos el gradiente descendiente, calculando la derivada parcial de la ecuación 3.2.1 de MSE (nuestra función de costo en este caso) con respecto a cada parámetro del vector de parámetros que queremos optimizar, es así que obtenemos la siguiente ecuación:

$$\frac{\partial}{\partial\theta_j}MSE(\theta) = \frac{2}{m}\sum_{i=1}^{m}(\theta^T \cdot \mathbf{x}^{(i)} - y^{(i)})x_j^{(i)} \tag{3.2.2}$$

Donde $\frac{\partial}{\partial\theta_j}MSE(\theta)$ es la derivada parcial con respecto al j-ésimo parámetro θ. Estos valores serán contenidos en el vector de gradiente $\nabla_\theta MSE(\theta)$, el cual puede ser calculado directamente aplicando gradiente:

$$\nabla_\theta MSE(\theta) = \begin{pmatrix} \frac{\partial}{\partial\theta_0}MSE(\theta) \\ \frac{\partial}{\partial\theta_1}MSE(\theta) \\ \vdots \\ \frac{\partial}{\partial\theta_n}MSE(\theta) \end{pmatrix} = \frac{2}{m}\mathbf{X}^T \cdot (\mathbf{X} \cdot \theta - \mathbf{y}) \tag{3.2.3}$$

Luego, el vector gradiente calculado es aplicado para ajustar los parámetros del modelo. esto se hace para cada iteración de la siguiente forma:

$$\theta^{(\text{siguiente})} = \theta - \eta\nabla_\theta MSE(\theta) \tag{3.2.4}$$

Finalmente se puede observar que para este caso, el gradiente descendiente usa todo el set de entrenamiento para cada instancia en la ecuación 3.2.3 A este tipo de gradiente descendiente se le denomina *Batch Gradient Descent*, debido a que utiliza todos los datos o el *Batch* completo en cada iteración. Lo anterior puede ser un problema debido a que puede volver lento el proceso de optimización, por esto mismo es que existen otros tipos de gradiente descendiente que se repasarán rápidamente a continuación:

- *Stochastic Gradient Descent* propone un funcionamiento opuesto al *Batch Gradient Descent* recién expuesto, ya que en lugar de tomar todo el set de datos, usa una única instancia de entrenamiento al azar para cada iteración, por lo que es mucho más rápido el proceso de optimización y calculo del gradiente, pero sufre de una función de costo irregular, con el error en cada iteración disminuyendo y aumentando de forma intermitente y únicamente decreciendo en promedio. A su vez, se le hace más difícil al algoritmo encontrar los valores más óptimos de los parámetros, pero si encuentra buenos valores rápidamente. Por último, el hecho que avance de forma irregular la optimización ayuda a evitar estancarse en mínimos locales, y encontrar más fácilmente el mínimo absoluto (aunque le sea difícil estabilizarse en un sólo valor).

- *Mini-Batch Gradient Descent* corresponde a un algoritmo que se puede

entender como el punto medio entre el *Stochastic GD* y el *Batch GD* anteriormente explicados, ya que los gradientes son calculados sobre pequeños sets de datos o instancias llamados *Mini-batch*, los cuales son elegidos aleatoriamente y subdivididos del set principal o *Batch*. Lo anterior permite al algoritmo de optimización ser relativamente más rápido que el *Batch GD* y menos errático que el *Stochastic GD*. Notar también que lo anterior depende del tamaño de los *Mini-batch*, por lo que se puede hacer aún menos errático al proceso utilizando *Mini-batchs* de mayor tamaño.

3.2.3. Funciones de activación

Las funciones de activación tiene la tarea de realizar la toma de decisiones de la neurona, dependiendo de la información integrada a través de las conexiones y pesos con la capa de neuronas anterior. Dependiendo de lo anterior es que se transmitirá una respuesta numérica a la siguiente capa de neuronas, representando una decisión (Nwankpa et al., 2018).

En lugar de utilizar funciones de tipo Signo o Escalón, en este caso se usan funciones logísticas o Sigmoides, debido a que estás poseen gradiente, lo cual sirve durante el proceso de entrenamiento, además que son no-lineales. Usualmente se utilizan funciones de activación del tipo Sigmoide, que es definida de la siguiente forma:

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{3.2.5}$$

Esta función tiene forma de S, es continua y diferenciable, y sus valores varían entre 0 y 1, lo cual genera salidas normalizadas, debido a esto es que se sugiere su uso para patrones con un comportamiento similar.

Por otro lado, existen otras dos funciones de activación que son ampliamente usadas y que puede ser útil comparar, como se podrá ver también en la figura 3.2.3:

- Función Tangente Hiperbólica (*Tanh*): Bastante similar en características a la función Sigmoide, con la diferencia que sus valores varían entre -1 y 1, por lo que es centrada en 0. Se define como:

$$tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{3.2.6}$$

- Función ReLU: Su nombre proviene de *Rectified Linear Unit* y es básicamente una función plana desde $-$ínf hasta 0, para luego pasar a una función recta en función de z hasta ínf. Es continua pero no diferenciable en 0, tiene la ventaja de ser más rápida de calcular, además de que inhibe y activa neuronas dependiendo del valor que resulte de ella (inhibe si $z < 0$, activa si $z > 0$). Por último, esta función no es normalizada, y se encuentra definida por la siguiente ecuación:

$$ReLU(z) = \begin{cases} z & , z \geq 0 \\ 0 & , z < 0 \end{cases} \tag{3.2.7}$$

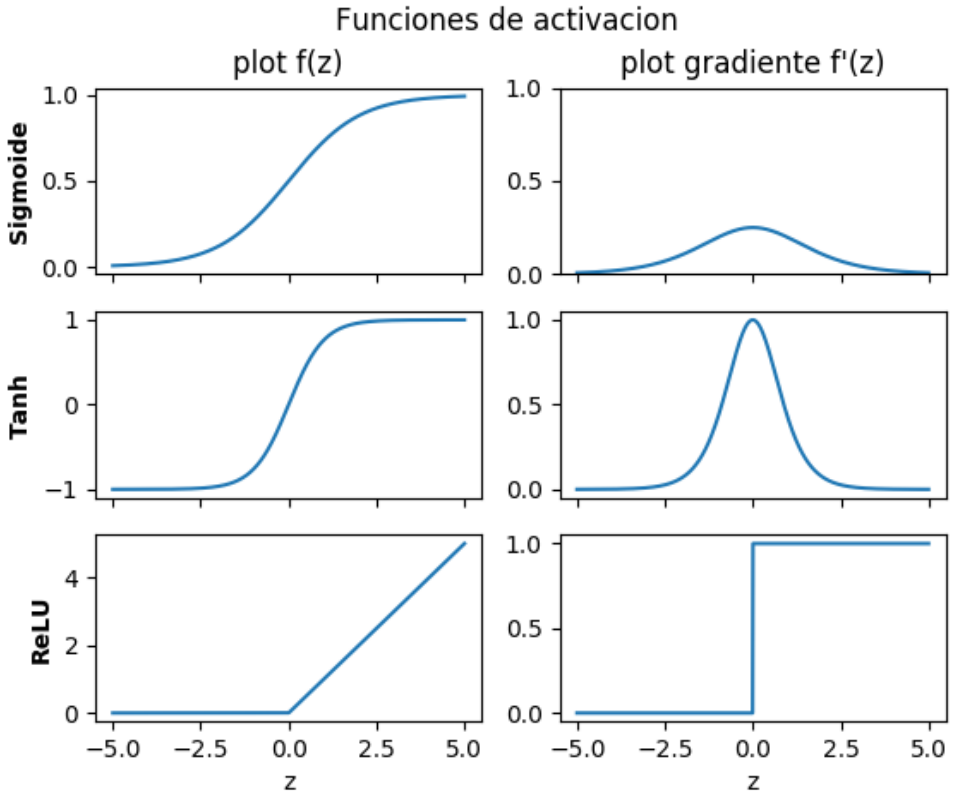


Figura 3.2.3: Comparativa de las funciones de activación a ser utilizadas en este estudio. Columna izquierda: Gráficos de las funciones de activación Sigmoide, Tangente Hiperbólica y ReLU (de arriba hacia abajo). Columna Derecha: Gráficos de los gradientes de las funciones de activación ya mencionadas, en el mismo orden anterior.

3.2.4. Regresión Softmax

Ahora que está definida nuestra red neuronal y se sabe a grandes rasgos como funciona internamente su arquitectura, se puede ver que es lo que ocurre en las capas de salida y como estas entregan una predicción del resultado.

Generalmente en el caso de la clasificación binaria, se toma el resultado de la neurona de salida y se interpreta su resultado numérico (según como se le entrenó y como lo indiquen las funciones de activación) tomando así una decisión y asignándole una clase, obteniendo una predicción. Sin embargo, se puede tomar una aproximación probabilística al problema, mediante *Logistic Regression* o Regresión logística, la cual básicamente estima la probabilidad (entre 0 y 1) de que cierta instancia pertenezca a una clase mediante la sumatoria ponderada de los patrones de entrada (trabaja como una función de activación). Esta probabilidad es calculada a través de una función Sigmoide tal como se definió en la ecuación 3.2.5. Finalmente, en base a esta probabilidad se predice un resultado, de la forma:

$$\hat{y} = \begin{cases} 0 & ,\hat{p} < 0{,}5 \\ 1 & ,\hat{p} \geq 0{,}5 \end{cases} \tag{3.2.8}$$

En donde $\hat{y}$ corresponde al valor objetivo predicho y $\hat{p}$ la probabilidad de clase estimada para la instancia usando la ecuación 3.2.5. Como se puede ver, desde 0.5 de probabilidad se predice la pertenencia a la clase 1, y en caso contrario la clase 0. Ahora, esto es diferente cuando no se trata de una clasificación binaria sino de una multiclase, ya que una sola función Sigmoide deja de ser suficiente. Es aquí que es necesaria la introducción del concepto de *Softmax regression* o Regresión Softmax.

La Regresión Softmax (su nombre proviene de la función *argmax*, pero suavizada) se puede entender como la generalización a múltiples clases de la Regresión logística y funciona de la siguiente manera: Considerando sólo una instancia de ejemplo, el modelo calcula una puntuación para dicha instancia con respecto a cada clase (en base a lo que entrega la red), luego se obtiene la probabilidad de cada clase similarmente como se hace en Regresión logística pero esta vez aplicando la función Softmax a las puntuaciones ya mencionadas, por lo que la clase predicha será aquella de mayor probabilidad (por consecuencia, de

mayor puntuación) ((Bishop, 2006), (Géron, 2017)). La función Softmax se define como:

$$\hat{p}_k = \sigma(\mathrm{s}(\mathbf{x}))_k = \frac{e^{(s_k(\mathbf{x}))}}{\sum_{j=1}^{K} e^{(s_j(\mathbf{x}))}} \tag{3.2.9}$$

Con $\hat{p}_k$ la probabilidad de pertenencia a la clase k y $\mathrm{s}(\mathbf{x})$ el vector de puntuaciones para cada clase para los datos $\mathbf{x}$. Para estimar la probabilidad de pertenecer a la clase k calcula la exponencial de la puntuación de la clase k dividida por la sumatoria de todas exponenciales de las puntuaciones de cada clase. Es necesario notar que la función *Softmax* se reduce y equivale a una Regresión logística en el caso de una clasificación binaria.

3.2.5. Cross-entropy

Como ya hemos visto en otras secciones, típicamente es utilizado el MSE o RMSE para medir el error de la toma de decisiones de la red neuronal, o sea, se utilizan como función de costo. Para este caso, se utilizará *Cross-entropy* o la Entropía cruzada, cuyo concepto se entiende como la medida de diferencia entre dos distribuciones de probabilidad dada una variable aleatoria. En teoría informática, la Entropía cruzada corresponde al número promedio de bits para enviar información desde una distribución p, pero usando un modelo q (que es una aproximación de p), o sea, si q es un modelo muy similar a la distribución p, en promedio se usarán pocos o ningún bit extra, por lo tanto la Entropía cruzada será baja, pero en el caso contrario en donde q es una mala aproximación de p, para enviar la misma información se usarán en promedio una mayor cantidad de bits (debido a que existe una mayor incertidumbre en la información a enviar), por lo que la Entropía cruzada será mayor. Matemáticamente la Entropía cruzada entre las distribuciones p y q se define de la siguiente manera:

$$H(p, q) = -\sum_{x} p(x)\log(q(x)) \tag{3.2.10}$$

Esto considerando el caso de distribuciones de probabilidad discreta. De lo mencionado anteriormente, se puede hacer una idea de su utilidad como función de coste o error en optimización, pues si utilizamos como distribución p la probabilidad esperada u objetivo de cada clase, y q como la probabilidad predicha para cada clase (aquí entra en función lo explicado en la sección 3.2.4), se dará el caso en

que si estas distribuciones son muy diferentes entre sí, la Entropía cruzada lo penalizará y por lo tanto será alta. Así, el objetivo es minimizar esta función ((Bishop, 2006); (Géron, 2017)). La Entropía cruzada como función de coste se define:

$$J(\Theta) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log(\hat{p}_k^{(i)}) \tag{3.2.11}$$

En donde Θ corresponde a la matriz de parámetros, $y_k^{(i)}$ es la probabilidad de la clase objetivo para la instancia i, y $\hat{p}_k^{(i)}$ es la probabilidad predicha por la red para la clase k (proviene de la ecuación 3.2.9). Notar que en este caso $y_k^{(i)}$ es 1 si para la instancia i la clase objetivo es k, en caso contrario es 0, debido a que sabemos de antemano cual es la clase a la cual corresponde cada instancia y por lo tanto dicha instancia tiene probabilidad total de pertenecer a aquella clase.

Finalmente, se utiliza el procedimiento ya explicado en la sección 3.2.2, que corresponde a calcular el gradiente de 3.2.11 y aplicar Gradiente descendiente para encontrar la matriz Θ que minimiza el error de la ecuación 3.2.11.

3.2.6. Matriz de confusión

La matriz de confusión es una herramienta que permite evaluar y visualizar de buena manera el desempeño de un clasificador, en las columnas se muestran las predicciones de cada clase, mientras que las filas las clases observadas (reales). Notar que para utilizar la matriz de confusión ya se debe tener un set de resultados o predicciones del modelo, al igual que los datos originales con los que se va a comparar, usualmente se utilizan sets de entrenamiento o validación y sus correspondientes predicciones. Un ejemplo de como se ve una matriz de confusión puede ser el siguiente:

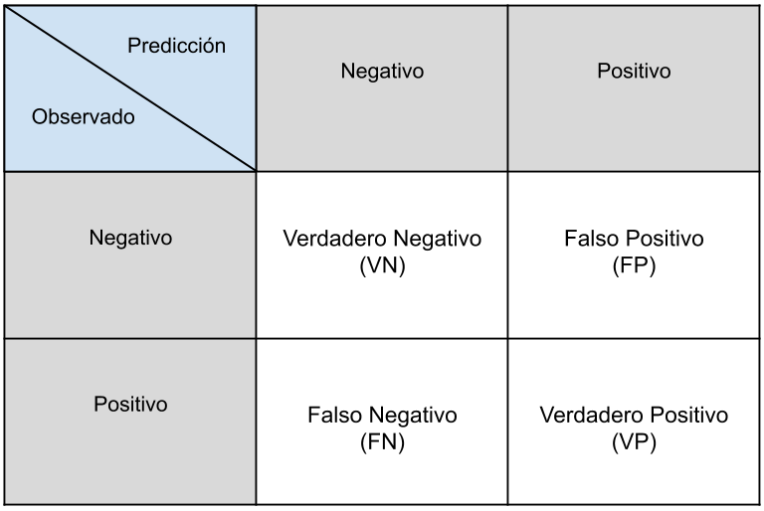


Figura 3.2.4: Matriz de confusión para una clasificación binaria, en ella se muestra la ubicación de los Verdaderos negativos (VN), Falsos positivos (FP), Falsos negativos (FN) y Verdaderos positivos (VP). Estos elementos son fundamentales para el calculo de distintas métricas.

Otro de los beneficios de utilizar la matriz de confusión, es que de ella se derivan distintas métricas de desempeño que analizan aspectos de la calidad del clasificador, además que los hace visualmente comprobables (Géron, 2017). Entre ellos tenemos:

- Exactitud: Corresponde a la razón entre predicciones correctas y el total de predicciones realizadas, se calcula de la siguiente manera:

$$\text{exactitud} = \frac{VP + VN}{VP + VN + FP + FN} \tag{3.2.12}$$

- Precisión: Es la razón entre predicciones correctas y el total de predicciones positivas realizadas por el clasificador. Una alta precisión nos dice que el clasificador difícilmente confundirá otra clase con la correcta. Esta medida de calcula como:

$$\text{precisión} = \frac{VP}{VP + FP} \tag{3.2.13}$$

- Sensibilidad: Es la razón entre predicciones correctas y el total de observaciones positivas. Una alta sensibilidad se traduce a que el clasificador identifica correctamente las observaciones positivas y no las confunde con

otra clase. Se define de la siguiente forma:

$$\text{sensibilidad} = \frac{VP}{VP + FN} \tag{3.2.14}$$

4. Metodología

4.1. Desarrollo de base de datos

Se contó con acceso a las series de tiempo de distintas estaciones instaladas en las zonas cercanas al Volcán Copahue, entregadas por Centro de Monitoreo Volcánico de la Universidad de Concepción. De estas estaciones, cuyas ubicaciones se encuentran especificadas en la Figura 4.1.1 y Tabla 4.1.1, se eligió aquella que presentara una mayor calidad en sus registros, comparando distintas señales, de lo cual se concluyó que la mejor estación para los fines de este estudio es la denominada COPA (ver figura 4.1.2).

N°	Estación	Latitud [°]	Longitud [°]
1	CALL	-37.9033	-71.5308
2	COPA	-37.9550	-71.2435
3	MAYA	-37.9924	-71.4590
4	COP2	-38.0327	-71.3058
5	TOL1	-38.3724	-71.6781
6	LONQ	-38.3890	-71.5653
7	MANZ	-38.4620	-71.6833

Tabla 4.1.1: Identificación y ubicación de las estaciones de la figura 4.1.1.

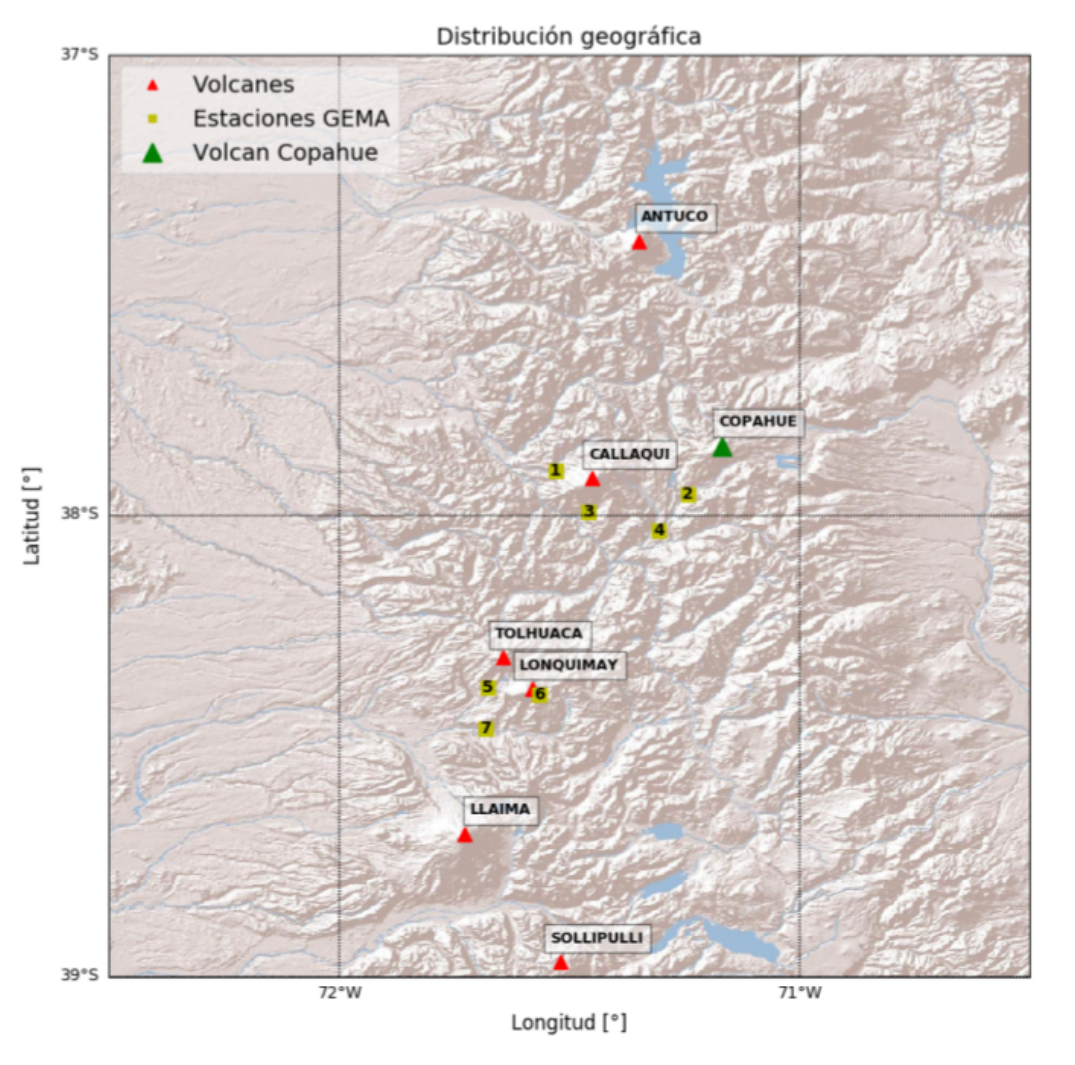


Figura 4.1.1: Localización de las distintas estaciones de las que se tuvo información, los números hacen referencia a la tabla 4.1.1 para su identificación. Notar que COPA es la estación más cercana al Vn. Copahue, marcado en color verde. El resto de volcanes se encuentran marcados en triángulos rojos.

Se tomó en consideración datos desde enero del año 2018 hasta mediados del 2020, cercano al mes de Junio, los cuales se encuentran en archivos diarios en formato SEED, con una frecuencia de muestreo de 25 Hz. Estos registros, junto con el catálogo de eventos detectados por la red del Centro de Monitoreo Volcánico, fueron los cimientos de la base de datos.

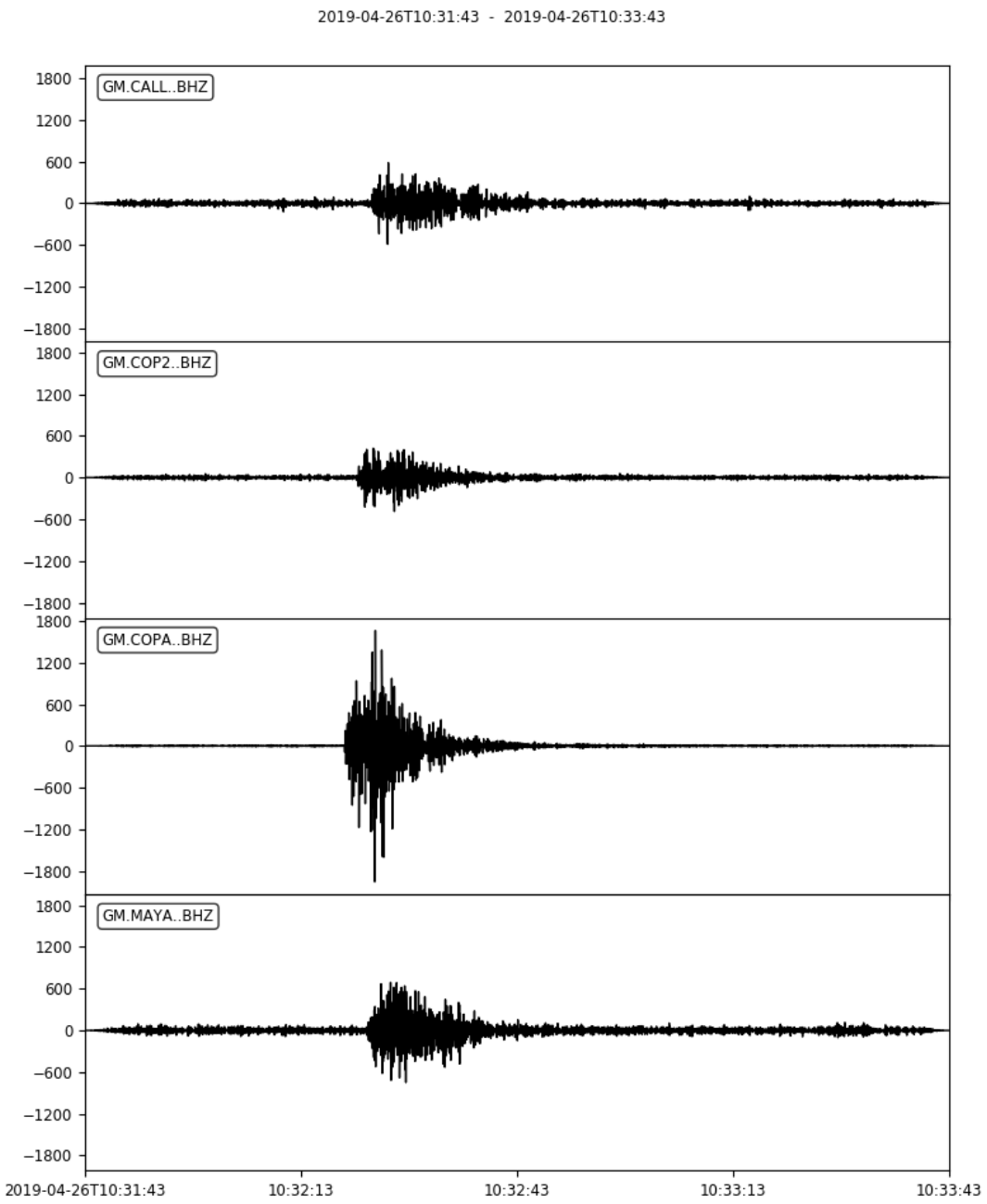


Figura 4.1.2: Comparación de la señal recibida de un evento localizado en la caldera del Vn. Copahue, con fecha 2019-04-26 10:32:13 en las estaciones CALL, COP2, COPA y MAYA, respectivamente. Notar que la señal es recibida de mejor manera y mayor amplitud en COPA.

En términos generales, el siguiente paso a seguir fue transformar la base de datos a un patrón que pueda ser aprendido por el MLP (Wang and Teng, 1995) calculando el STA/LTA recursivo (3.1.2) de la señal y cortando los primeros 2 segundos del evento (o sea, la primera llegada asociada

a la onda P), correspondiente a 50 puntos según la frecuencia de muestreo ya establecida, de modo que estos datos sean introducidos de forma directa a la red.

Cabe notar que, para el correcto funcionamiento de la red neuronal clasificadora, es necesario que esta aprenda características tanto de eventos como de ruido sísmico, por lo que a continuación se ahondará en el desarrollo de ambas clases de datos, detallando su pre-procesamiento, extracción y transformación a patrones reconocibles por la red.

4.1.1. Eventos sísmicos

En primer lugar, se desarrolló un código en Python llamado *gen_ database.py*, a través de ObsPy, que permitió cargar y cortar las trazas correspondientes a los eventos catalogados en secciones de 2 minutos (desde 40 segundos antes, y 80 segundos después del tiempo de inicio estipulado en el catálogo) de forma automática, a partir de la base de datos. Estas trazas recortadas fueron separadas en una base manejable (en formato SEED, con sismogramas incluidos para su revisión) y ordenadas por fecha de registro, obteniendo un total de 182 sismos registrados, a la espera del pre-procesamiento.

Este llamado pre-procesamiento constó principalmente de la limpieza de las trazas, quitando la media y tendencia de los datos, además de aplicar *tapering* o ventana de Hann [1] para luego filtrar entre 3.5 y 10 Hz, con tal de incrementar la calidad de la señal. Finalmente, se quitó las secciones de la traza con *taper* o suavizadas, para poder calcular el STA/LTA recursivo asociado a la señal, mediante la ecuación 3.1.5. Para esto, se utilizó un STA y LTA de 0.5 y 10 segundos, además de umbrales THR_ON y THR_OFF de 3 y 1.5 respectivamente.

Al contar con la serie de STA/LTA, se procedió a extraer una sección de un total de 2 segundos de largo (50 puntos de datos), específicamente $\pm$ 1 segundo alrededor del *Trigger On* del correspondiente evento, entregado por la rutina de ObsPy de *Triggering*. Todo lo anterior, incluyendo el pre-procesamiento, se desarrolló a través de un código de Python denominado *gen_ input_ COPA.py*. Luego, aún dentro del mismo código, se procedió a guardar los patrones obtenidos

[1] Ventana de Hann: Se aplica para suavizar los extremos mediante un coseno ponderado. Este proceso se realiza para evitar errores debido al truncamiento.

como vectores en archivos .txt, en conjunto al gráfico de los 2 segundos de datos, para una posterior revisión.

Finalmente, los patrones de cada uno de los 182 eventos fueron revisados manualmente, con el objetivo de eliminar aquellos que presentaran errores, almacenando así 166 muestras (cada una correspondiente a un evento) las cuales fueron posteriormente introducidas directamente a la red como sets de datos para su entrenamiento.

4.1.2. Ruido sísmico

Con respecto a la generación de patrones para ruido sísmico, se creó un código de Python de manera similar al caso de los eventos sísmicos, denominado *gen_input_noise_COPA.py*. El proceso de extracción en este caso fue un poco más complicado, debido a que ya no se tiene un catálogo o *Triggers* que sirva de guía para extraer secciones, por lo tanto, se debió utilizar aleatoriedad en el proceso. Primero, se usó nuevamente la base de datos diaria y se tomó un día de forma aleatoria, el cual fue subdividido en trazas de 5 minutos, evitando aquellas que contuviesen un evento ya listado en el catálogo. Posteriormente, se realizó el pre-procesamiento de igual forma a lo realizado con los eventos sísmicos, obteniendo el STA/LTA de cada traza al final utilizando los mismos parámetros del caso de eventos.

Luego, dentro del script ya mencionado, se analizó cada traza de STA/LTA de forma que se cumplieran ciertas condiciones, como una desviación estándar >1 para evitar los artefactos producidos por trazas interpoladas, cero *peaks* para ruido uniforme, y un máximo de 4 *peaks* para ruido muy variante, de forma de abarcar variados comportamientos del ruido de fondo, y así evitar que la red confunda ruido muy variante en amplitud con eventos, minimizando falsos positivos. Aquellas trazas que cumplieran con las condiciones de ruido variante se le extrajo 2 segundos sobre un *peak* aleatorio, mientras que para el ruido más uniforme, se tomó un punto medio de las secciones de 5 minutos definidas anteriormente, en donde se extrajo los 2 segundos necesarios para el patrón.

Finalmente, se obtuvo un número indefinido de muestras a partir del ruido,

por lo que la única consideración que se tuvo es que el número de patrones para el ruido debiese equiparar al de muestras de eventos durante el entrenamiento (166 hasta el momento), por lo que las muestras sobrantes fueron desechadas, posterior a la revisión pertinente para eliminar aquellas que mostraran errores o no tuviesen un comportamiento normal, similar a lo realizado en el caso de eventos sísmicos, también de forma manual.

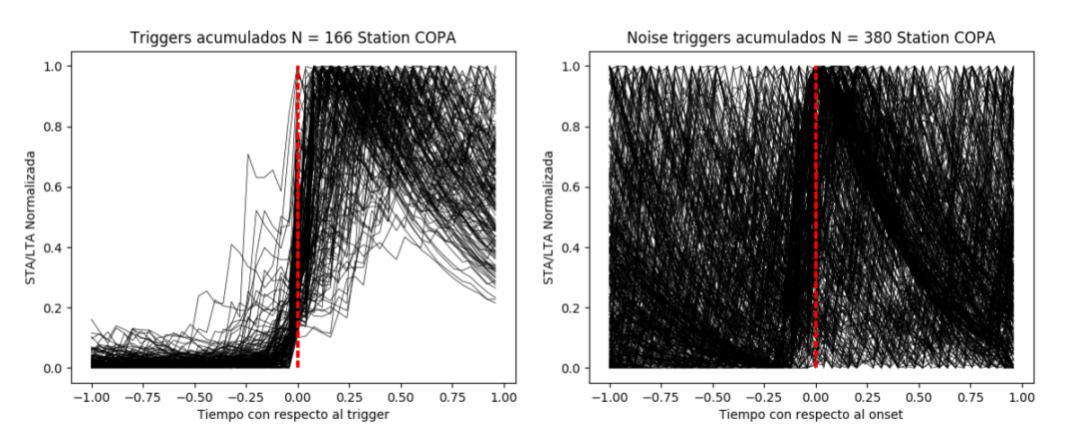


Figura 4.1.3: Muestras obtenidas, normalizadas individualmente, con el objetivo de mostrar el patrón o forma característica de la clase. Izquierda: Patrón de los eventos sísmicos, con un marcado escalón en el *Trigger* (linea punteada roja). Derecha: Patrón del ruido, con observable aleatoriedad. Parámetros usados según estipulado en 4.1.2 o 4.1.1.

Es posible observar de la figura 4.1.3 los patrones que se espera la red pueda identificar, para así clasificar correctamente las clases de eventos y ruido, sumando al proceso las diferencias de amplitud, que obviamente hay entre señales correspondientes a sismos y señales de ruido de fondo. Destacar, que en la figura 4.1.3 cada muestra está normalizada de forma independiente de el resto del set, sólo con fines de visualización del patrón acumulado, posteriormente se tomará su valor de amplitud individual en consideración al ingresar a la red.

4.2. Construcción red neuronal

Finalizada la generación de la base de datos inicial junto con las muestras que representaran a cada clase, se procedió a la construcción de la red a la que serán introducidos estos datos. En la presente sección, se detallará este proceso, además de la estructura utilizada (o sea, las características básicas del MLP) y el entrenamiento de la red.

4.2.1. Arquitectura de MLP

Como ya se mencionó en la sección 3.2.1.1, la red utilizada es una de tipo Perceptrón Multicapa o MLP. La arquitectura de la red para esta implementación consta de una capa de entrada de 50 neuronas, ya que las muestras constan de 50 puntos de datos. Debido a que el caso de estudio es un reconocimiento de patrones simple, se consideró usar una sola capa oculta, con un número de neuronas ocultas variable de 1 a 20 neuronas. El número final de estas fue determinado mediante comparación de desempeño y error, lo que será mostrado más adelante, mientras que la capa de salida fue de 2 neuronas, con cada una representando una clase. Las interconexiones de las neuronas, refiriéndose a los pesos, fue definida como aleatoria cada vez que se ejecutaba un nuevo modelo, lo cual fue añadido junto con todas las demás características y configuraciones mencionadas, dentro de un código de Python, gracias a la librería Tensorfow, denominado *training_ mlp.py*, con el objetivo del desarrollo de modelos de MLP.

Por otro lado, en cuanto a las funciones de activación, tal como fue mostrado en la sección 3.2.3, se varió entre las 3 funciones presentadas y se eligió aquella que se comportaba de mejor forma y con menor error. Un resumen de la arquitectura de los modelos de la red MLP se muestra a continuación en la figura 4.2.1.

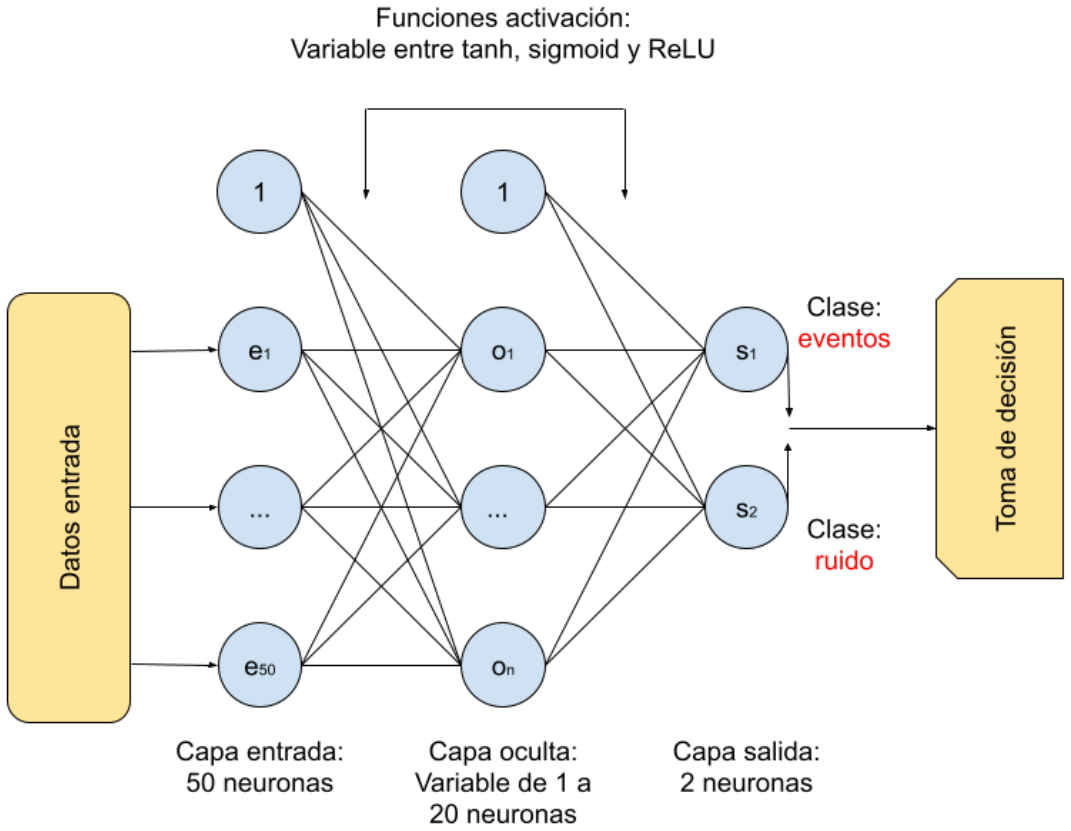


Figura 4.2.1: Esquema de la red MLP, y las variaciones ingresadas. Basado de la figura 3.2.1.

Finalmente, para la estructura de salida, se utilizó lo especificado en las secciones 3.2.4 y 3.2.5 (*Softmax* y *Cross entropy*) como función de activación de salida y función de coste de la red respectivamente. Lo anterior, también fue incluido en el código *training_ mlp.py*, para el desarrollo de los distintos modelos.

4.2.2. Entrenamiento de MLP

Para entrenar a cada modelo de la red, se utilizó la misma base de datos, ya especificada en 4.1, en donde de las 166 muestras de eventos se separó 150 para utilizar en el entrenamiento, mientras que los 16 restantes fueron asignados a una traza continua para testeos posteriores. A este set de datos, se le sumaron 150 muestras aleatorias de ruido para representar equitativamente ambas clases, formando así la base de entrenamiento inicial con un total de 300 patrones.

Por otro lado, dentro del código para el entrenamiento *training_ mlp.py*, se le

asignó a cada una de las 300 muestras su valor objetivo, que corresponde a su clase asociada, denominada y (las muestras corresponden a x). En el caso de eventos sísmicos el valor objetivo es 1, mientras que para la clase de ruido el valor es 0. Luego, al set de entrenamiento y de valores objetivos respectivos se les realizó una permutación o mezcla homogénea, para después subdividirlo en sets de validación, test y entrenamiento, que son definidos a continuación:

- **Set validación**: Corresponde al set de datos que es utilizado para calcular medidas de desempeño y error, durante y después del proceso de entrenamiento, con la particularidad de que el modelo no ha visto en ningún momento estos datos, por lo que sirve para analizar que tan bien el modelo generaliza frente a nuevos datos. Se utilizó el 20 % del set inicial para constituir este set, quedando en un total de 60 muestras.

- **Set para test**: Es el conjunto de datos cuya finalidad, en este caso, es construir matrices de confusión y predicciones de prueba, y generar otras medidas como la exactitud, fuera del proceso de entrenamiento. Al igual que el set de validación, estos datos en ningún momento han sido analizados por la red. Nuevamente se hizo uso del 20 % de datos, esta vez del set de entrenamiento restante del set de validación, por lo que se contó con 48 muestras.

- **Set entrenamiento**: Tal como lo explica el nombre, es el set cuya única función es ser ingresado al modelo para entrenarlo. Esto es importante debido a que el utilizar este set para otras funciones puede entregar resultados sesgados, como medir el desempeño o error después del entrenamiento. Lo anterior ocurre debido a que la red ya conoce estos datos y sus clases, y ya se encuentra ajustado con respecto a ellos. Para este estudio, son 192 muestras las que constituyen al set de entrenamiento.

Para entrenar a los distintos modelos se aplicó el algoritmo de *Mini-batch gradient descent* dentro del código *training_mlp.py*, mostrado en la sección 3.2.2 (excepto en los casos experimentales mencionados), con una tasa de aprendizaje de 0.005 y 500 iteraciones. En cuanto a la normalización o estandarización de los datos, se abordó ambos casos, pero se tomó la decisión de utilizar la estandarización, debido a que es más fácil de aplicar a nuevos datos.

Por otro lado, los datos fueron ingresados con y sin estandarización, esto debido a la variedad de funciones de activación (normalizadas y no-normalizadas) en conjunto a la naturaleza de los datos que conforman cada muestra, los cuales tienen valores bajos y no presentan gran variación (un orden de magnitud como máximo). Un ejemplo del proceso que atraviesan los datos se puede observar de la figura 4.2.2.

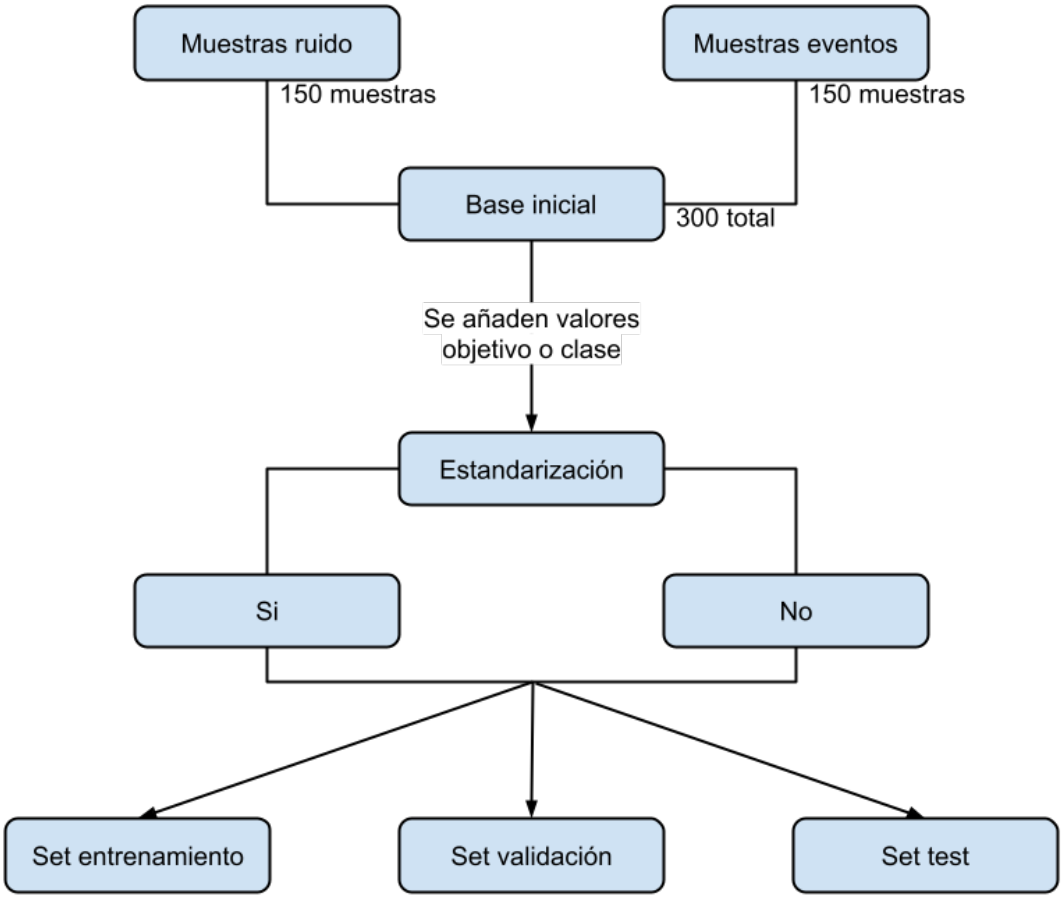


Figura 4.2.2: Esquema del procedimiento de los datos y separación en sets.

4.3. Comparación de modelos

Con el objetivo de evaluar los distintos comportamientos de las funciones de activación, la influencia de la cantidad de neuronas ocultas y los resultados de estandarizar los datos, se realizó una comparación de los modelos mediante distintas métricas y métodos, los cuales se especificarán al final de esta sección en conjunto al proceso de selección. Por ahora, se describirá la generación de los

modelos que luego se sometieron a comparación.

Para generar los modelos de forma consecutiva se creó una rutina de Python, llamada *best_ hidden.py*. Lo primero que se debe destacar, es que se mantuvo la base de datos y su distribución de sets para todas las versiones generadas, al igual que la estructura básica y parámetros, como la tasa de aprendizaje y número de iteraciones, similarmente a lo hecho con el código de *training_ mlp.py*. En cuanto al proceso de generación de los modelos, primero para cada función de activación se hizo variar la cantidad de neuronas ocultas de 1 a 20, y se creó 3 veces cada versión (o sea, se generaron 3 modelos de 1 neurona, 3 de 2 neuronas, etc) con el objetivo de asegurarse de que algún modelo no cayera en un mínimo local, manteniendo de aquellas 3 versiones la que presentara menor error. Luego, se repitió el mismo proceso anterior para cada función de activación, pero sin estandarización de los datos.

Al momento de generar cada modelo, se visualizó el error y exactitud para el set de entrenamiento y validación, generando un registro solamente para el caso de validación, además de la exactitud para el set de test. Por otro lado, también se guardaron matrices de confusión con el set de entrenamiento y de test, en conjunto a las métricas asociadas como la exactitud, precisión, y sensibilidad.

Finalmente, gracias a un código de apoyo para generar figuras, para cada función de activación, con y sin estandarización, se graficó la exactitud dentro del set de test versus la cantidad de neuronas ocultas, lo mismo se realizó con el error del set de validación pero con un gráfico 3D, añadiendo el número de iteraciones como variable extra, lo que nos permite analizar como se reduce el error durante el proceso de entrenamiento para cada caso. Por último, se generó una figura compuesta, que incluye la exactitud del test y el error de validación contra el número de neuronas ocultas, lo que muestra si existe alguna relación entre la variación de exactitud y el error.

Así, se generaron los gráficos y registros necesarios, gracias al código de apoyo *plotter.py*, para lograr seleccionar los mejores modelos según las métricas descritas, para luego realizar las últimas pruebas de rendimiento y análisis de las correspondientes matrices de confusión. Es posible observar de forma resumida y

simplificada el funcionamiento de los códigos mencionados en esta sección, en la figura 4.3.

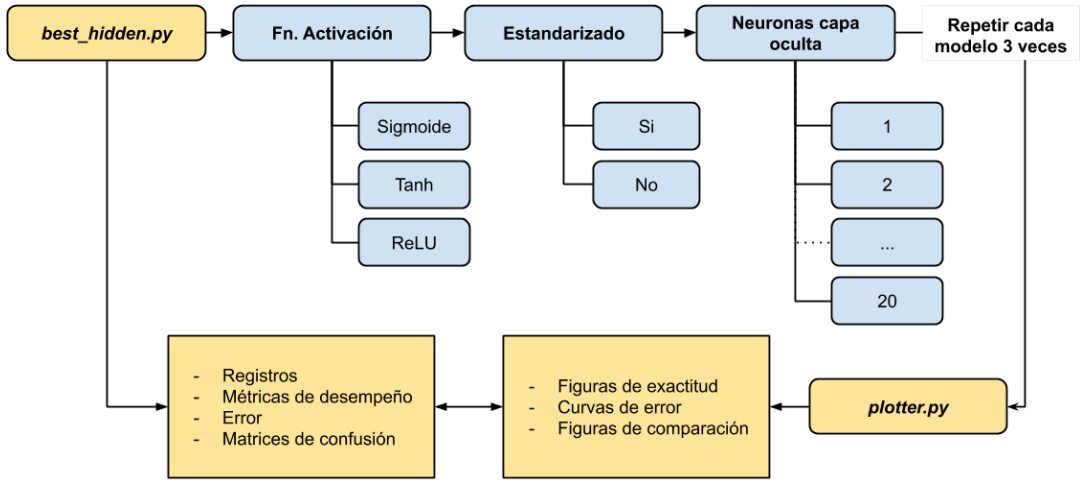


Figura 4.3.1: Esquema de la generación de modelos y registros para su posterior comparación, mediante las métricas y gráficos mencionados.

4.4. Prueba de desempeño en versión final

De manera manual, de cada set de modelos por función de activación, se seleccionó aquel que mostrara un menor error a la iteración 500, obteniendo así 6 versiones de MLP: 3 por función con estandarización y 3 sin estandarización. De estos 6, se realizó un test utilizando los datos que se separaron al principio (sección 4.2.2) en forma de traza continua (los 16 eventos restantes del entrenamiento).

El test consistió en ingresar secuencialmente a cada modelo la traza de un evento y una sección de ruido (obtenidos de la traza mencionada anteriormente), mediante un nuevo código llamado *run_ mlp.py*, analizando así la respuesta del STA/LTA, las neuronas de salida y la toma de decisión del modelo, con tal de ver su comportamiento en este nuevo formato de trazas, pues hay que tener en cuenta que hasta ahora el modelo solo ha trabajado con secciones de 2 segundos independientes. Este análisis visual tuvo por principal propósito, elegir un modelo que se comportara bien en ambas trazas con el mínimo de falsos positivos.

Por otro lado, las matrices de confusión generadas anteriormente con el código *best_ hidden.py* para cada modelo, sumado a las métricas de exactitud, precisión y

sensibilidad asociadas, fueron utilizadas como información de apoyo para intentar dar explicación a los posibles resultados de las pruebas finales de desempeño.

Finalmente, para expandir el caso de las trazas, y así generar un registro y estadística del desempeño de la red en un caso real, es que se utilizó el caso de enjambres sísmicos detectados originalmente por la estación COPA. El punto comparativo en este caso fue el STA/LTA, ya que al presentarse el modelo de MLP como una mejora del STA/LTA en el que se basa, es natural querer comparar los resultados entre ambos métodos.

Para lo anterior, se creó una nueva rutina (*testing_mlp.py*) y con ella se llevó un registro de los eventos catalogados, las detecciones de STA/LTA y del modelo MLP. En los dos últimos casos se habla de detecciones sospechosas pero no de eventos, pues para declarar un evento se necesita confirmación de más estaciones en el caso real. Es así, que se analizó la cantidad de falsos positivos entregados por STA/LTA y MLP, además de la capacidad de detectar los eventos ya catalogados por la red MLP, e incluso su habilidad para encontrar nuevos posibles eventos o sospechosos que no hubiesen sido catalogados anteriormente, notando obviamente que en este caso solamente se presume el nuevo evento ya que haría falta la confirmación con otras estaciones, lo cual no es posible al momento de la escritura de esta tesis, ya que sería necesario entrenar y homologar resultados sobre otra estación, para que estas se encuentren en igualdad de condiciones.

5. Resultados

5.1. Extracción de parámetros

Los primeros resultados obtenidos son predecesores a la generación de los modelos MLP, específicamente relacionados al proceso de extracción de patrones. Tal como fue mencionado en la sección 4.1, estos corresponden a los patrones de los eventos y ruidos sísmicos registrados, acumulando un total de 300 muestras distribuidas equitativamente en ambas clases. Un ejemplo de los patrones obtenidos se puede ver en la siguiente imagen:

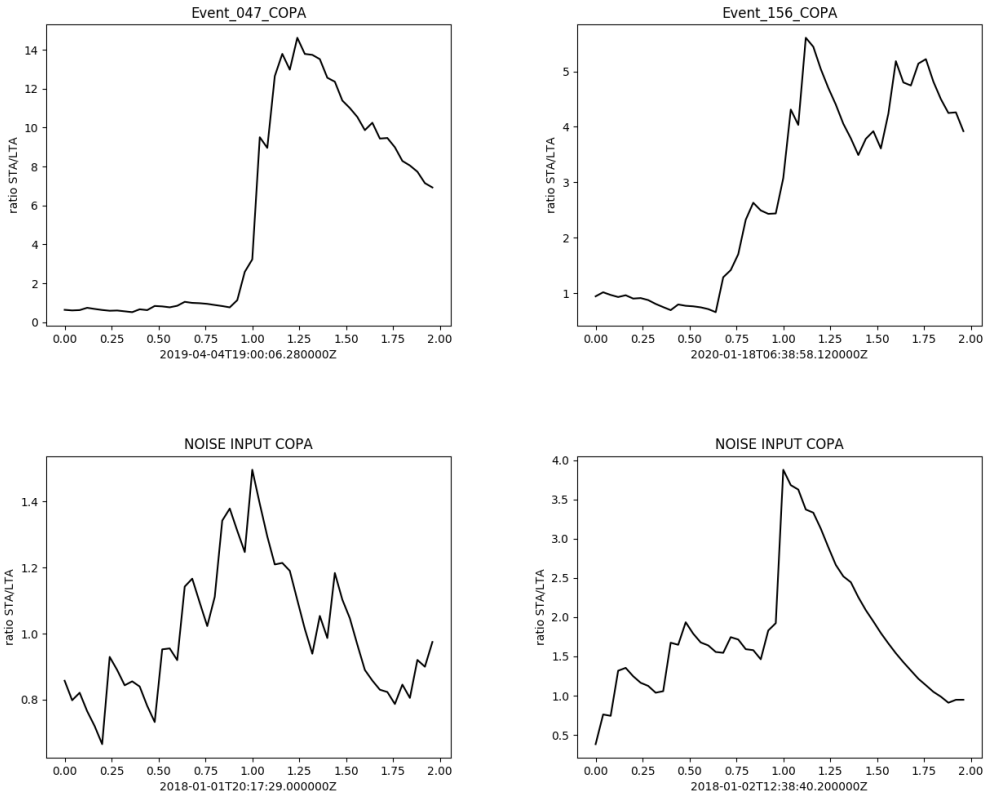


Figura 5.1.1: Superior: Ejemplos de muestras registradas para eventos 47 (Izquierda) y 156 (Derecha). Inferior: Muestras de ruido, para caso normal (Izquierda) y el caso con amplitud variable (Derecha).

Es posible observar la similitud de las muestras incluidas en la figura 5.1.1,

especialmente para el caso de eventos, con los patrones presentados en 4.1.3 característicos de cada clase (por ejemplo, la muestra de evento 47 presenta una forma similar al escalón típico de eventos). Recordar que estos son ejemplos de muestras que son acumuladas para generar la figura 5.1.1, sin considerar la amplitud de las muestras dado que son normalizados. En esta caso, es posible analizar la amplitud de la señal de STA/LTA y su diferencia entre las clases, tal como será ingresado a la red.

5.2. Red neuronal MLP

Los principales resultados de esta tesis provienen de la red MLP, su entrenamiento utilizando distintas variantes, los procesos de comparación y selección de los mejores modelos, y la evaluación de su desempeño en distintos aspectos. Estos resultados concernientes a los diversos ámbitos mencionados, son presentados a continuación.

5.2.1. Primeros modelos

Durante las primeras pruebas del algoritmo y el funcionamiento de la red MLP, y tras realizar los primeros entrenamientos de modelos, se obtuvieron algunos resultados preliminares, referentes a modelos de ejemplo basados en la arquitectura de red utilizada por (Wang and Teng, 1995). Estos primeros registros, si bien no tienen mayor influencia en posteriores análisis ni forman parte de la comparación y selección de modelos, sirven como ejemplo del uso y potencial del algoritmo, el funcionamiento de los scripts y métricas, además de presentar la posibilidad de obtener mejores modelos, a través de la optimización y variación de sus parámetros.

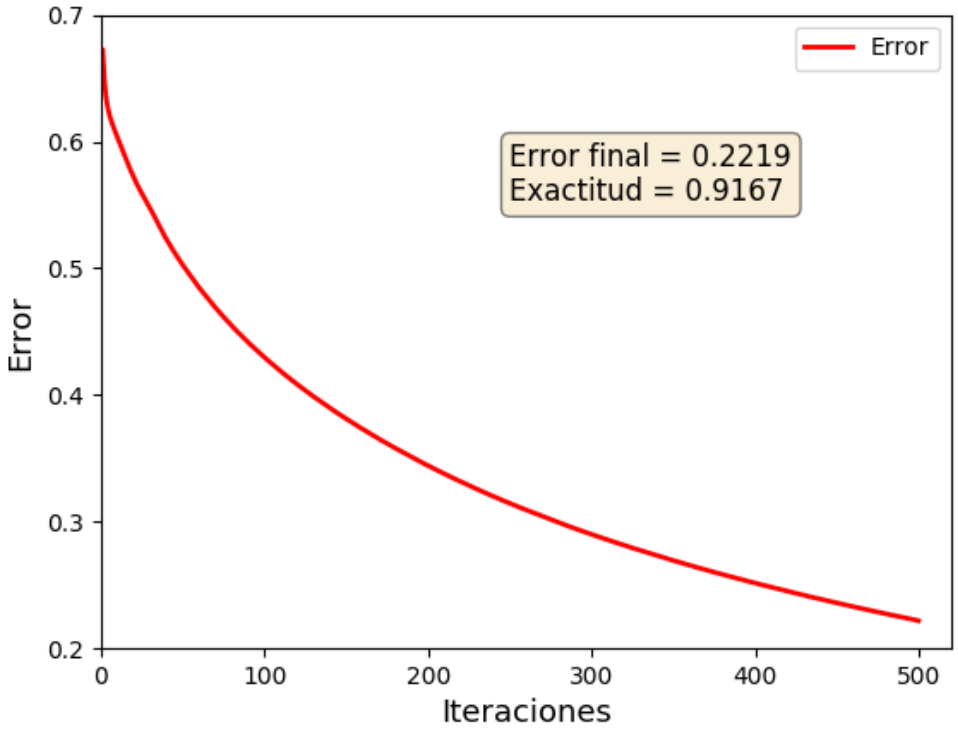


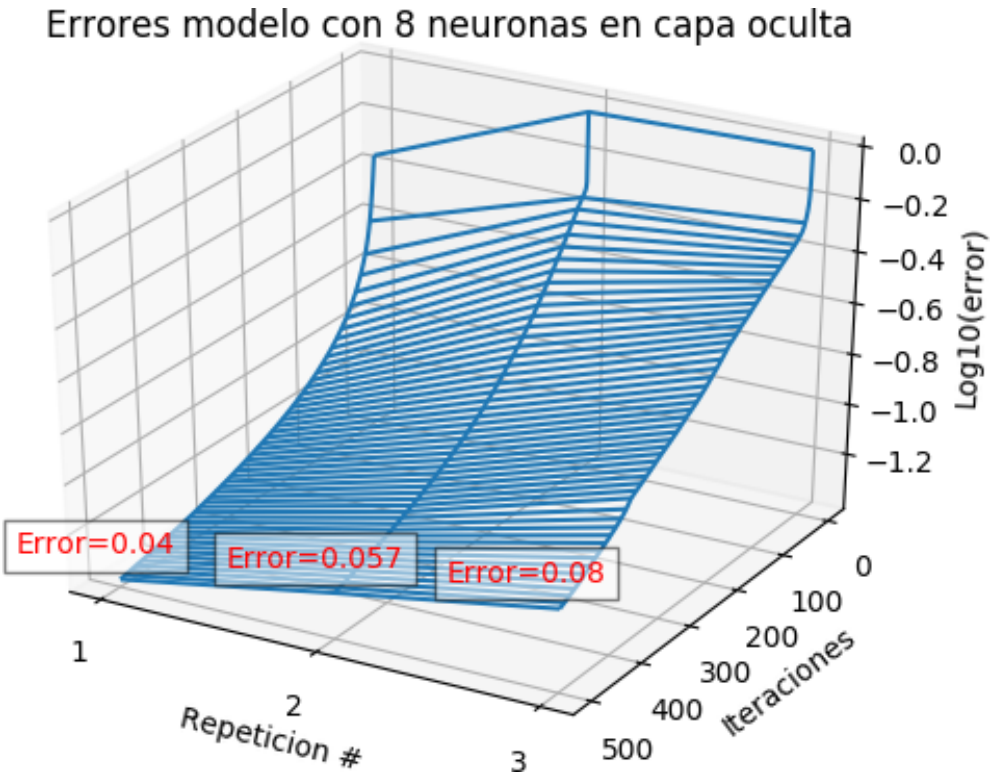
Figura 5.2.1: Curva de error y exactitud sobre set de validación de un modelo MLP con 8 neuronas en capa oculta y función de activación Sigmoide. Error final corresponde al error a la iteración 500, mientras que Exactitud es calculado mediante 3.2.12.

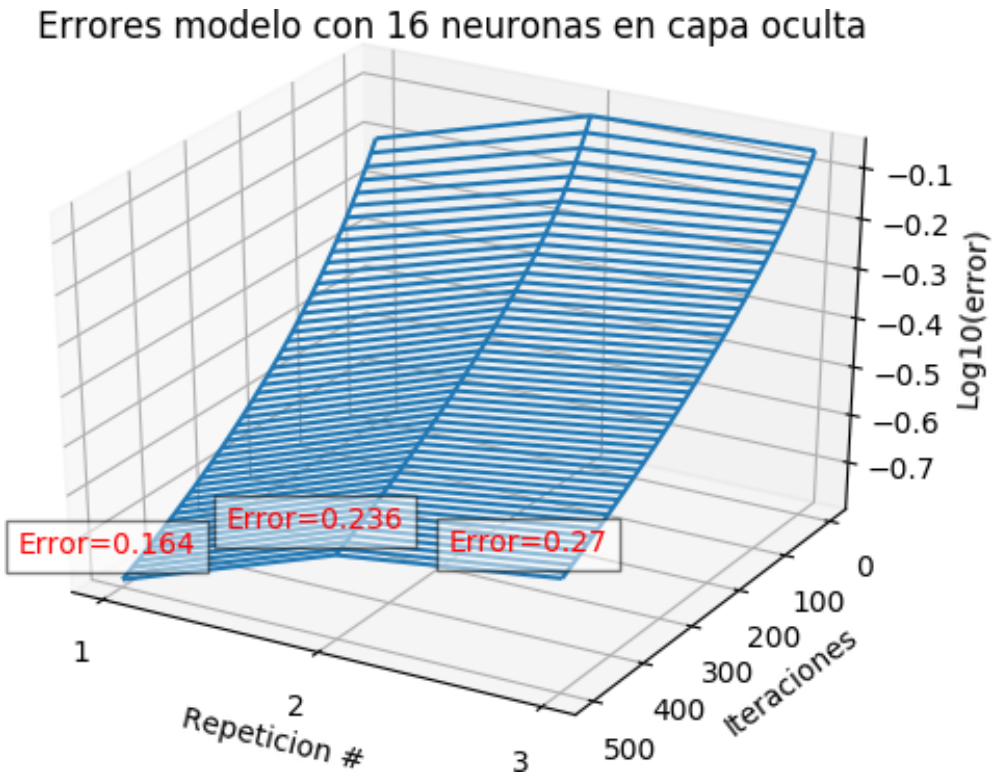
La curva de error que muestra la figura 5.2.1 se puede identificar como una caída exponencial, forma que se repetirá para los demás modelos, mientras que su valor final de 0.2219 corresponde a un valor que posiblemente puede ser mejorado (recordar que el error se basa en *Cross-entropy*, ver sección 3.2.5). Por otro lado, la exactitud obtenida (basada en el set de validación al igual que el error) se muestra con un alto valor de 91.67 %, pero es necesario obtener mayor información de su variación generando mayor cantidad de modelos, pues no se conoce su influencia aún.

5.2.2. Comparación de modelos

A la hora de generar múltiples modelos de MLP con distintas arquitecturas, y con el propósito de compararlos, se obtuvieron variadas métricas para contrastar su desempeño y quedarnos con los mejores modelos, tal como se mencionó en 4.3.

Frente a esto, uno de los primeros resultados en torno al aspecto de comparación son las curvas de error entre versiones de la misma arquitectura de red. A modo de ejemplo, y debido a la gran cantidad de modelos generados, se muestran a continuación variados casos de modelos y sus distintas versiones, y como se seleccionó el mejor de ellos.




Errores modelo con 16 neuronas en capa oculta
Error=0.164
Error=0.236
Error=0.27
Repeticion #
Iteraciones
Log10(error)
−0.1
−0.2
−0.3
−0.4
−0.5
−0.6
−0.7
1
2
3
0
100
200
300
400
500

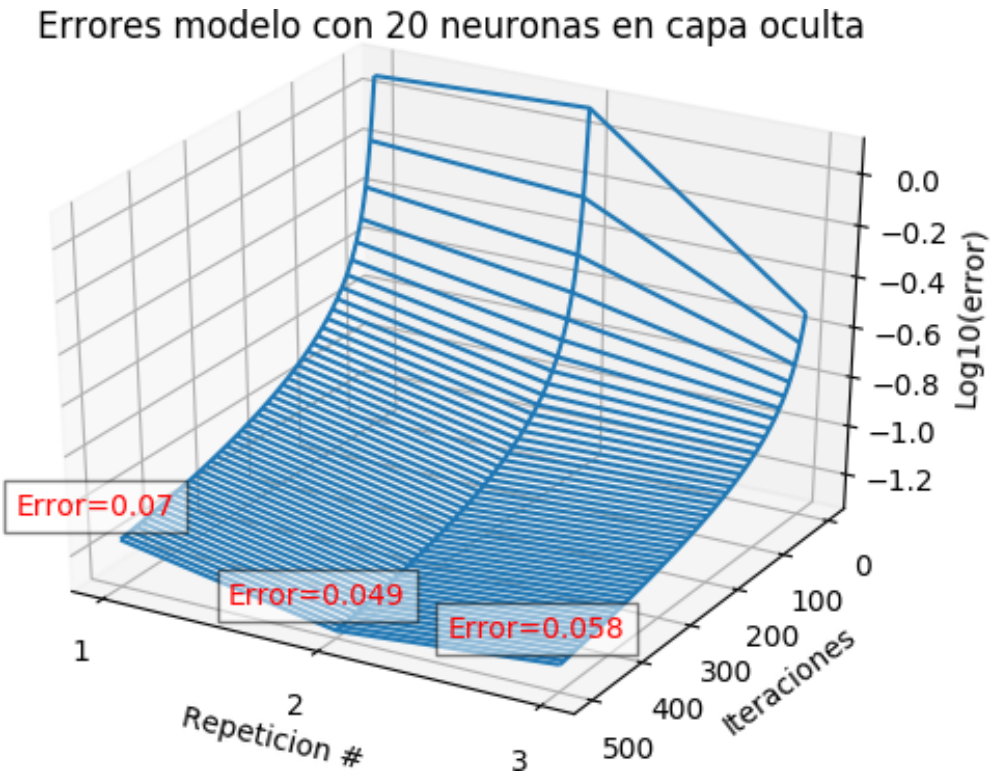


Figura 5.2.2: Curvas de error para las versiones de modelos, la escala del error es logarítmica (para mejor visualización) mientras que los valores de error en rojo, corresponden al valor normal. 1° imagen: Caso de función ReLU con 8 neuronas en capa oculta. 2° imagen: Caso de función Sigmoide con estandarización y 16 neuronas en capa oculta. 3° imagen: Caso de función Tanh con estandarización y 20 neuronas en capa oculta.

De la figura 5.2.2 es posible observar, para 3 casos de ejemplo, el progreso del error a través de sus distintas versiones durante el proceso de entrenamiento, hasta llegar a los valores finales denotados en rojo. A partir de esta información, es que se selecciona al modelo con menor error de sus 3 versiones. Al terminar este primer proceso de selección, quedan 20 modelos restantes para cada función de activación, una por cada número de neuronas, que pasan a un proceso de comparación posterior.

Para mayor organización, se mostrarán los resultados de la comparación por función de activación a continuación:

- **Función ReLU sin estandarización de datos:**

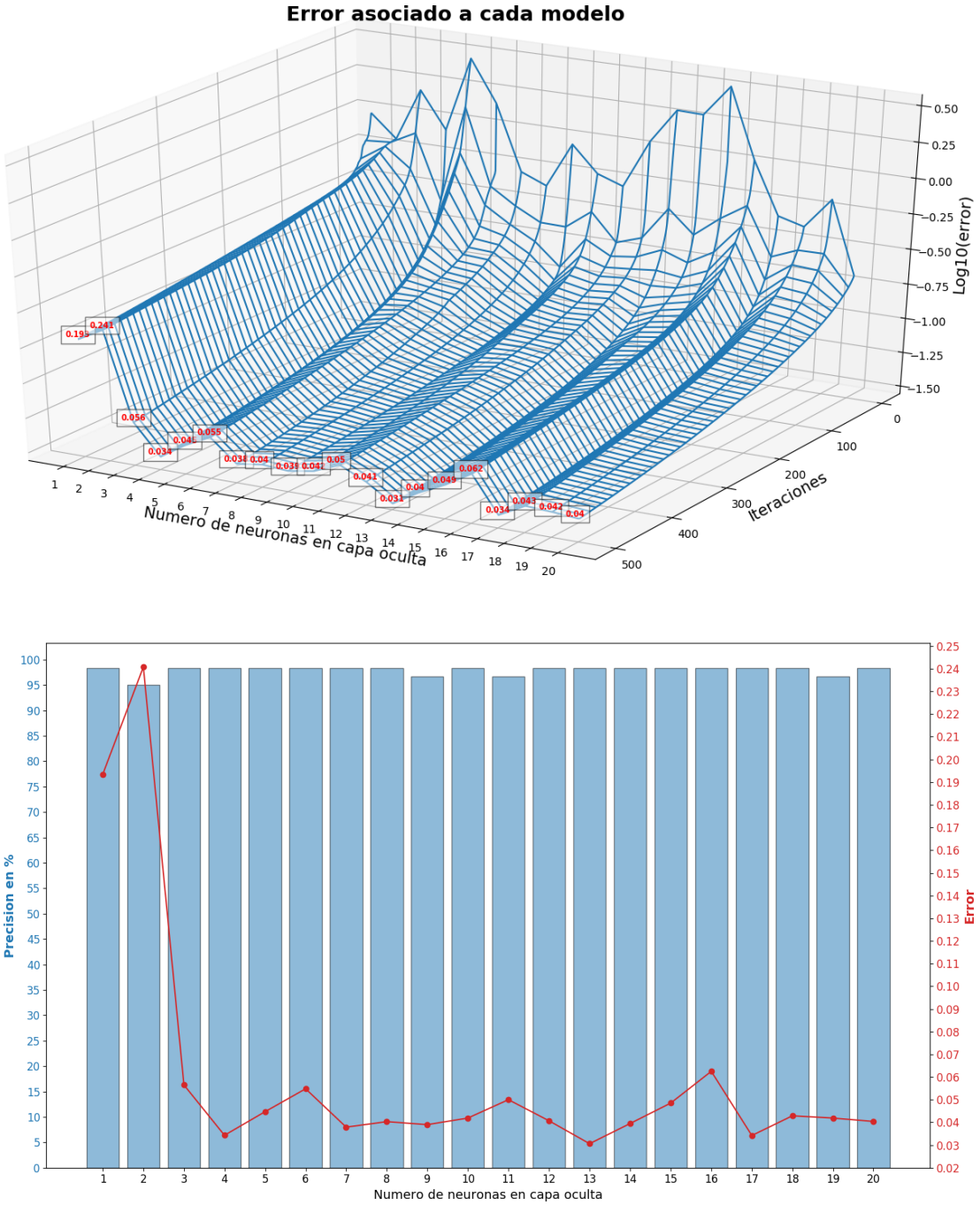
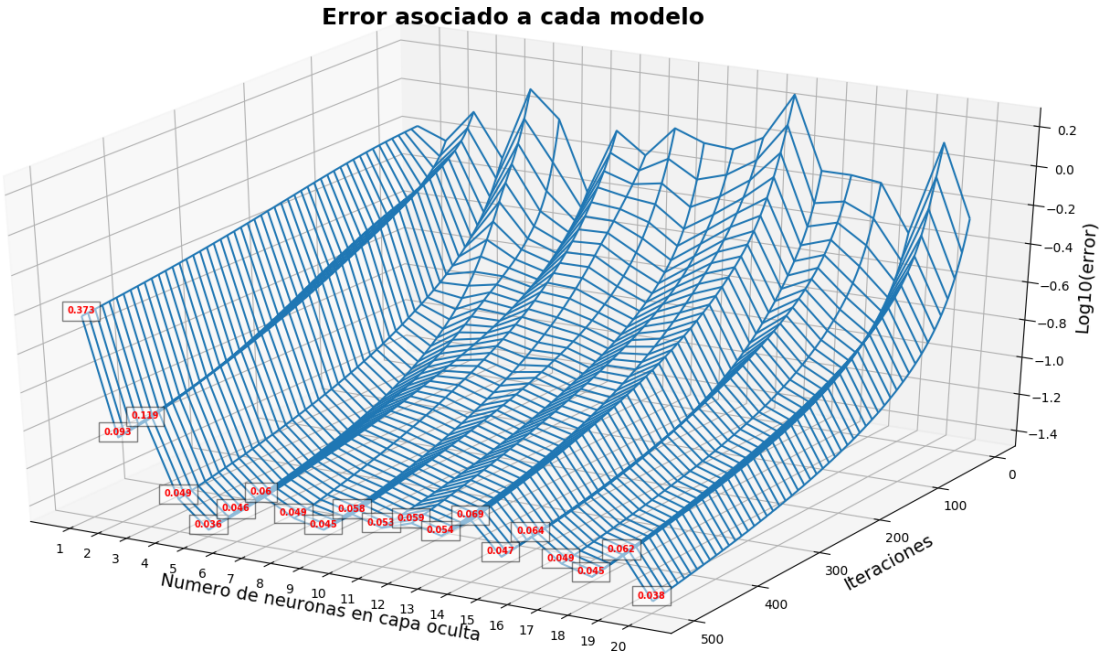


Figura 5.2.3: Primera imagen: curvas de error y error final asociado a cada modelo según número de neuronas en la capa oculta para el caso de la función ReLU sin estandarizar. Segunda imagen: En rojo el error final para cada modelo, mientras que en barras azules la exactitud asociada.

Para este caso, de la figura 5.2.3 es posible extraer información importante sobre el error y su evolución según las neuronas en la capa oculta. En general, el error es muy pequeño a partir de 4 neuronas hasta 20, variando alrededor

de 0.04, además, este decae rápidamente dentro de las 500 iteraciones en el entrenamiento, por lo que posiblemente aquellos correspondan a sus valores estables. Por otro lado, la exactitud se mantiene alta y constante para la gran mayoría de los modelos, lo que nos dice que la exactitud no tiene mucha relación con el error, y tampoco con la cantidad de neuronas.

- **Función ReLU con estandarización de datos:**



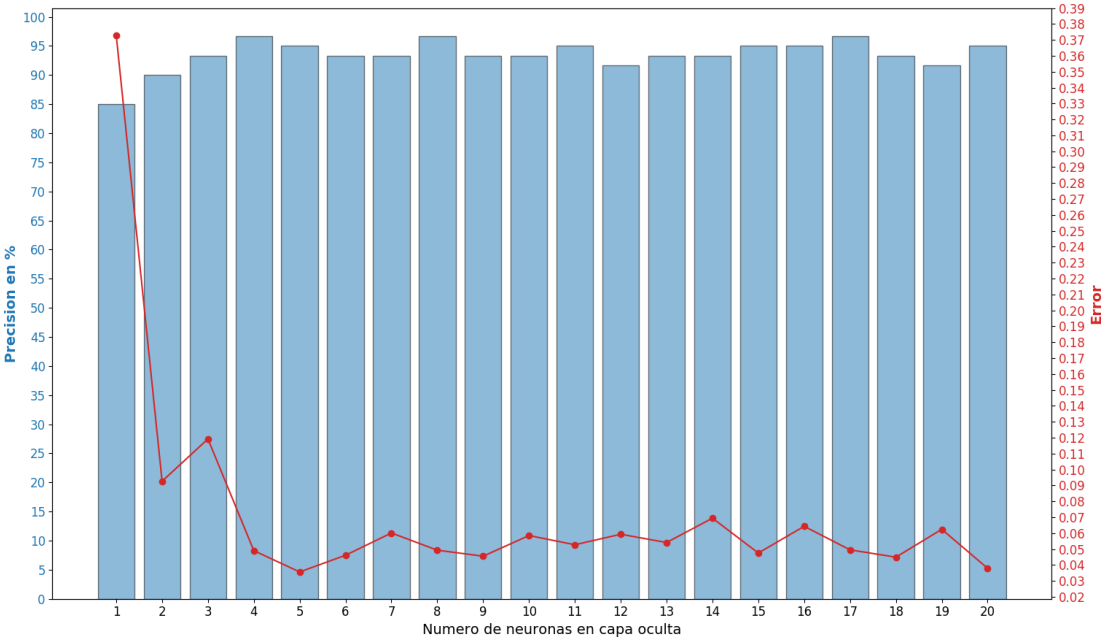


Figura 5.2.4: Primera imagen: curvas de error y error final asociado a cada modelo según número de neuronas en la capa oculta para el caso de la función ReLU con estandarización. Segunda imagen: En rojo el error final para cada modelo, mientras que en barras azules la exactitud asociada.

De manera similar a lo visto en la figura 5.2.3, la figura 5.2.4 muestra la evolución del error, alcanzando un valor estable a partir de las 4 neuronas, con la diferencia de que este demora un poco más en estabilizarse a lo largo de las iteraciones, dando origen a curvas menos pronunciadas. En cuanto a la exactitud exhibida por los modelos, esta se muestra más variable pero manteniéndose mayor a 90 % para la mayoría de los modelos, además, no es posible observar mayor influencia de la estandarización de datos por lo menos para la función ReLU.

- **Función Sigmoide sin estandarización de datos:**

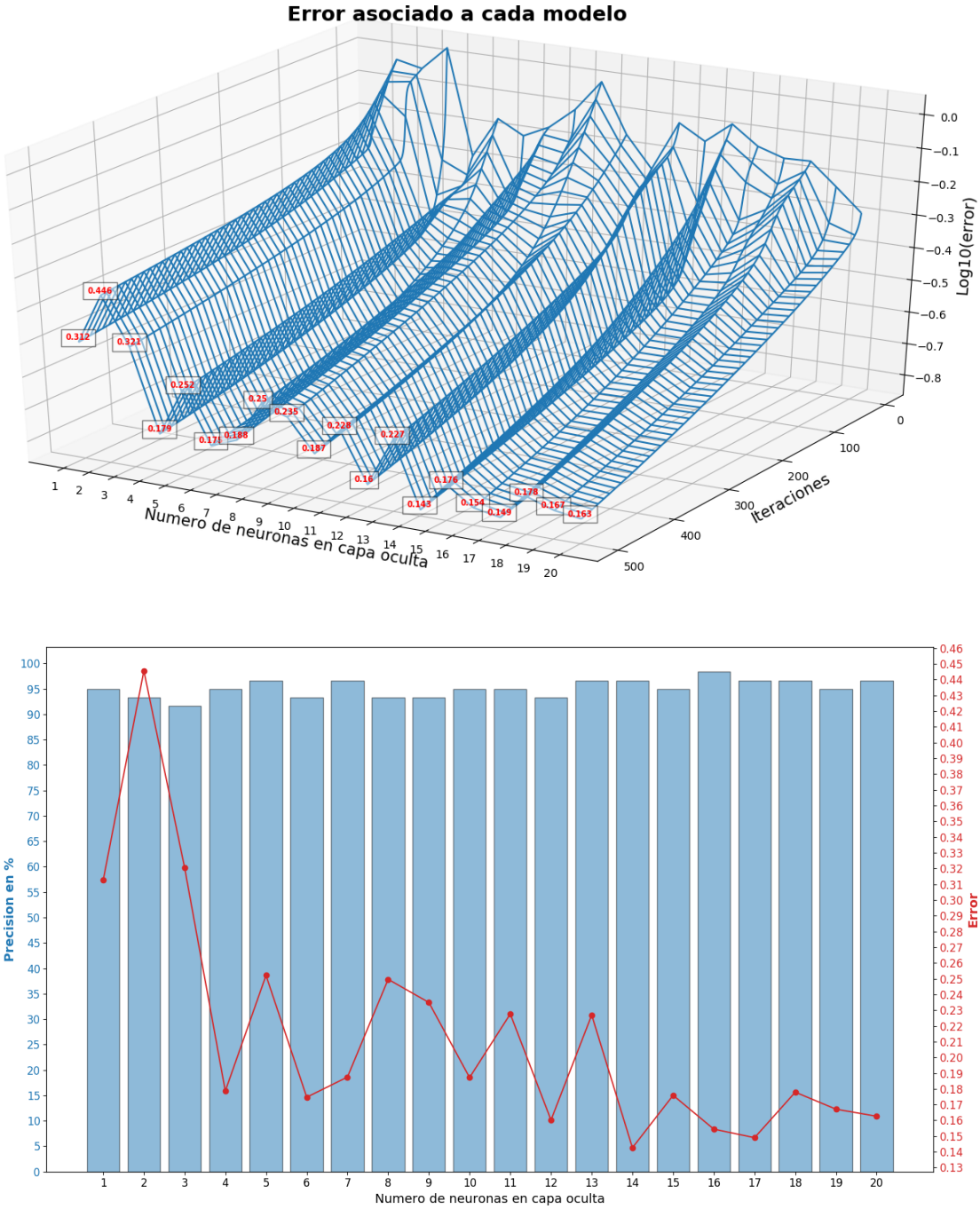
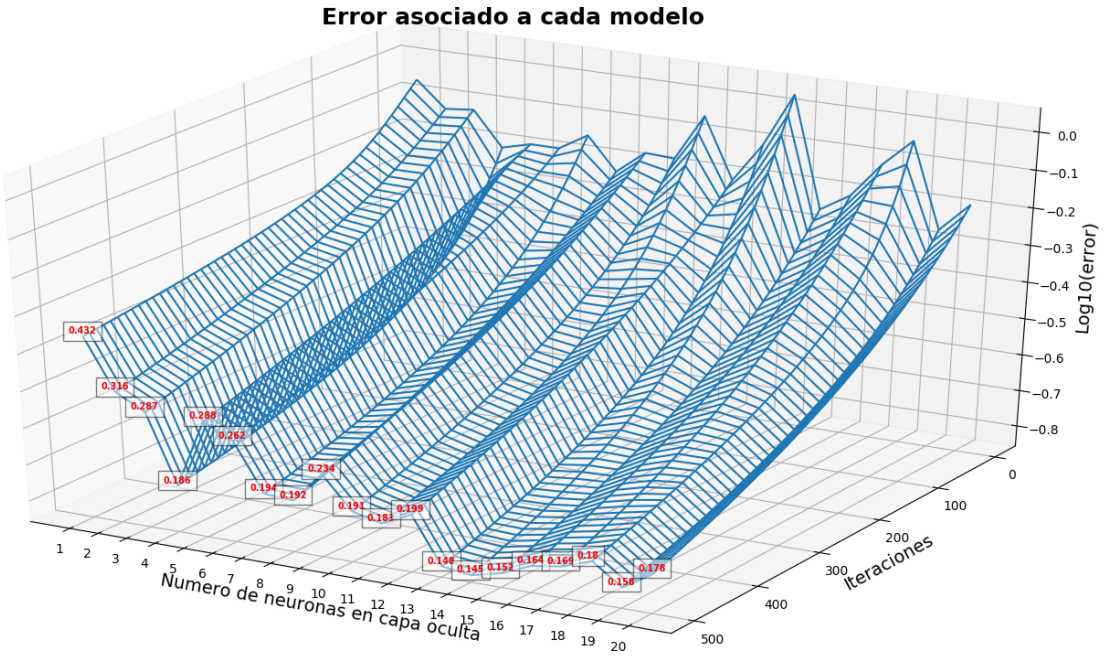


Figura 5.2.5: Primera imagen: curvas de error y error final asociado a cada modelo según número de neuronas en la capa oculta para el caso de la función Sigmoide sin estandarizar. Segunda imagen: En rojo el error final para cada modelo, mientras que en barras azules la exactitud asociada.

La figura 5.2.5 presenta diferencias con lo ya mostrado para la función ReLU. En primer lugar, es posible observar una tendencia a la baja del error según se incrementan las neuronas en la capa oculta, además, el error para las

neuronas más altas, es más elevado en comparación a lo visto en las figuras 5.2.3 y 5.2.4, con un orden de magnitud de diferencia. En segundo lugar, la disminución del error al aumentar las iteraciones es relativamente suave, en contraste a las curvas de error en la función ReLU. Por el lado de la exactitud, este se mantiene en altos niveles a pesar de todo, pero más cerca del 90 % que del 100 %, como fue el caso del ReLU no estandarizado.

- **Función Sigmoide con estandarización de datos:**



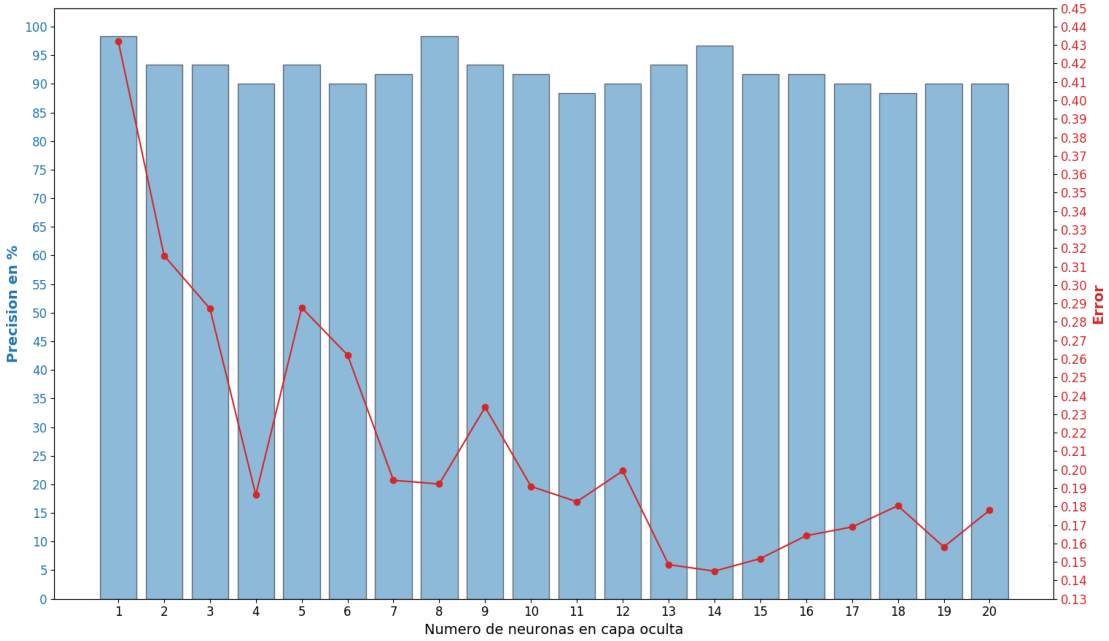


Figura 5.2.6: Primera imagen: curvas de error y error final asociado a cada modelo según número de neuronas en la capa oculta para el caso de la función Sigmoide con estandarización. Segunda imagen: En rojo el error final para cada modelo, mientras que en barras azules la exactitud asociada

Como se puede notar de la figura 5.2.6, las similitudes con su contra-parte no estandarizada son bastantes, desde los rangos de valores del error al comportamiento de las curvas de error en la iteración 500, mostrando similares valores para las distintas neuronas, presentando como único detalle una leve mayor suavidad en estas curvas de error. La exactitud, por otro lado, presenta mayor variabilidad, manteniéndose cercano al 90 % con pocos modelos superando el 95 %, a diferencia de su versión no estandarizada.

- **Función Tanh sin estandarización de datos:**

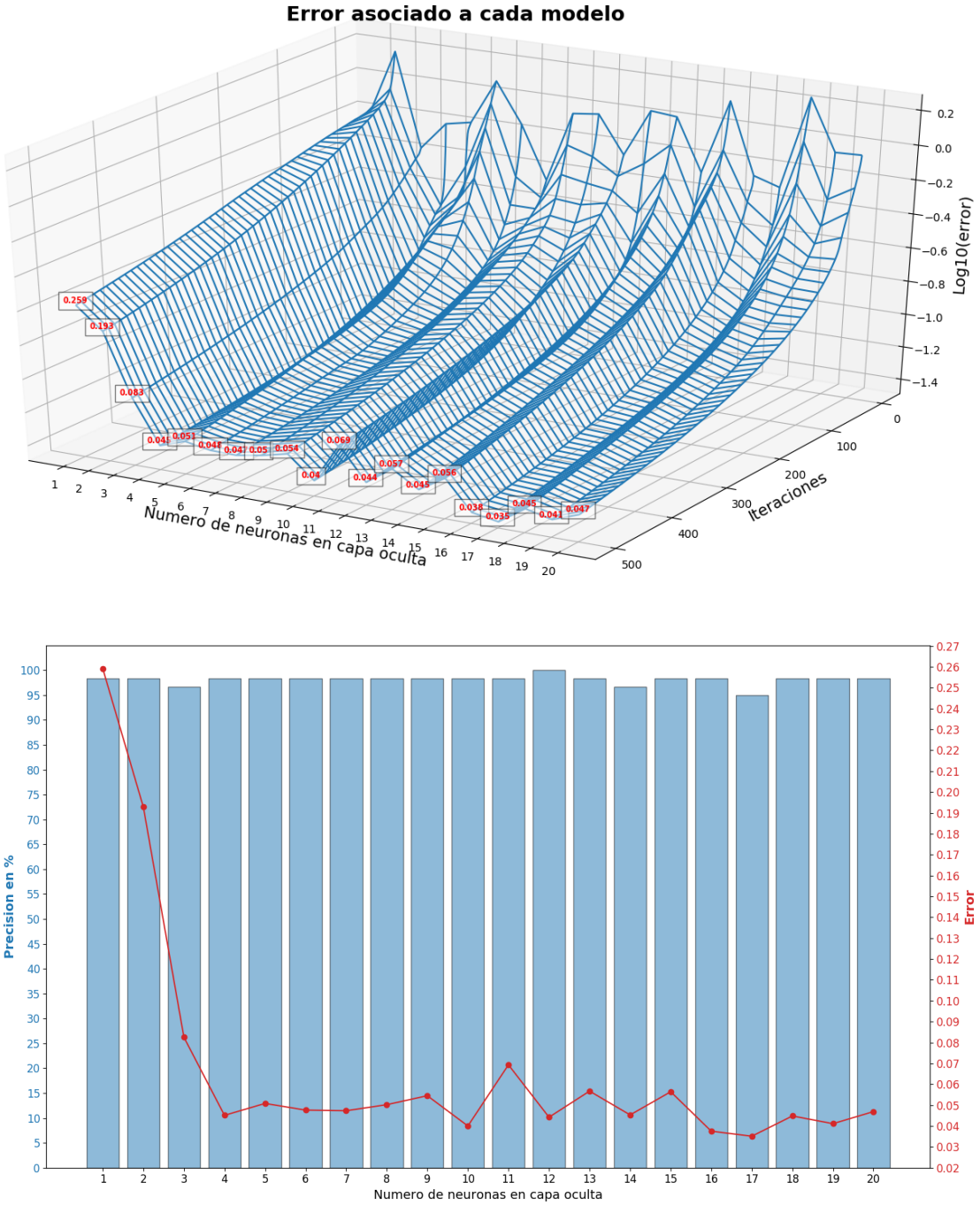


Figura 5.2.7: Primera imagen: curvas de error y error final asociado a cada modelo según número de neuronas en la capa oculta para el caso de la función Tanh sin estandarizar. Segunda imagen: En rojo el error final para cada modelo, mientras que en barras azules la exactitud asociada.

Se extrae de la figura 5.2.7 que para el caso de la función Tanh, los errores son bajos y estables para la mayoría de los modelos, al igual que la exactitud, por lo que no es posible establecer una relación con la cantidad de neuronas.

En cuanto a la caída del error con las iteraciones, esta es similar a las de las otras funciones presentadas.

- **Función Tanh con estandarización de datos:**

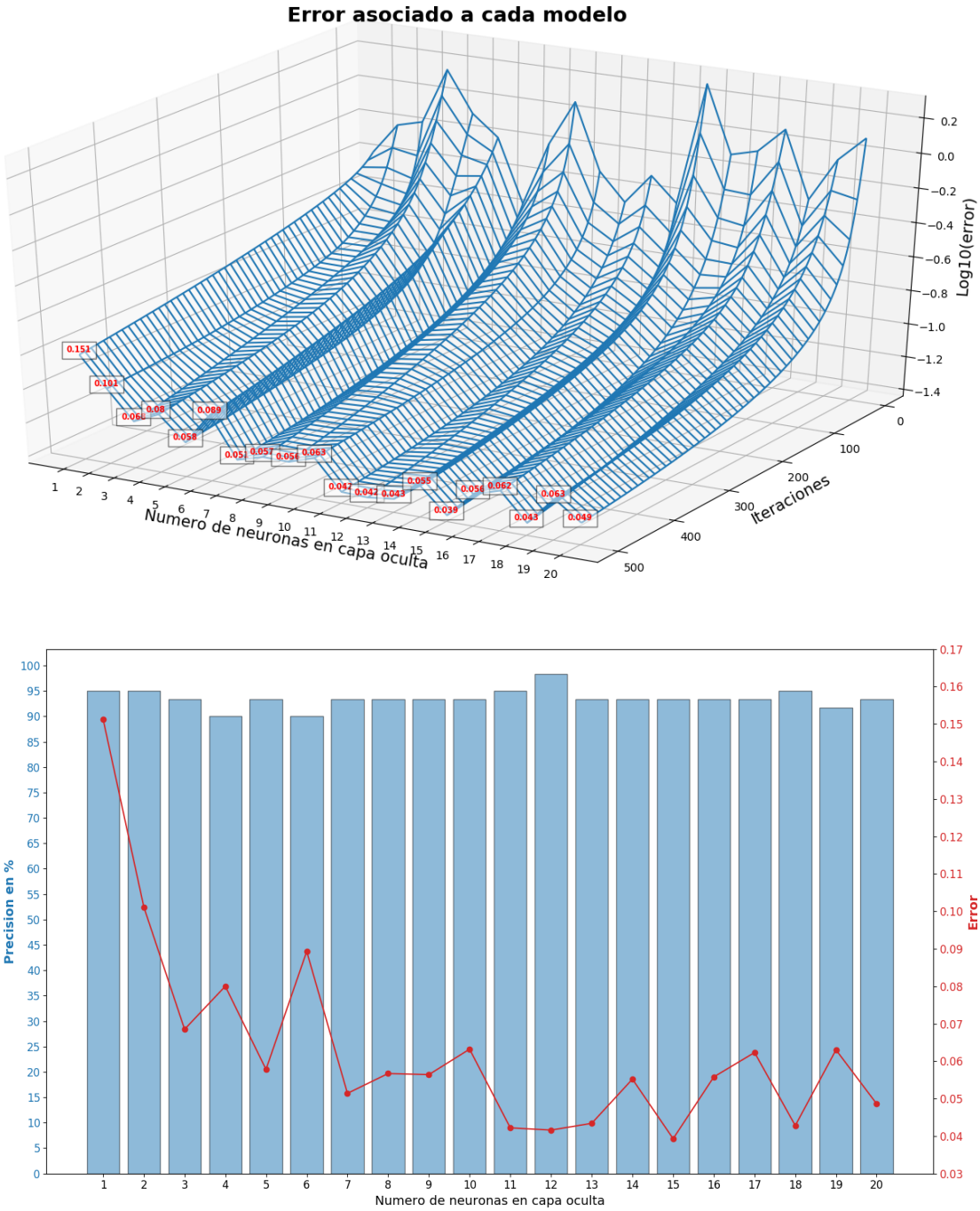


Figura 5.2.8: Primera imagen: curvas de error y error final asociado a cada modelo según número de neuronas en la capa oculta para el caso de la función Tanh con estandarización. Segunda imagen: En rojo el error final para cada modelo, mientras que en barras azules la exactitud asociada.

Tomando lo visto en el caso de la función ReLU (figuras 5.2.3 - 5.2.4), se puede notar que sus resultados son similares a lo obtenido en las figuras 5.2.7 y 5.2.8. Los valores del error final son muy levemente mayores para Tanh en comparación a ReLU, al igual que el pronunciamiento de la curva, que parece ser levemente mayor y con más suavidad, la única diferencia observable está asociada a los primeros modelos (1 a 4 neuronas) en que el valor de error final es mucho más bajo a lo que ya se había establecido como tendencia en otros casos. Por el lado de la exactitud, es posible notar mayores diferencias con los resultados de datos no estandarizados, pues estos valores se quedan entre 90 y 95 %.

Para finalizar, se debe volver a destacar las similitudes entre los resultados de las funciones ReLU y Tanh en cuanto a error, en donde la función Sigmoide registra errores más elevados. De la estandarización y no estandarización de los datos, es posible observar una leve incidencia en la exactitud, mostrándose valores levemente menores para los modelos estandarizados, pero fuera de eso los resultados son similares para las distintas funciones.

A modo de resumen, a continuación una tabla que concentra toda la información expuesta en esta subsección:

N°	ReLU	ReLU Std.	Sigmoide	Sigmoide Std.	Tanh	Tanh Std
1	0.193	0.373	0.312	0.432	0.259	0.151
2	0.241	0.093	0.446	0.316	0.193	0.101
3	0.056	0.119	0.321	0.287	0.083	0.068
4	0.034	0.049	0.179	0.186	0.045	0.08
5	0.045	0.036	0.252	0.288	0.051	0.058
6	0.055	0.046	0.175	0.262	0.048	0.089
7	0.038	0.06	0.188	0.194	0.047	0.051
8	0.04	0.049	0.25	0.192	0.05	0.057
9	0.039	0.045	0.235	0.234	0.054	0.056
10	0.042	0.058	0.187	0.191	0.04	0.063
11	0.05	0.053	0.228	0.183	0.069	0.042
12	0.041	0.059	0.16	0.199	0.044	0.042
13	0.031	0.054	0.227	0.148	0.057	0.043
14	0.04	0.069	0.143	0.145	0.045	0.055
15	0.049	0.047	0.176	0.152	0.056	0.039
16	0.062	0.064	0.154	0.164	0.038	0.056
17	0.034	0.049	0.149	0.169	0.035	0.062
18	0.043	0.045	0.178	0.18	0.045	0.043
19	0.042	0.062	0.167	0.158	0.041	0.063
20	0.04	0.038	0.163	0.178	0.047	0.049

Tabla 5.2.1: Tabla resumen para el error final, según la función de activación utilizada. N° se refiere al número de neuronas en capa oculta. Std se refiere a modelos estandarizados.

N°	ReLU	ReLU Std.	Sigmoide	Sigmoide Std.	Tanh	Tanh Std
1	98.3 %	85.0 %	95.0 %	98.3 %	98.3 %	95.0 %
2	95.0 %	90.0 %	93.3&	93.3 %	98.3 %	95.0 %
3	98.3 %	93.3 %	91.7 %	93.3 %	96.7 %	93.3 %
4	98.3 %	96.7 %	95.0 %	90.0 %	98.3 %	90.0 %
5	98.3 %	95.0 %	96.7 %	93.3 %	98.3 %	93.3 %
6	98.3 %	93.3 %	93.3 %	90.0 %	98.3 %	90.0 %
7	98.3 %	93.3 %	96.7 %	91.7 %	98.3 %	93.3 %
8	98.3 %	96.7 %	93.3 %	98.3 %	98.3 %	93.3 %
9	96.7 %	93.3 %	93.3 %	93.3 %	98.3 %	93.3 %
10	98.3 %	93.3 %	95.0 %	91.7 %	98.3 %	93.3 %
11	96.7 %	95.0 %	95.0 %	88.3 %	98.3 %	95.0 %
12	98.3 %	91.7 %	93.3 %	90.0 %	100.0 %	98.3 %
13	98.3 %	93.3 %	96.7 %	93.3 %	98.3 %	93.3 %
14	98.3 %	93.3 %	96.7 %	96.7 %	96.7 %	93.3 %
15	98.3 %	95.0 %	95.0 %	91.7 %	98.3 %	93.3 %
16	98.3 %	95.0 %	98.3 %	91.7 %	98.3 %	93.3 %
17	98.3 %	96.7 %	96.7 %	90.0 %	95.0 %	93.3 %
18	98.3 %	93.3 %	96.7 %	88.3 %	98.3 %	95.0 %
19	96.7 %	91.7 %	95.0 %	90.0 %	98.3 %	91.7 %
20	98.3 %	95.0 %	96.7 %	90.0 %	98.3 %	93.3 %

Tabla 5.2.2: Tabla resumen para la exactitud (en porcentaje) asociada a cada modelo, según la función de activación utilizada. N° se refiere al número de neuronas en capa oculta.

5.2.3. Matrices de confusión

En base a la información presentada en la subsección anterior, se seleccionaron los modelos candidatos que presentaran un menor error, y que corresponden a aquellos con un número alto de neuronas en la capa oculta o mayor complejidad. Para complementar los resultados de estos modelos candidatos, se obtuvo información de apoyo para cada uno de estos, correspondiente a sus matrices de confusión y métricas asociadas, lo cual fue mencionado en la sección 3.2.6. Las siguientes figuras y tablas resumen estos resultados:

Actual \ Predicción	Ruido	Evento
Ruido	26	0
Evento	1	33

(a) ReLU no estandarizado [13/2].

Actual \ Predicción	Ruido	Evento
Ruido	26	0
Evento	3	31

(b) ReLU estandarizado [20/3].

Actual \ Predicción	Ruido	Evento
Ruido	26	0
Evento	2	32

(c) Sigmoide no estandarizado [14/1].

Actual \ Predicción	Ruido	Evento
Ruido	26	0
Evento	2	32

(d) Sigmoide estandarizado [14/3].

Actual \ Predicción	Ruido	Evento
Ruido	26	0
Evento	3	31

(e) Tanh no estandarizado [17/2].

Actual \ Predicción	Ruido	Evento
Ruido	26	0
Evento	4	30

(f) Tanh estandarizado [15/3].

Tabla 5.2.3: Matrices de confusión asociadas a los modelos candidatos, basadas en el set de datos de test. Los números entre corchetes, en la forma [-/-] se refieren a la identificación del modelo candidato, por ejemplo, [13/2] se refiere al modelo de 13 neuronas, en su segunda versión (de 3). La estructura hace referencia a la figura 3.2.4.

Función activación	Exactitud	Precisión	Sensibilidad
ReLU	0.983	1.0	0.971
ReLU Std.	0.95	1.0	0.912
Sigmoide	0.967	1.0	0.941
Sigmoide Std.	0.967	1.0	0.941
Tanh	0.95	1.0	0.912
Tanh Std.	0.933	1.0	0.882

Tabla 5.2.4: Métricas asociadas a las matrices de confusión, basadas en el set de test.

De las dos tablas presentadas, es posible notar que los modelos candidatos presentan buenos resultados en cuanto a sus matrices de confusión y métricas correspondiente. Para todos los modelos la precisión es máxima, mostrando diferencias en la exactitud y sensibilidad. En este aspecto, parece ser que los modelos estandarizados son levemente peores en estos ámbitos, a excepción de los modelos con función Sigmoide, que exhiben similares resultados con las otras funciones de activación. Por último, cabe notar que para el caso de ReLU no

estandarizado el mejor modelo correspondía al de 5 neuronas en capa oculta (tercera versión), pero para evitar modelos de baja complejidad se utilizó el siguiente, de 20 neuronas (tercera versión).

5.2.4. Prueba de modelos

Para la última comparación de los modelos seleccionados, según fue mencionado en 4.4, se utilizan trazas de un evento registrado en la estación COPA, con fecha 30/03/2020-10:25:15, como también una sección de ruido registrado el 30/03/2020-00:00:27, para evaluar el comportamiento en datos continuos por parte de los modelos, además, de la cantidad de falsos positivos que puedan arrojar. La estructura de las imágenes que se mostrarán a continuación, viene de la siguiente forma:

- Superior: Traza del registro de la estación COPA.
- Medio-Superior: STA/LTA calculado a partir de la traza, con Threshold/Trigger ON (rojo) y Threshold/Trigger OFF (azul) graficados tal como se vería con el Software de Triggering de ObsPy, a modo de referencia de la detección con STA/LTA.
- Medio-Inferior: Respuesta de las neuronas de salida de la red MLP, en rojo la salida de la neurona asociada al ruido, mientras que en azul la salida correspondiente a evento, de modo que, cuando la salida de evento supera al valor asociado a ruido, se marca una predicción de evento.
- Inferior: Clasificación de cada punto de la traza según la predicción de la red MLP, con 0 = ruido y 1 = evento, declarados con los mismos criterios ya mencionados anteriormente según las salidas de las neuronas. La recta vertical roja marca las predicciones de eventos.

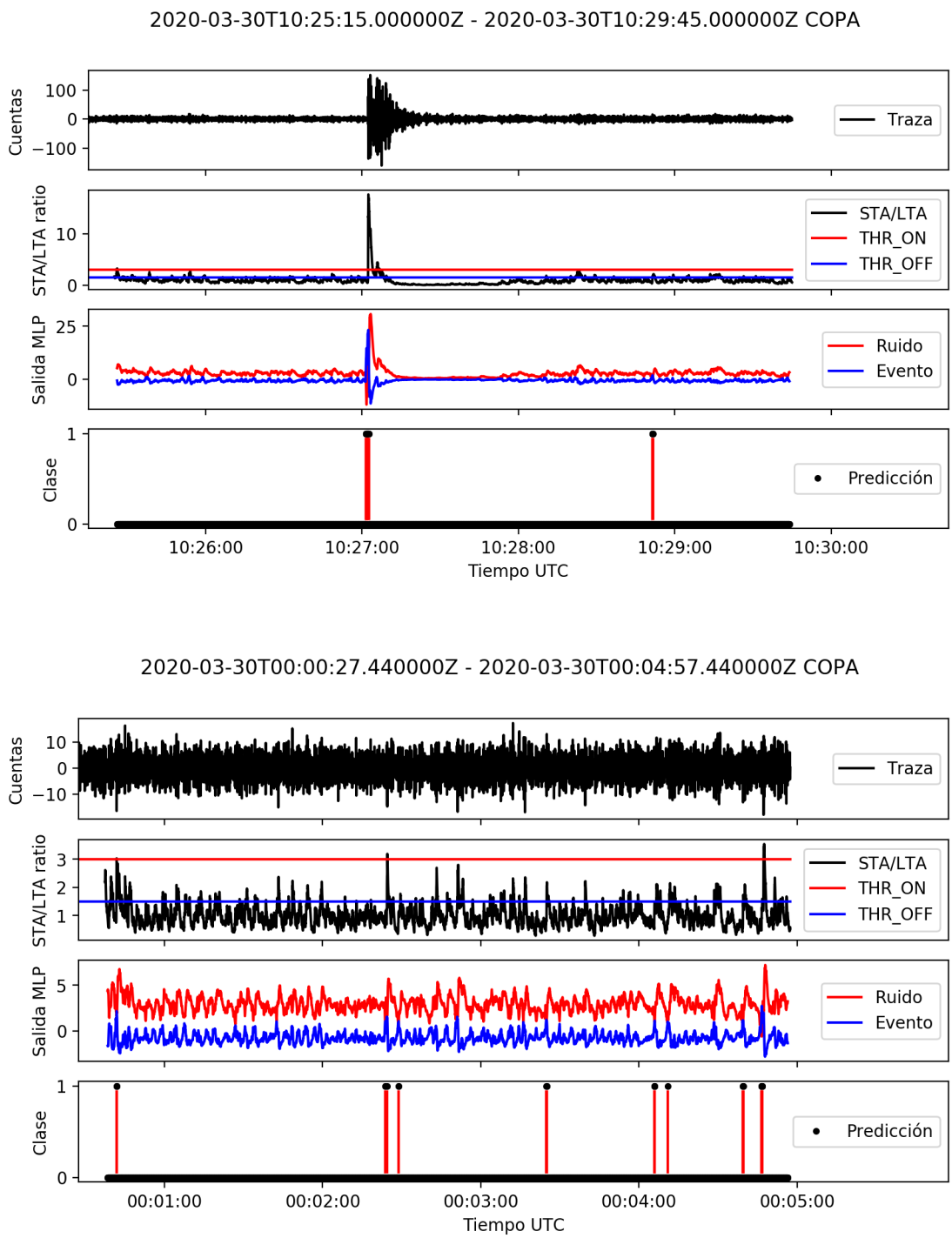


Figura 5.2.9: MLP con función ReLU y 13 neuronas capa oculta. Primera imagen: Respuesta del modelo a una traza continua de un evento. Segunda imagen: Repuesta para una traza de ruido sísmico.

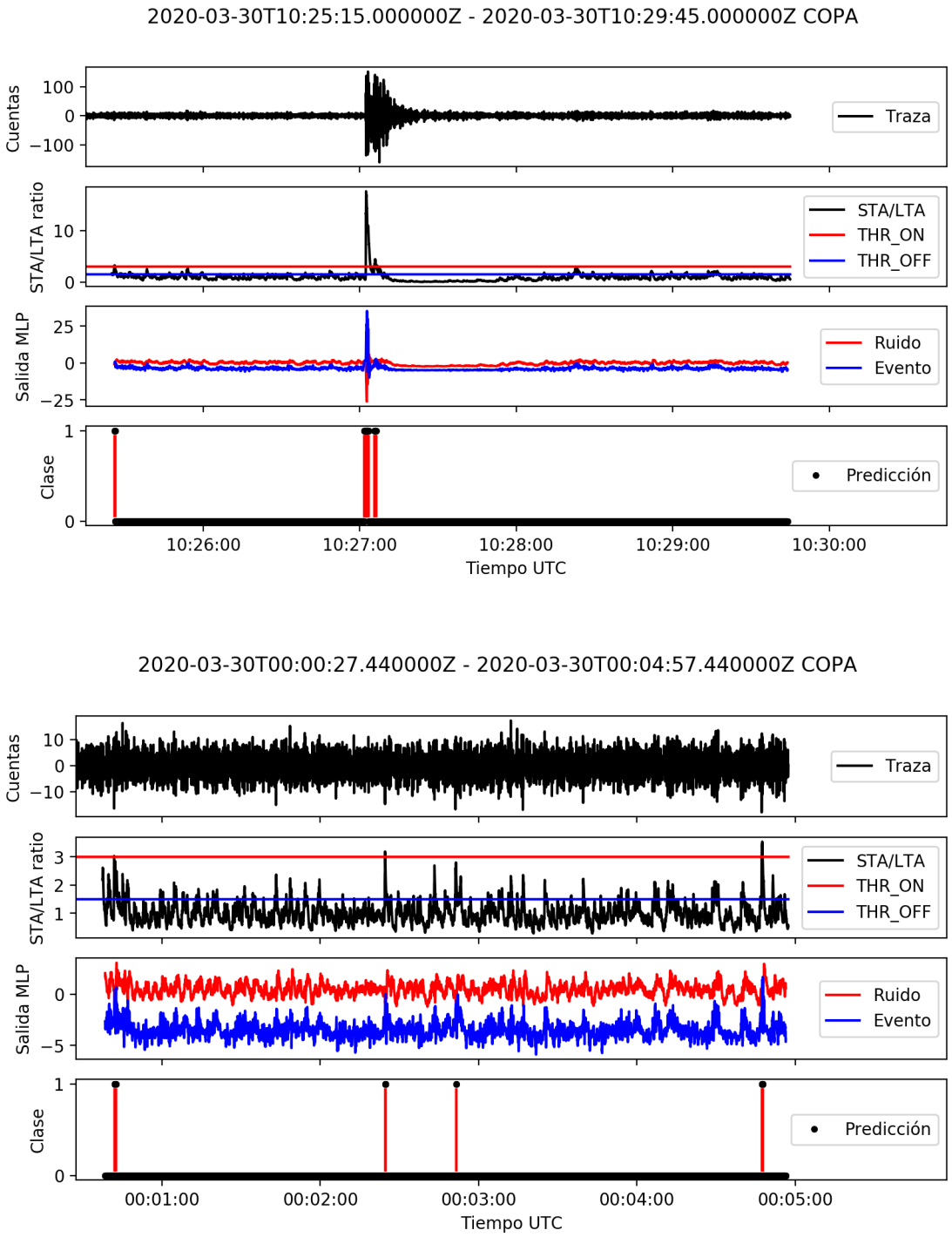


Figura 5.2.10: MLP con función ReLU estandarizado y 20 neuronas capa oculta. Primera imagen: Respuesta del modelo a una traza continua de un evento. Segunda imagen: Repuesta para una traza de ruido sísmico.

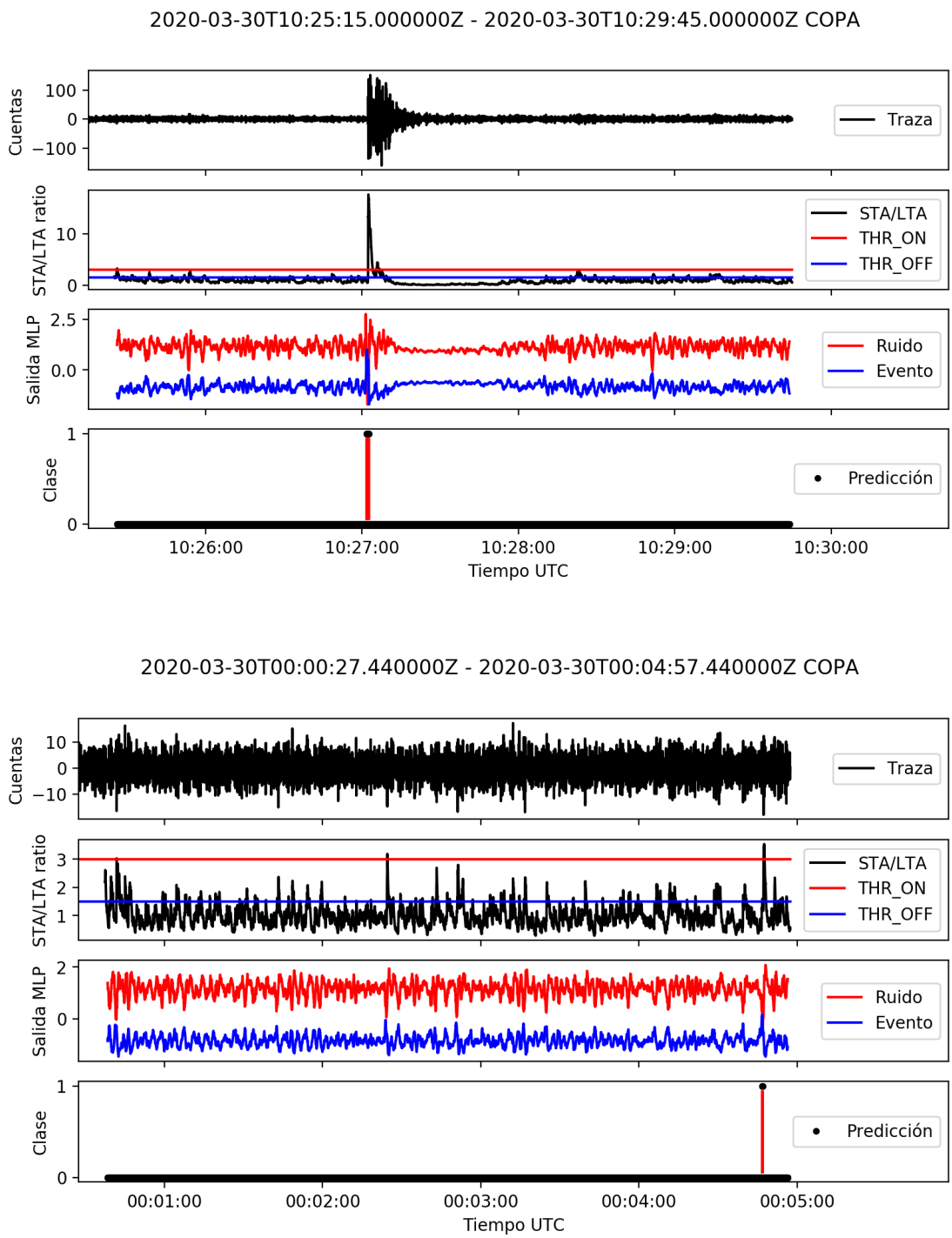


Figura 5.2.11: MLP con función Sigmoide y 14 neuronas capa oculta.Primera imagen: Respuesta del modelo a una traza continua de un evento. Segunda imagen: Repuesta para una traza de ruido sísmico.

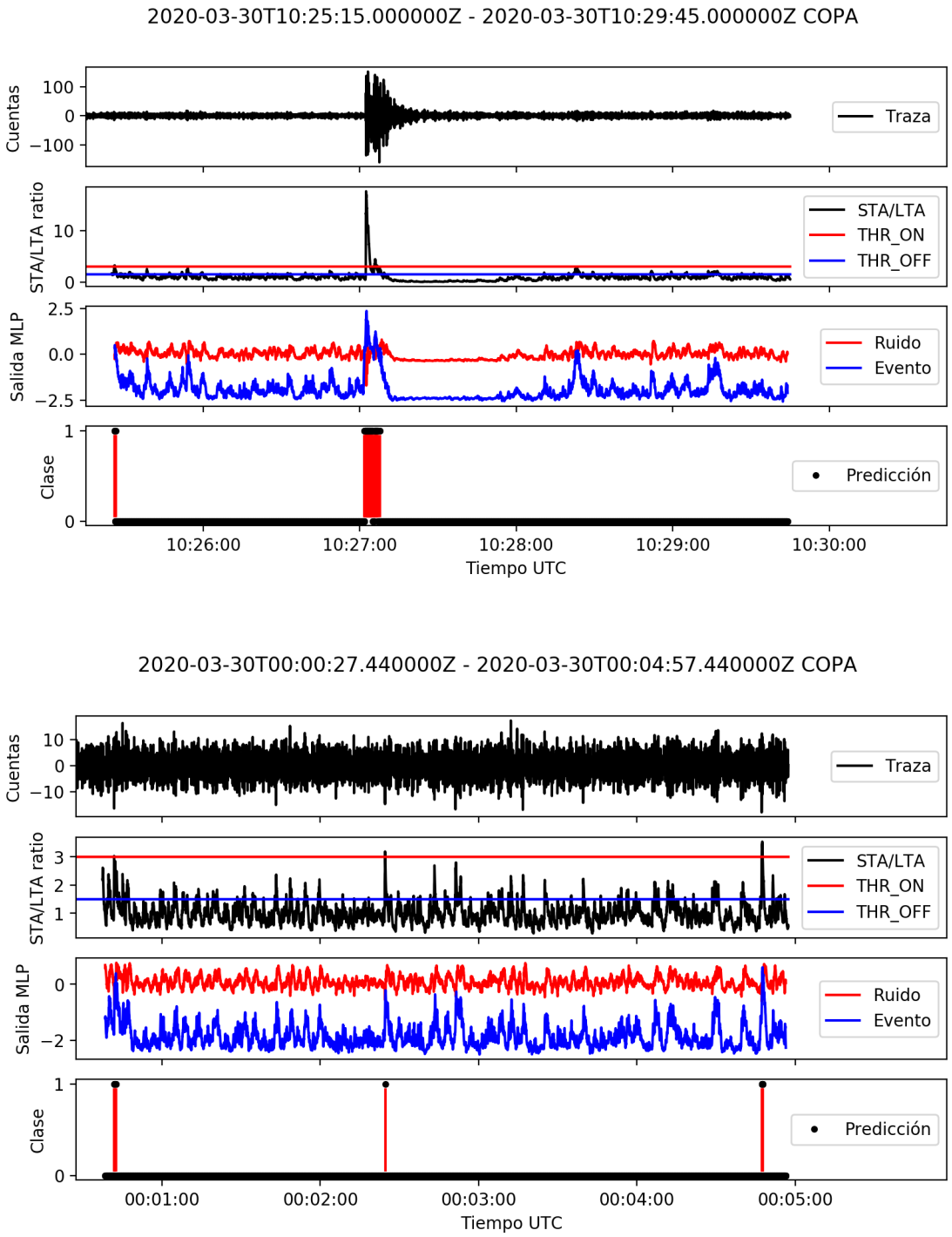


Figura 5.2.12: MLP con función Sigmoide estandarizado y 14 neuronas capa oculta. Primera imagen: Respuesta del modelo a una traza continua de un evento. Segunda imagen: Repuesta para una traza de ruido sísmico.

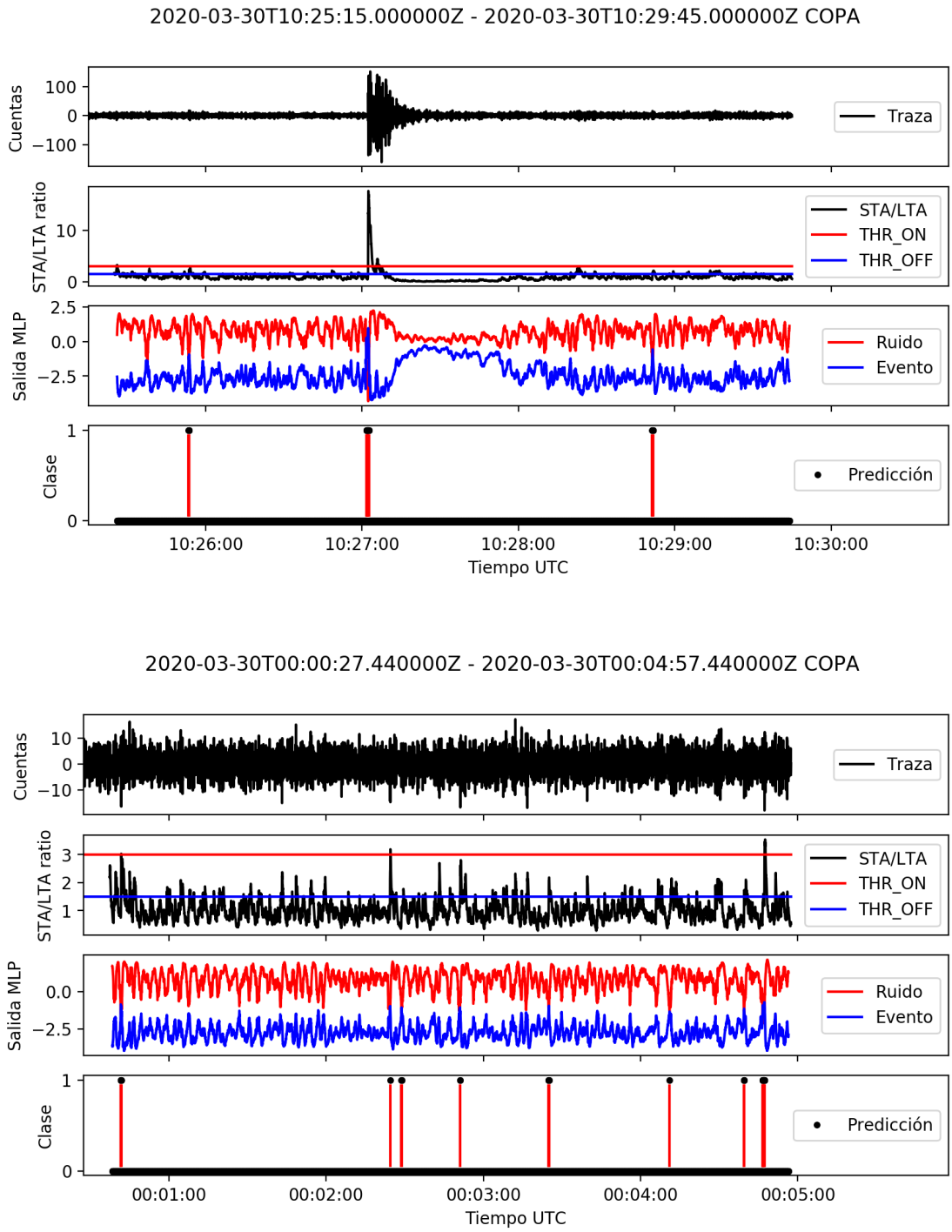


Figura 5.2.13: MLP con función Tanh y 17 neuronas capa oculta.Primera imagen: Respuesta del modelo a una traza continua de un evento. Segunda imagen: Repuesta para una traza de ruido sísmico.

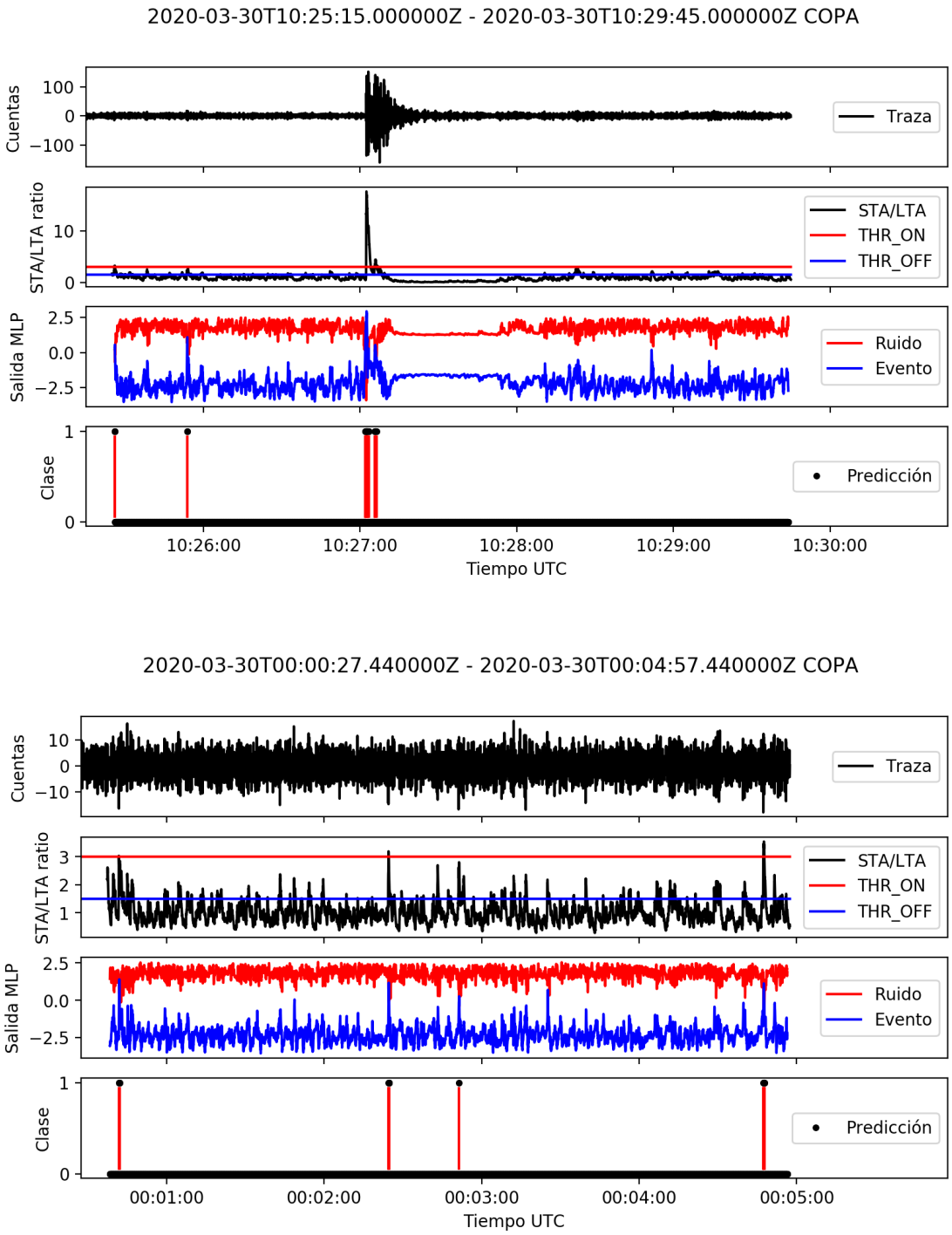


Figura 5.2.14: MLP con función Tanh estandarizado y 15 neuronas capa oculta.Primera imagen: Respuesta del modelo a una traza continua de un evento. Segunda imagen: Repuesta para una traza de ruido sísmico.

A partir de los resultados mostrados en las figuras 5.2.9 a 5.2.14, es posible observar que el modelo que entrega menor cantidad de falsos positivos, corresponde al que utiliza función Sigmoide, con un leve mejor desempeño para el caso no-

estandarizado. Para el resto de modelos, su desempeño es igual o menor a lo esperado solamente utilizando STA/LTA, lo cual no es lo óptimo para este trabajo.

5.2.5. Otros modelos

Frente a los resultados obtenidos de las pruebas y comparación de modelos, se decidió integrar para la última sección tres modelos extra, considerando lo sugerido por las distintas comparaciones. En primer lugar, estos modelos utilizan la función de activación Sigmoide, dado su bajo número de falsos positivos, a su vez, la capa oculta se compondrá por 20 neuronas ocultas, debido a los bajos errores que los altos números de neuronas mostraron anteriormente. Los resultados y estructuras de estos modelos se presentarán a continuación:

- **MLP con 20 neuronas no estandarizado:** A diferencia del modelo obtenido del proceso de selección, este presenta mayor complejidad debido a su mayor número de neuronas en la capa oculta (20 vs 14).
- **MLP con 20 neuronas estandarizado:** Similarmente al anterior, el objetivo es añadir complejidad, a su vez de introducir la estandarización, debido a que la diferencia obtenida en los resultados anteriores no era demasiada en cuanto a falsos positivos.
- **MLP con 2 capas ocultas no estandarizado:** Con el propósito de integrar incluso más complejidad a la arquitectura de la red, es que se introdujo este modelo extra. La primera capa oculta consta de 20 neuronas, mientras que la segunda solo presenta 10 (hay que conservar la forma de embudo de la estructura). No se estandarizan los datos en este caso debido a que solo se quiere ver la influencia de las dos capas ocultas.

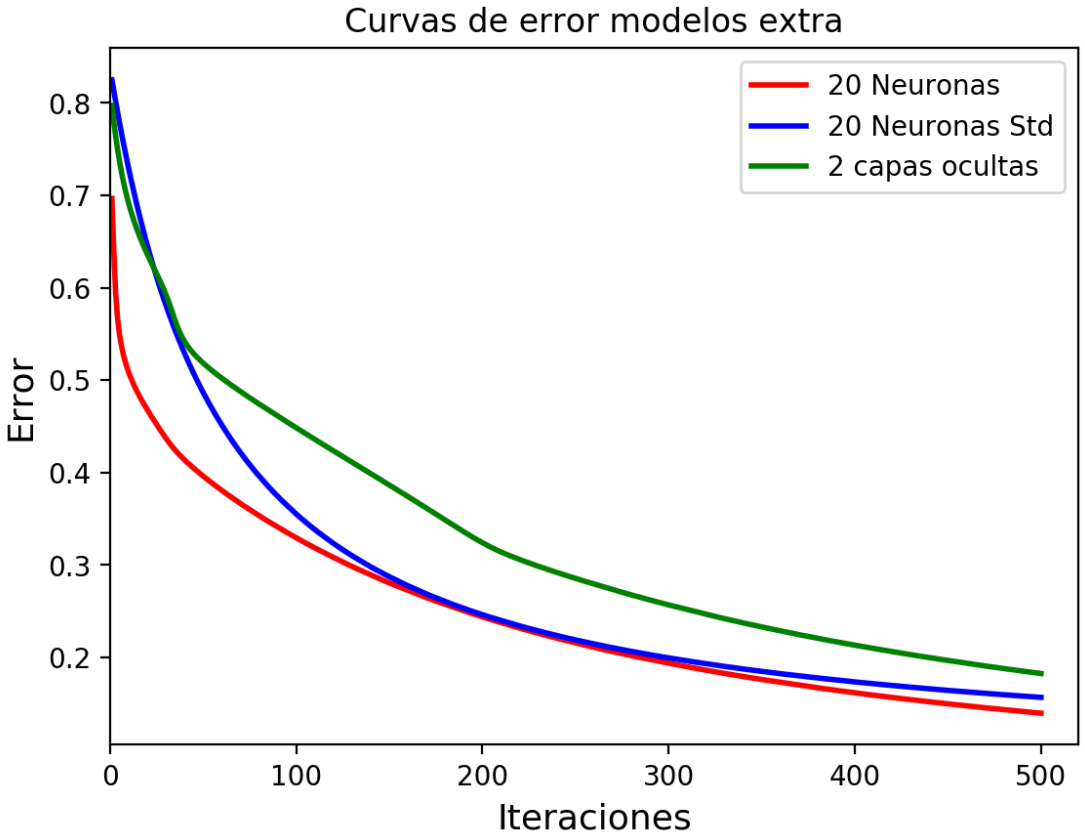


Figura 5.2.15: Curvas de error para los 3 modelos extras.

Modelo	Exactitud	Precisión	Sensibilidad	Error final
20 Neuronas	0.917	1.0	0.853	0.139
20 Neuronas Std	0.90	1.0	0.823	0.157
2 capas ocultas	0.917	1.0	0.853	0.183

Tabla 5.2.5: Resumen de métricas de matrices de confusión y error final, según figura 5.2.15

.

Se observa de la figura 5.2.15 que el modelo con menor error de este set corresponde al de 20 neuronas no estandarizado. En cuanto a las otras métricas asociadas a sus matrices de confusión, obtenidas en la tabla 5.2.5, los tres modelos se presentan de manera muy similar, donde aquel de 20 neuronas estandarizado muestra un leve menor desempeño.

Actual		Predicción: Ruido	Predicción: Evento
	Ruido	26	0
	Evento	5	29

(a) 20 neuronas no estandarizado.

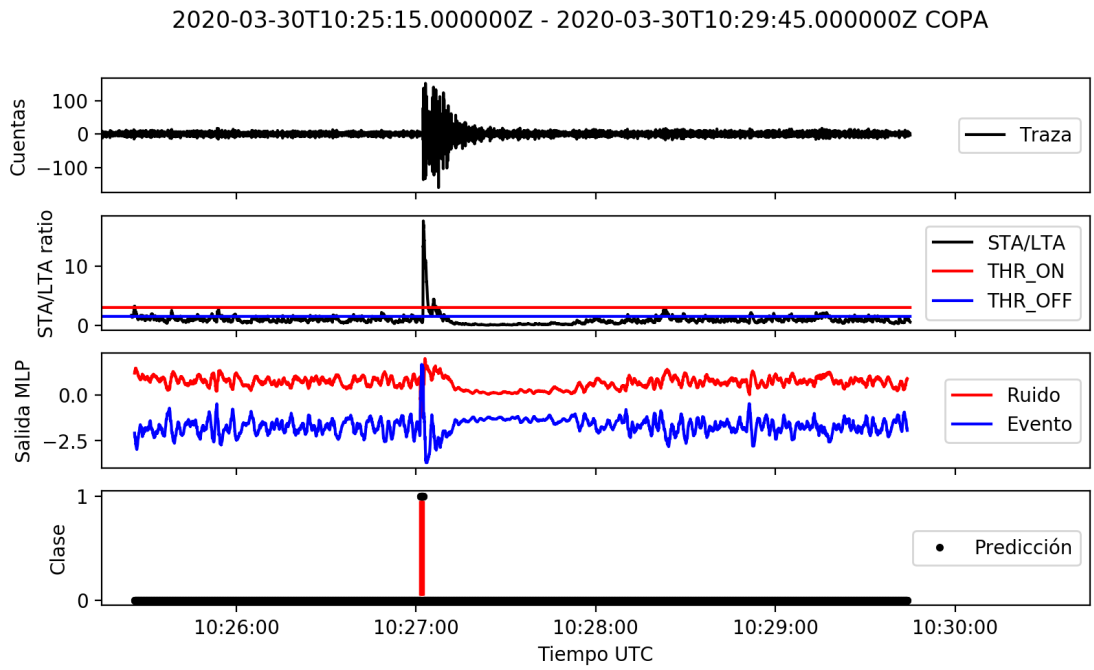
Actual		Predicción: Ruido	Predicción: Evento
	Ruido	26	0
	Evento	6	28

(b) 20 neuronas estandarizado.

Actual		Predicción: Ruido	Predicción: Evento
	Ruido	26	0
	Evento	5	29

(c) 2 capas ocultas 20 y 10 neuronas.

Tabla 5.2.6: Matrices de confusión para los modelos extras basados en función Sigmoide y el set de datos de validación.



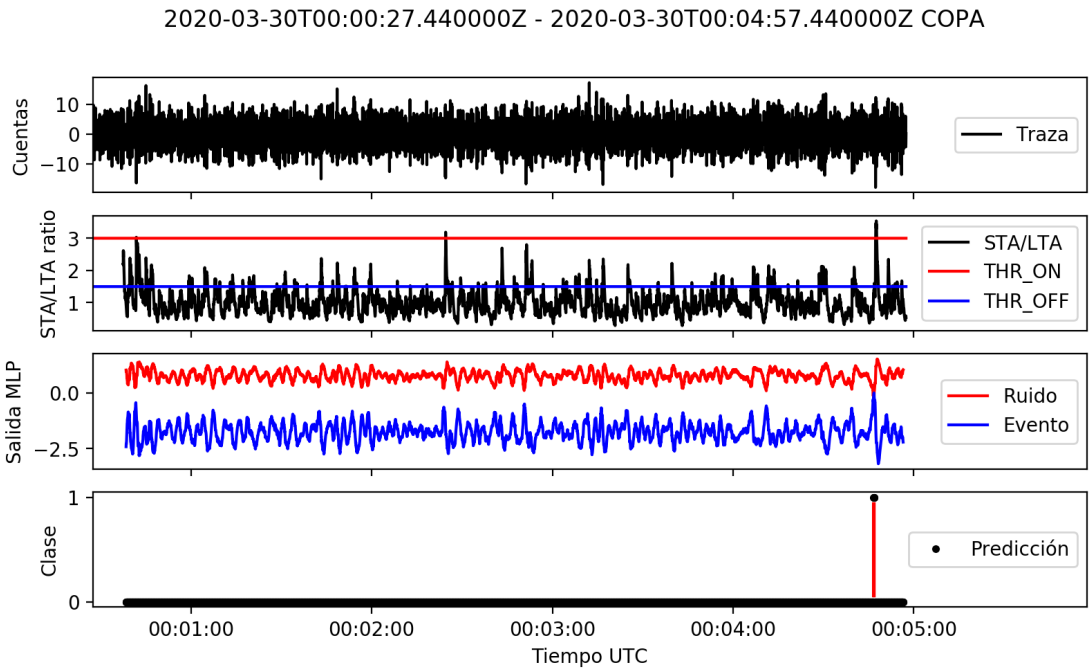
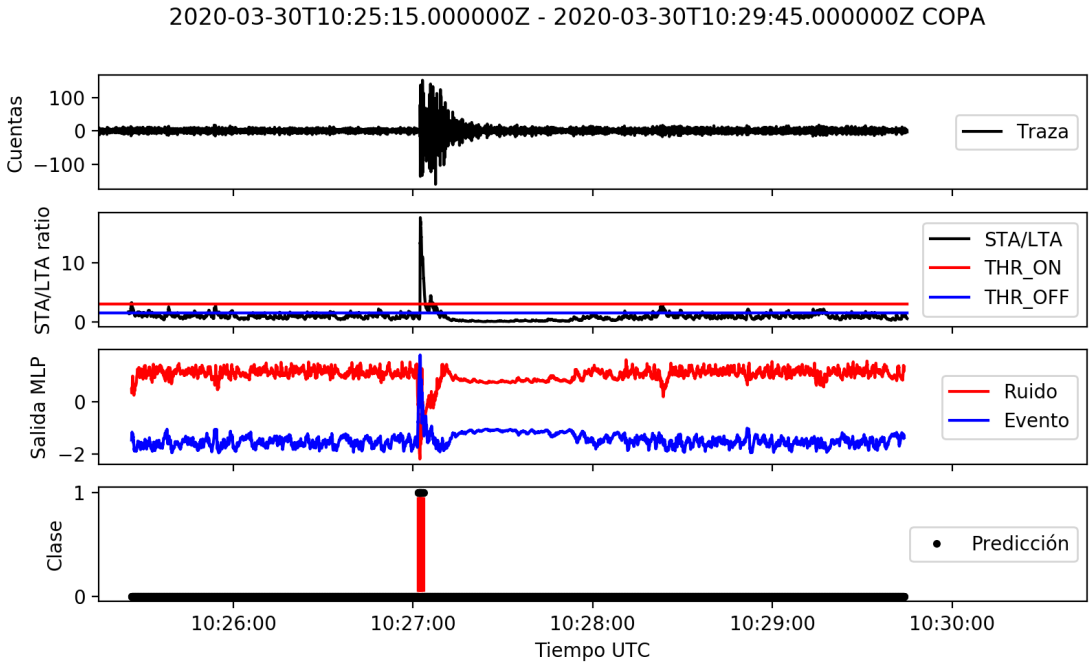


Figura 5.2.16: Modelo extra MLP con función Sigmoide y 20 neuronas en la capa oculta, no estandarizado. Primera imagen: Respuesta del modelo a una traza continua de un evento. Segunda imagen: Repuesta para una traza de ruido sísmico.



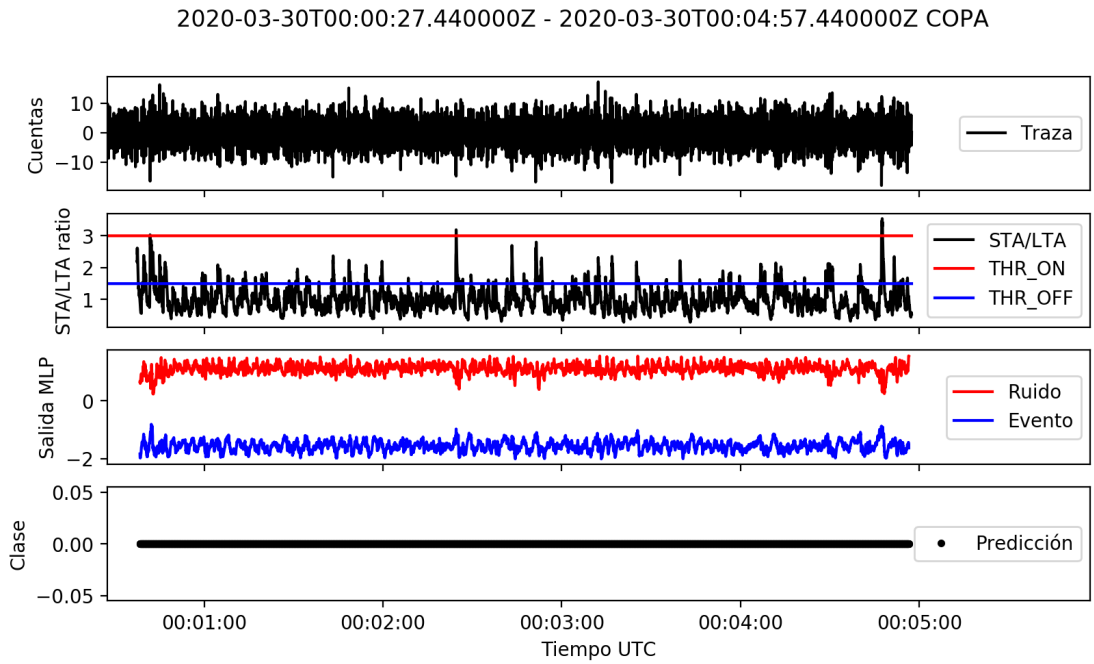
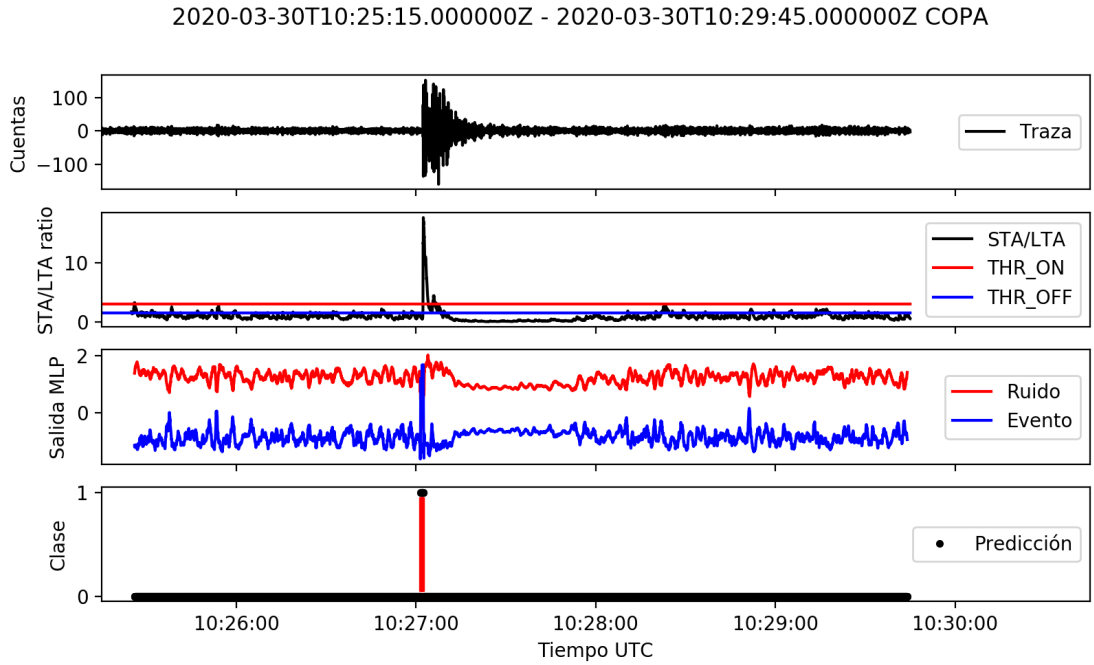


Figura 5.2.17: Modelo extra MLP con función Sigmoide y 20 neuronas en la capa oculta, estandarizado. Primera imagen: Respuesta del modelo a una traza continua de un evento. Segunda imagen: Repuesta para una traza de ruido sísmico.



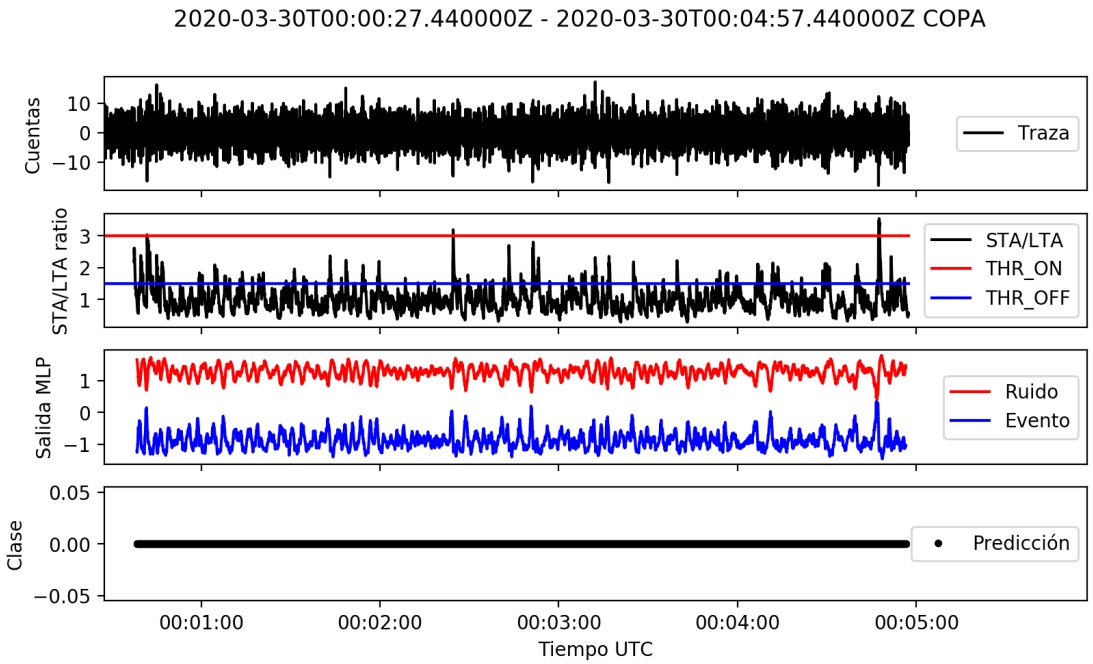


Figura 5.2.18: Modelo extra MLP con función Sigmoide y 2 capas ocultas, con 20 y 10 neuronas en cada capa, no estandarizado. Primera imagen: Respuesta del modelo a una traza continua de un evento. Segunda imagen: Repuesta para una traza de ruido sísmico.

La prueba con trazas continuas (figura 5.2.16 - 5.2.18) para los tres modelos extra resulta excepcional en cuanto a número de falsos positivos respecta. Para el modelo de 20 neuronas no estandarizado, este solo presenta un falso positivo en la sección de ruido, mientras que el evento lo identifica de forma precisa. El mismo modelo pero estandarizado lo hace aún mejor, con una sección de ruido limpia, en donde nos podemos fijar que las respuestas entre las neuronas de salidas se alejan entre si mucho más que en el resto de modelos (Salida MLP). Finalmente, el modelo de dos capas ocultas obtiene similares resultados, pero muestra un comportamiento en las neuronas de salida cercano a arrojar un falso positivo.

5.2.6. Estadística de desempeño

Siguiendo con la evaluación del desempeño en trazas continuas, se puso a prueba los distintos modelos disponibles para el caso de un enjambre sísmico, ocurrido durante el 03/04/2020 a lo largo de la madrugada, cuantificando las detecciones por parte de los modelos MLP y el algoritmo de STA/LTA.

De lo anterior, se obtuvieron los siguientes resultados:

Modelo	MLP	Erróneos	MLP Corregido	Mejora FP
14 Neuronas v1	418	42	376	60.5 %
20 Neuronas	284	15	269	71.7 %
20 Neuronas Std	327	215	112	88.2 %
2 capas ocultas	312	34	278	70.8 %

Tabla 5.2.7: Resumen de detecciones para el enjambre del 03/04/2020, utilizando MLP y comparado al algoritmo STA/LTA. La columna Erróneos corresponde a las detecciones falsas debido a multi-detecciones. MLP Corregido son las detecciones menos las Erróneas. Por último, la columna Mejora FP es la disminución (o mejora) en falsos positivos versus las detecciones de STA/LTA, que equivalen a 952 para el enjambre.

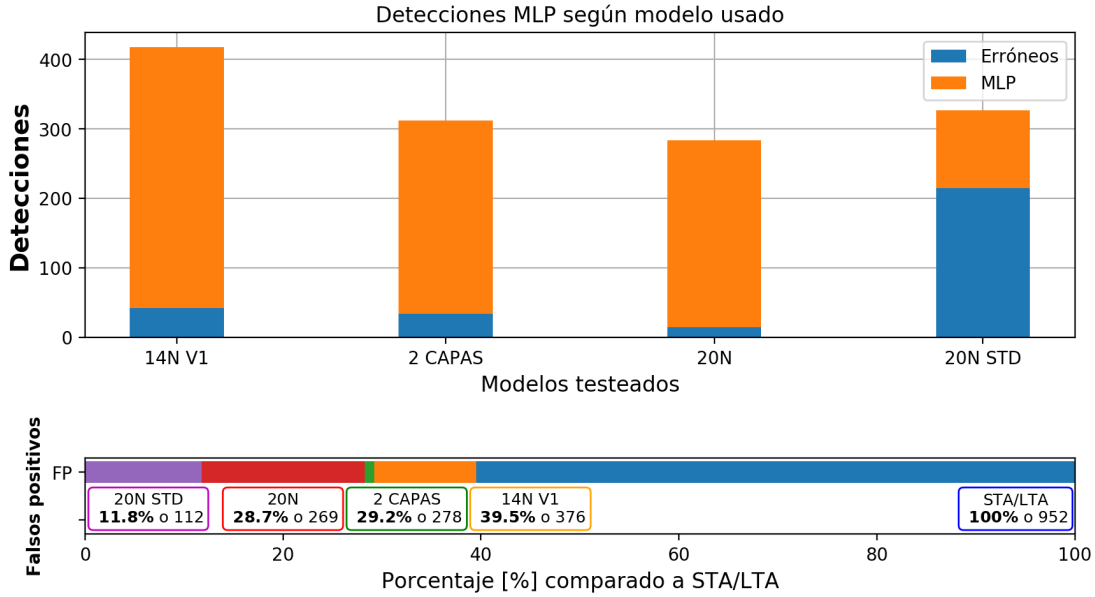


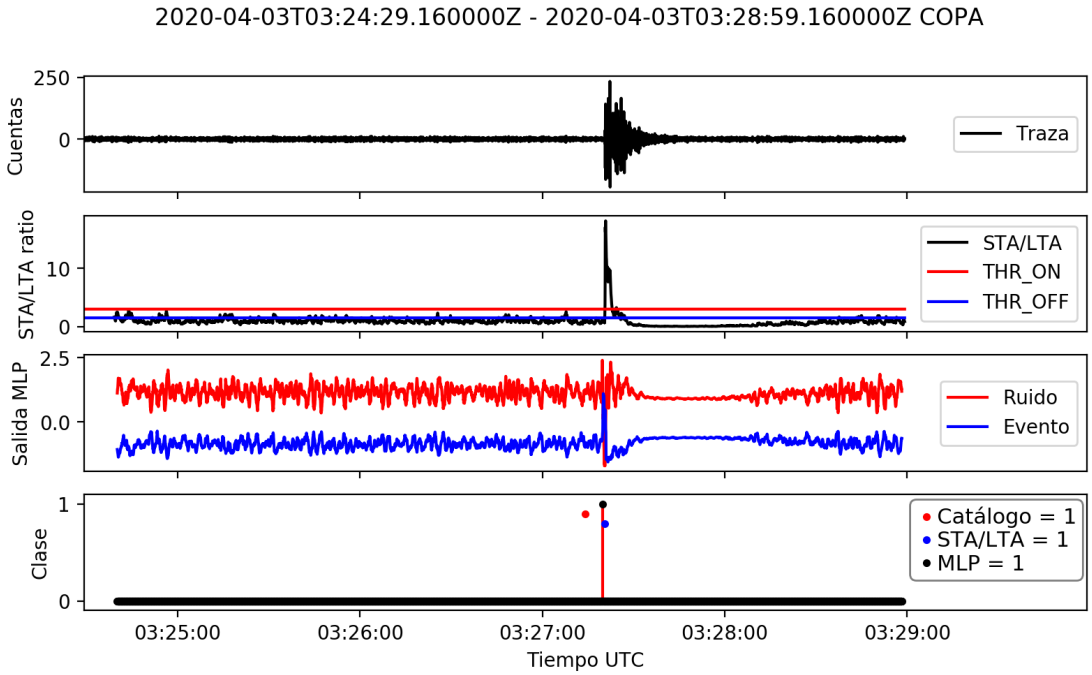
Figura 5.2.19: Resultados de detecciones para el enjambre del 03/04/2020. Superior: Detecciones según los distintos modelos de MLP utilizados, con las multi-detecciones asociadas a cada uno. Inferior: Porcentaje de falsos positivos comparado al total de detecciones por parte del STA/LTA, sin contar erróneos ni eventos catalogados.

Cabe destacar que dentro de la traza correspondiente al día del enjambre, se encuentran catalogados 10 eventos en total, los cuales no se consideran dentro de la contabilidad. En este aspecto, todos los modelos registraron los 10 eventos, demostrando que, a pesar de disminuir la tasa falsos positivos, no se vio afectada la capacidad de detección de estos eventos pre-listados.

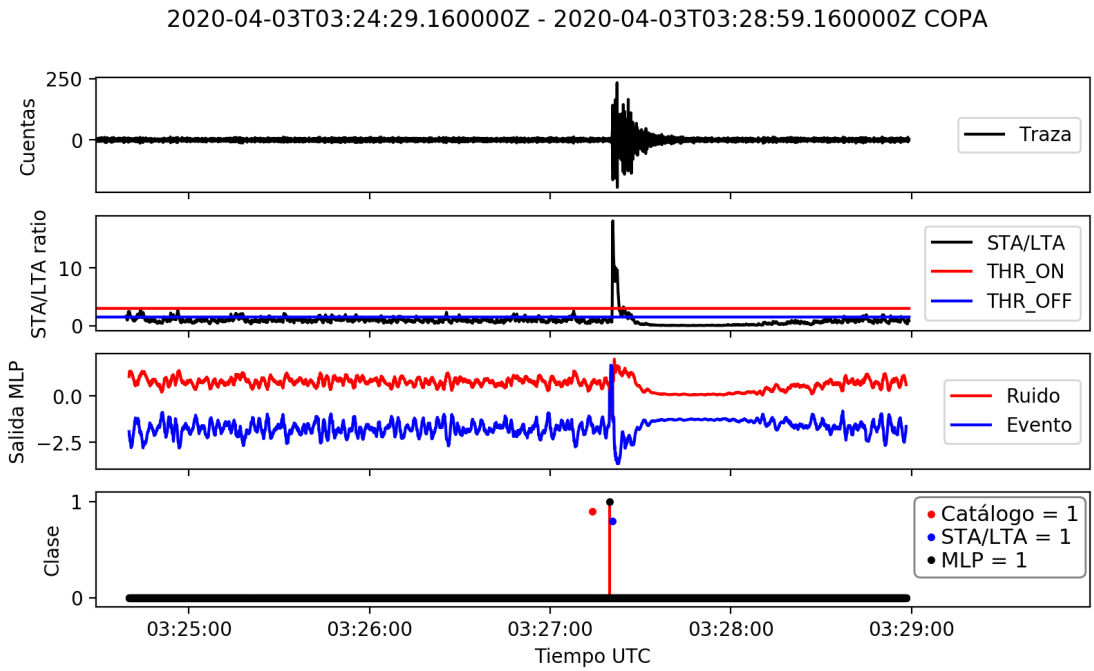
A partir de la tabla 5.2.7 y figura 5.2.19 se observa el desempeño de los 4

modelos, que corresponden al seleccionado en la comparación de modelos y los 3 extras, añadidos en la sub-sección anterior. De estos resultados, se nota que el modelo con menos cantidad de detecciones y falsos positivos, corresponde al modelo de 20 neuronas ocultas estandarizado, presentando una mejora de alrededor del 88.2 %, con la desventaja de mostrar un alto número de detecciones erróneas o repetidas.

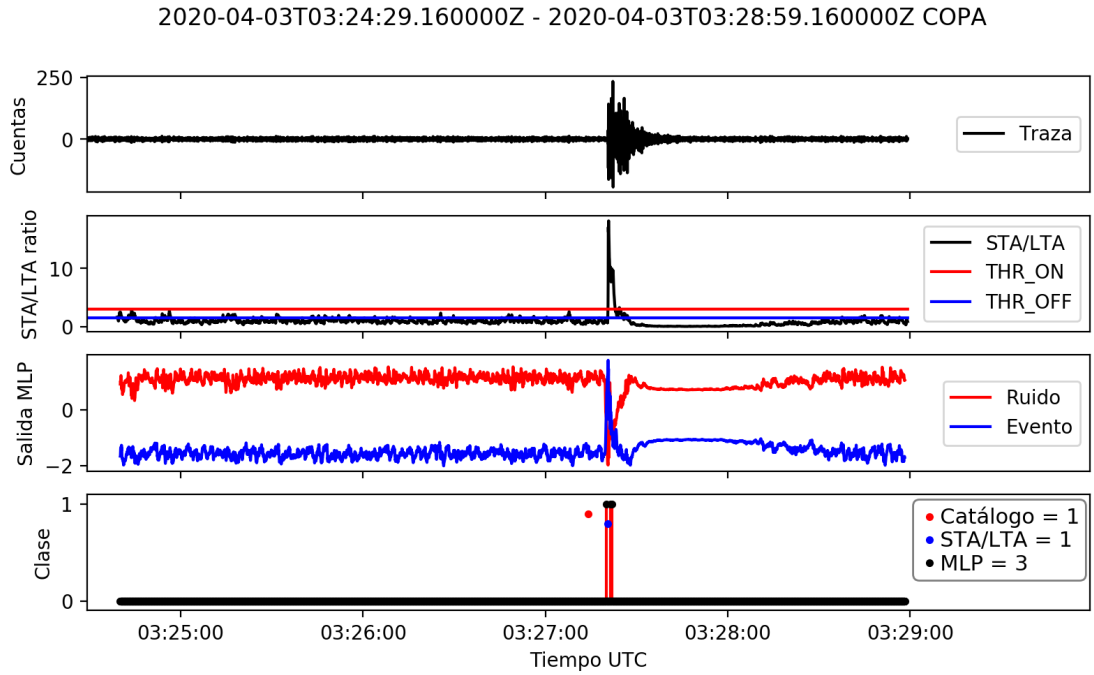
Comparados con el *Triggering* del STA/LTA, todos los modelos exhiben una mejora en el aspecto de falsos positivos desde un 60 a 90 %, manteniendo una precisión excelente de los eventos ya listados. Para mayor comparación entre los modelos y el STA/LTA, se presentan a continuación algunas situaciones de ejemplo dentro de la secuencia del enjambre sísmico:



a) Registro de un evento para la secuencia del enjambre, utilizando el modelo con Sigmoide 14 Neuronas Versión 1 no estandarizado, obtenido a partir de la selección de modelos.



b) Registro de un evento para la secuencia del enjambre, utilizando el modelo con Sigmoide 20 Neuronas no estandarizado, obtenido de los modelos extras.



c) Registro de un evento para la secuencia del enjambre, utilizando el modelo con Sigmoide 20 Neuronas estandarizado, obtenido de los modelos extras.

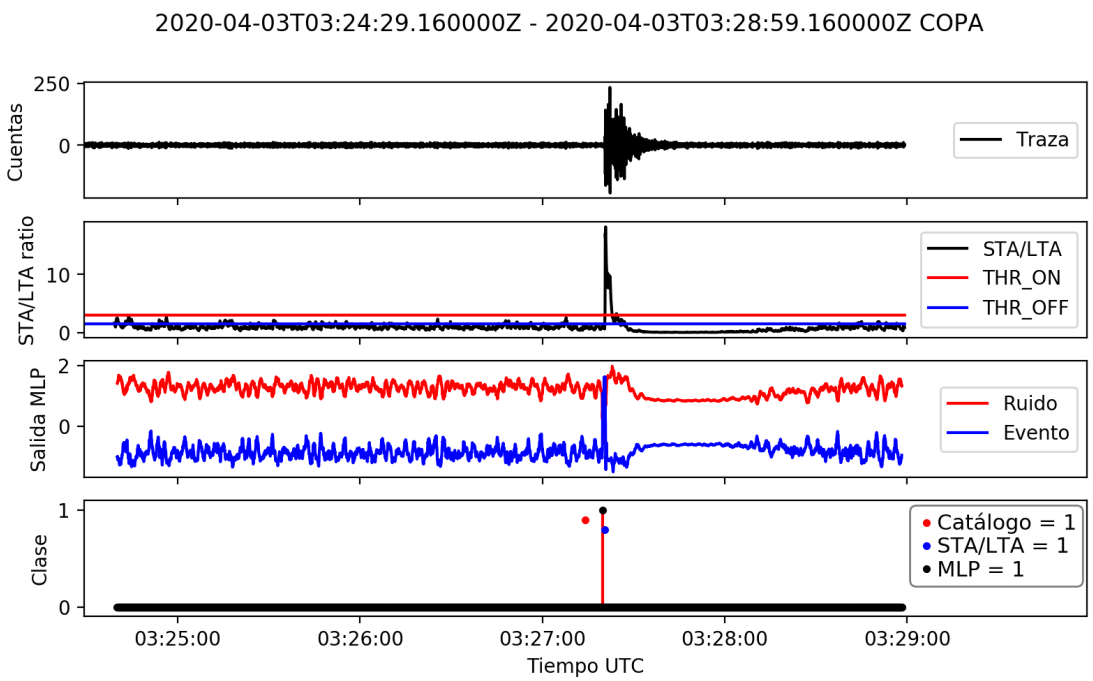
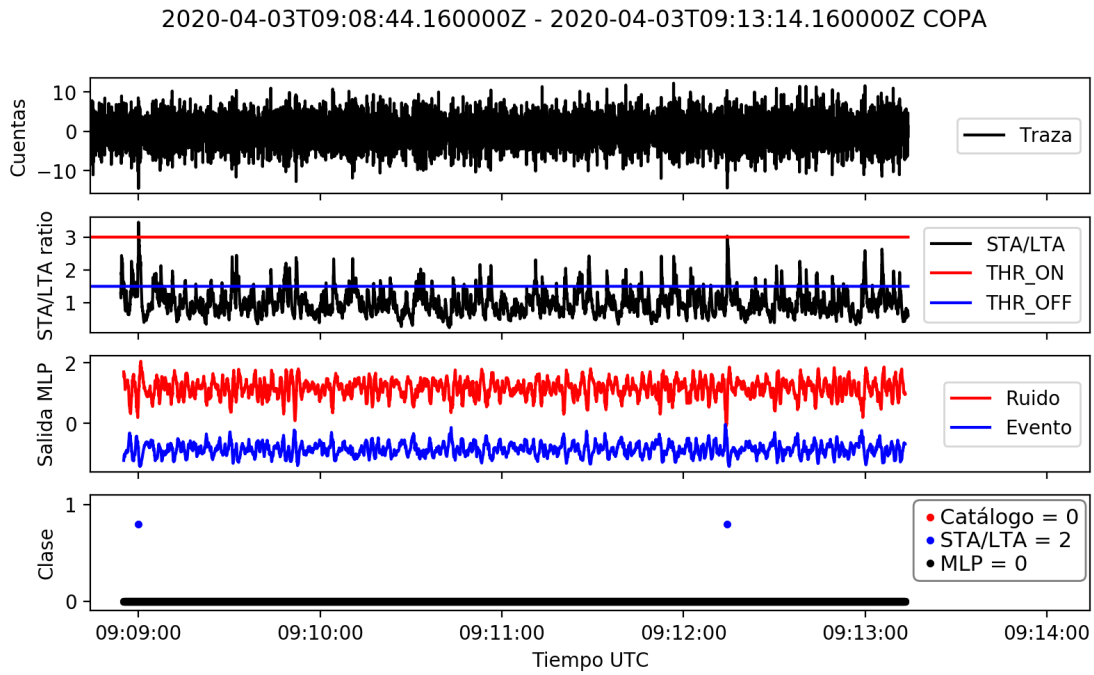
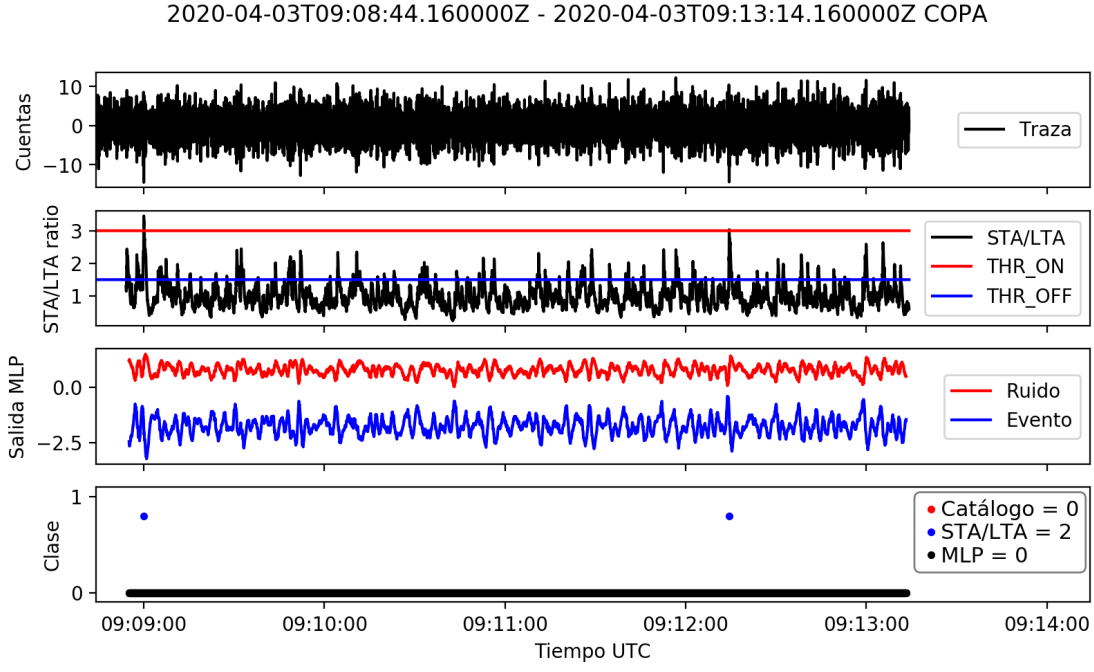


Figura 5.2.20: d) Registro de un evento para la secuencia del enjambre, utilizando el modelo con Sigmoide de 2 capas ocultas (20 y 10 neuronas) no estandarizado, obtenido de los modelos extras. Las imágenes siguen la misma estructura especificada en 5.2.4, con la diferencia que la sección inferior muestra y contabiliza los registros de catálogo en la traza, así como las detecciones MLP y STA/LTA.

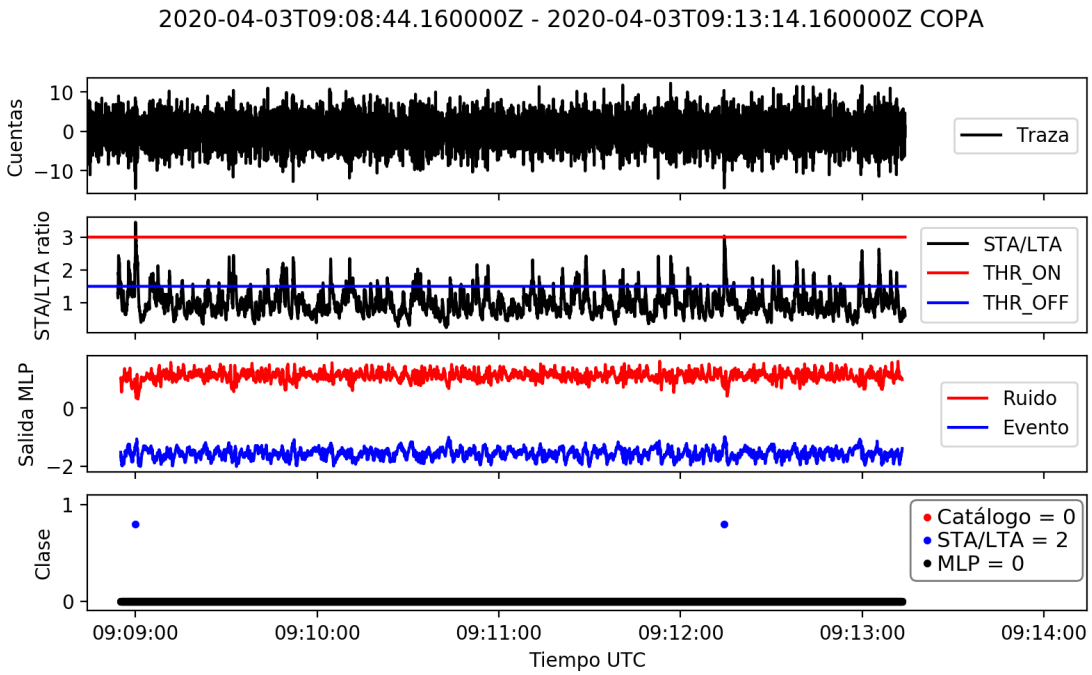
Para el caso del registro del evento, los distintos modelos lo identifican y clasifican correctamente, la diferencia aparece en la respuesta de las neuronas de salida (ver 5.2.20), pues para el modelo de 20 neuronas estandarizado, las neuronas marcan una salida muy clara para el ruido y el evento, lo cual no se repite para los demás modelos, que aún identificando correctamente, sus salidas presentan un poco más de ruido.



a) Registro de ruido sísmico para la secuencia del enjambre, utilizando el modelo con Sigmoide 14 Neuronas Versión 1 no estandarizado, obtenido a partir de la selección de modelos.



b) Registro de ruido sísmico para la secuencia del enjambre, utilizando el modelo con Sigmoide 20 Neuronas no estandarizado, obtenido de los modelos extras.



c) Registro de ruido sísmico para la secuencia del enjambre, utilizando el modelo con Sigmoide 20 Neuronas estandarizado, obtenido de los modelos extras.

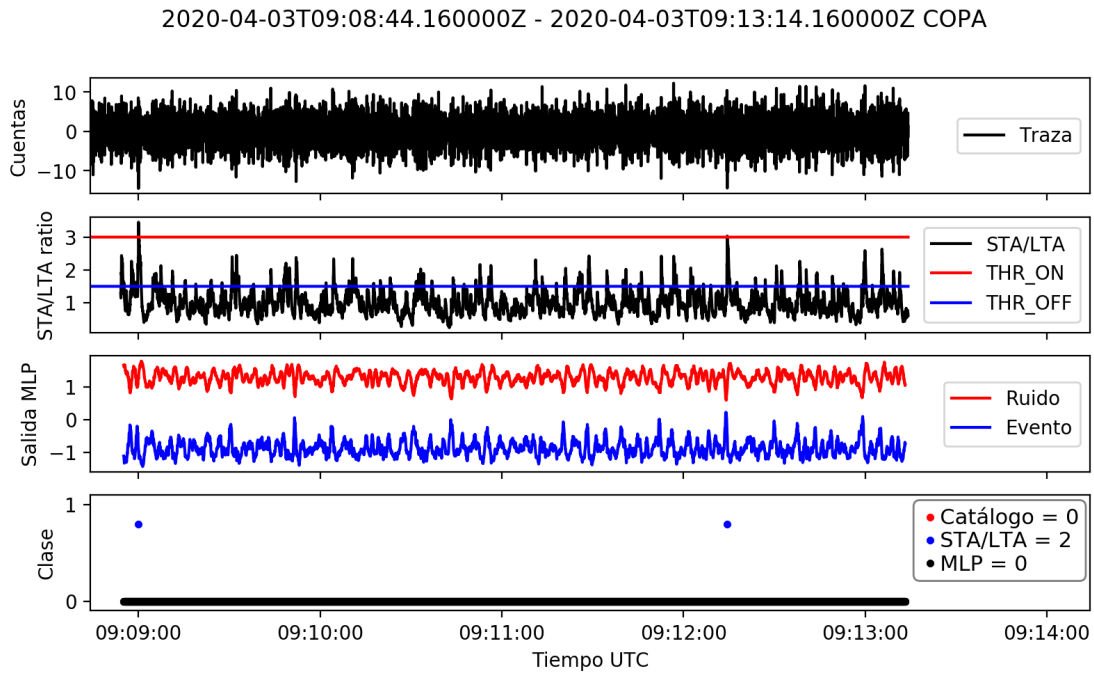
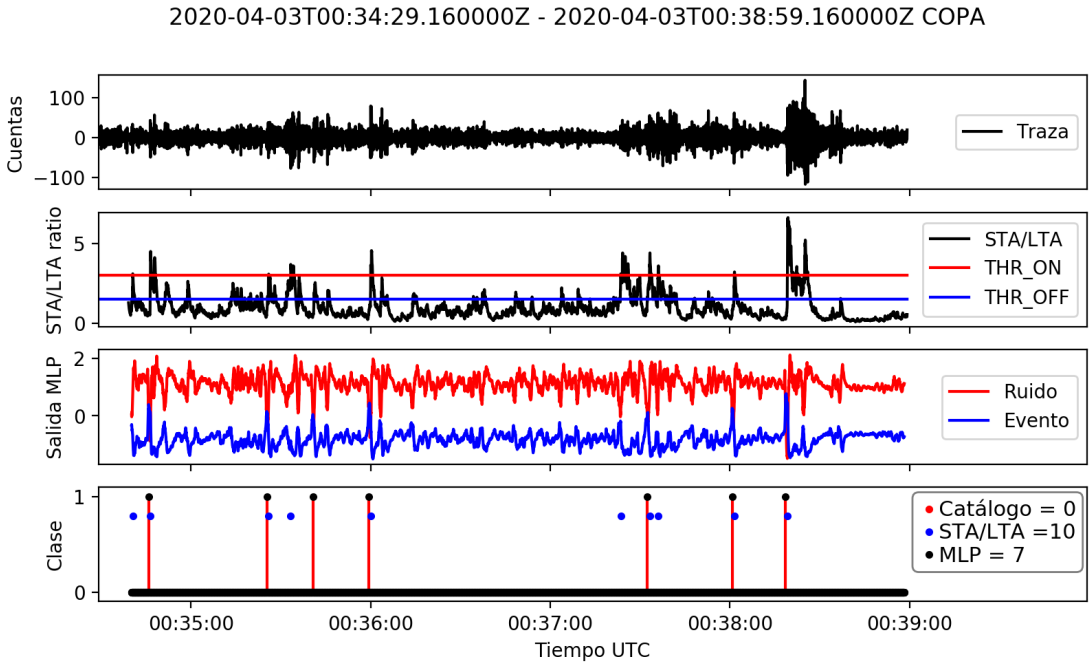
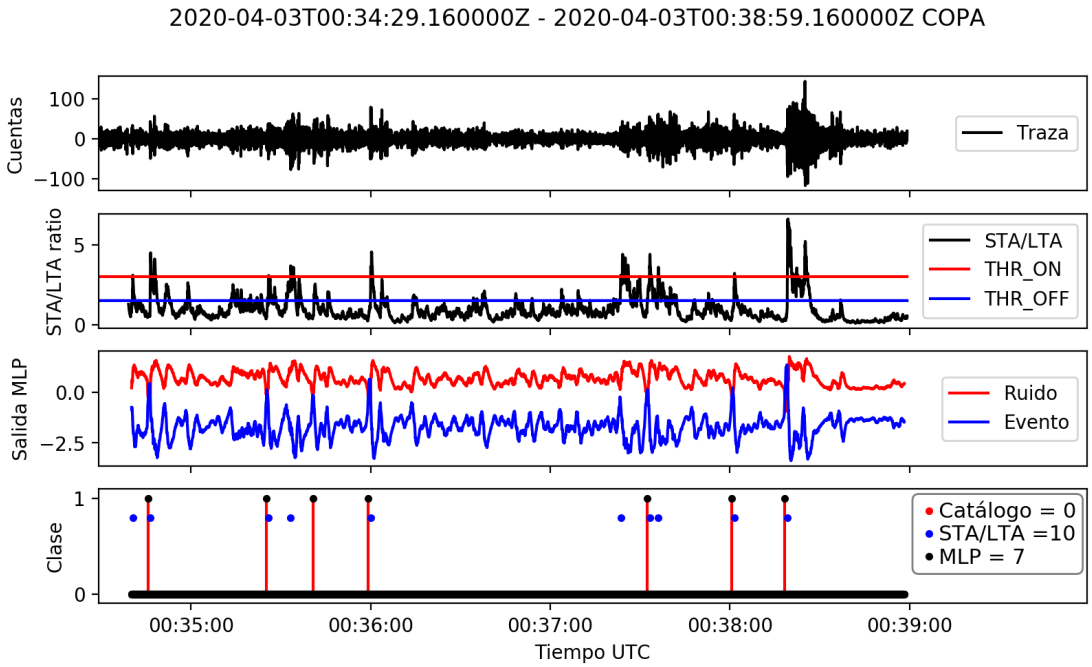


Figura 5.2.21: d) Registro de ruido sísmico para la secuencia del enjambre, utilizando el modelo con Sigmoide de 2 capas ocultas (20 y 10 neuronas) no estandarizado, obtenido de los modelos extras. Las imágenes siguen la misma estructura especificada en 5.2.4, con la diferencia que la sección inferior muestra y contabiliza los registros de catálogo en la traza, así como las detecciones MLP y STA/LTA.

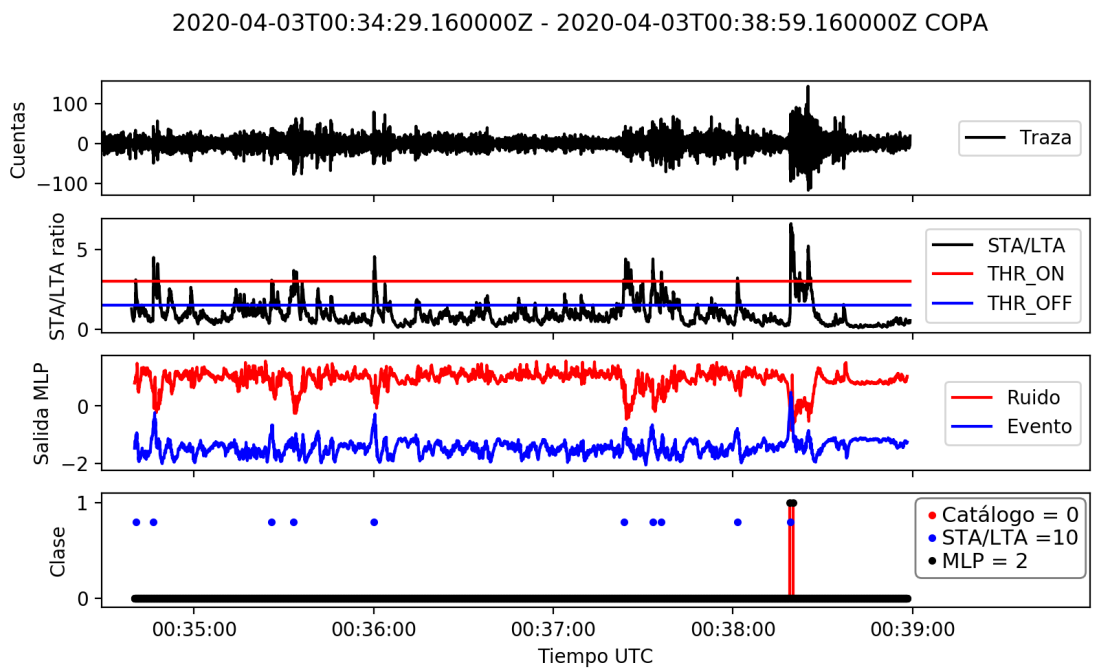
Similarmente a lo visto en la figura 5.2.20, en la figura 5.2.21 es posible observar que todos los modelos tienen éxito en clasificar el ruido sísmico tal como es, mientras el STA/LTA presenta dos marcas que no corresponden a eventos. Además, nuevamente el modelo que presenta mayor diferencia en las respuestas de las neuronas de salida, es el modelo extra de 20 neuronas estandarizado.



a) Registro de ruido sísmico con alto número de detecciones para la secuencia del enjambre, utilizando el modelo con Sigmoide 14 Neuronas Versión 1 no estandarizado, obtenido a partir de la selección de modelos.



b) Registro de ruido sísmico con alto número de detecciones para la secuencia del enjambre, utilizando el modelo con Sigmoide 20 Neuronas no estandarizado, obtenido de los modelos extras.



c) Registro de ruido sísmico con alto número de detecciones para la secuencia del enjambre, utilizando el modelo con Sigmoide 20 Neuronas estandarizado, obtenido de los modelos extras.

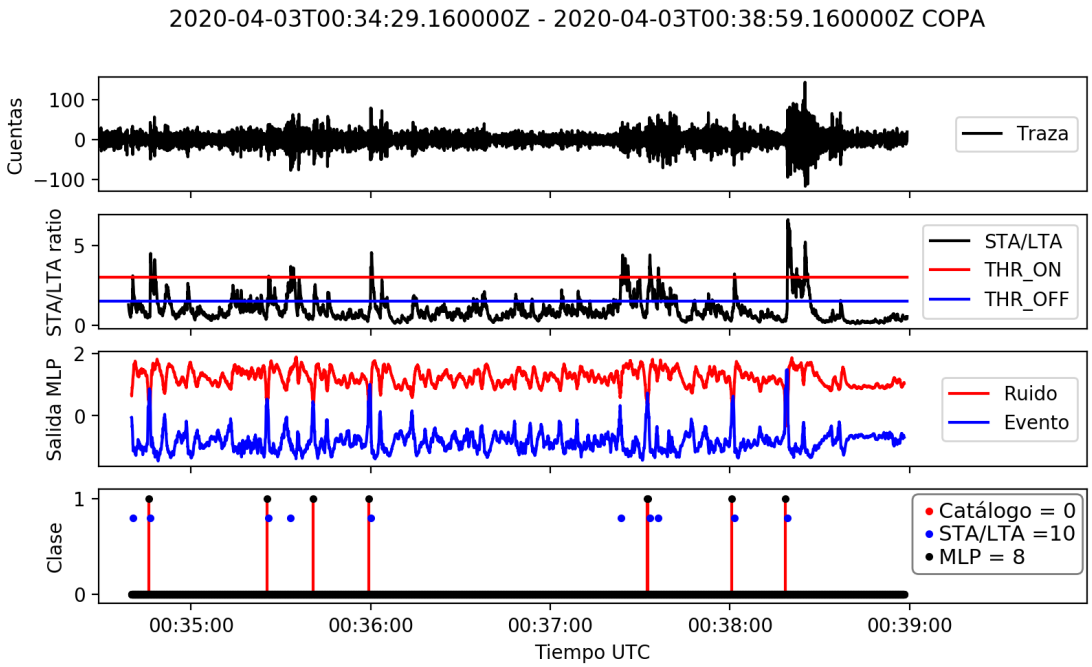


Figura 5.2.22: d) Registro de ruido sísmico con alto número de detecciones para la secuencia del enjambre, utilizando el modelo con Sigmoide de 2 capas ocultas (20 y 10 neuronas) no estandarizado, obtenido de los modelos extras. Las imágenes siguen la misma estructura especificada en 5.2.4, con la diferencia que la sección inferior muestra y contabiliza los registros de catálogo en la traza, así como las detecciones MLP y STA/LTA.

Por último, en la figura 5.2.22 vemos una sección de la traza durante el enjambre sísmico, con una variedad de detecciones utilizando STA/LTA. Para este caso, los modelos de MLP presentaron una variedad de detecciones y falsos positivos, aunque en menor medida comparado a lo conseguido con STA/LTA. Así, todos los modelos se comportan de manera similar, a excepción de aquel de 20 neuronas estandarizado, que arroja solo una detección de entre las 10 cuantificadas por STA/LTA, considerando que se presenta uno repetido.

De esta forma, sumado a lo visto en la tabla 5.2.7 y figura 5.2.19, se considera el modelo de 20 neuronas con función Sigmoide y estandarización como aquel que mejor desempeño presenta en la práctica.

6. Discusión

La premisa de este estudio es probar la posibilidad de detectar, de forma automática y óptima, eventos sísmicos dentro de los sismogramas, registrados por la estación COPA (Volcán Copahue) del Centro de Monitoreo Volcánico de la Universidad de Concepción, mediante el uso de redes neuronales artificiales. Variados artículos e investigaciones muestran la implementación de distintas redes neuronales u otros elementos de *Machine-learning*, con resultados prometedores en el ámbito de la Sismología, en aspectos de clasificación de señales ((Ibs-von Seht, 2008),(Del Pezzo et al., 2003)(Curilem et al., 2009)), detección de eventos ((Wang and Teng, 1995), (Tiira, 1999)), *picking* de fases ((Dai and MacBeth, 1997),(Zhao and Takano, 1999)), entre otros, sugiriendo que la introducción de alguna forma de inteligencia artificial o reconocimiento de patrones en métodos de detección de eventos, puede mejorar significativamente el desempeño de estos mismos.

Los resultados de esta tesis respecto al punto anterior, muestran en general una gran mejora del desempeño de los modelos finales comparado al método de STA/LTA, disminuyendo la cantidad de detecciones, y por lo tanto, los falsos positivos, y a su vez manteniendo la capacidad de detección de eventos ya listados en el catálogo proporcionado por el Centro de Monitoreo Volcánico, lo cual nos dice implícitamente que, en general, no hay gran presencia de falsos negativos, también observable en las medidas de sensibilidad. Esto es congruente con otros trabajos similares, en donde las más grandes mejoras se encuentran entre la disminución de falsos positivos e incremento de la capacidad de detección (Tiira, 1999), siendo este último aspecto no observable en esta tesis, pero que podría ser abordado en el futuro con un análisis más exhaustivo de los registros y la comparación con el método impulsivo de STA/LTA.

Como se especificó a lo largo de este trabajo, la red utilizada fue una de tipo MLP, la cual fue optimizada analíticamente mediante la discriminación de modelos según sus métricas de desempeño y error, según factores como las

neuronas ocultas o las funciones de activación utilizadas. Respecto a lo anterior, hay algunos puntos que pueden ser discutidos. En primer lugar, se observa que la red MLP es comúnmente utilizada para este tipo de propósitos, tal como lo prueban ciertos autores ((Wang and Teng, 1995), (Tiira, 1999), (Curilem et al., 2009)) , pero también se encuentran artículos que introducen otras metodologías asociadas a *Machine-learning* que podrían ser exploradas, por ejemplo, ((Ibáñez et al., 2009), (Kortström et al., 2016)). En este aspecto, también se podría evaluar la introducción de algoritmos para la optimización automática de los parámetros de la red, tal como ha sido realizado en Curilem et al. (2009).

Siguiendo en la línea anterior, se observa que las métricas usadas para obtener los mejores modelos no entregaron los resultados esperados, donde aquellos seleccionados por este método fueron descartados, o bien igualados o sobrepasados por los modelos extras en pasos posteriores, lo que indica que el error como única métrica no sería realmente un buen discriminante. En contraste, el análisis práctico, estadístico y visual realizado en cuanto a las detecciones y falsos positivos, comparando con los *Triggers* de STA/LTA, resultó excelente a la hora de elegir los mejores modelos, lo que nos mostraría que es mejor poner a prueba la red en trazas continuas, a secciones recortadas para cada clase, mientras que el error como medida debe ser complementado con otras métricas o métodos de selección.

En otras investigaciones se utilizan distintos y variados sets de patrones de entrada, lo cual se explica debido a que la definición de buenos patrones incide directamente en el éxito de la red neuronal. Bajo esta regla, para posibles mejoras de la red podrían incluirse nuevos patrones además del STA/LTA utilizado, como información espectral y/o estadística, por ejemplo ((Wang and Teng, 1997), (Wiszniowski et al., 2014)). Dentro de este aspecto, la utilización de 50 puntos de STA/LTA alrededor del *trigger* mostró buenos resultados, al representar de buena forma la característica impulsividad de una señal de evento, así, posiblemente la extensión de esta ventana de tiempo añadiría información redundante para esta característica. Por otro lado, con respecto a la información triaxial de la estación COPA, no es sabida su influencia en términos de identificación de sismos, pero si se ha mostrado útil para redes de *picking* de fases. Debido a lo anterior, no sería recomendable su utilización, considerando el mayor volumen de datos que

introduciría esto a la red, a no ser que se pretenda la identificación de fases sísmicas.

Así, con la inclusión de mayor información discriminante de las trazas, sería posible ampliar la función del MLP como detector a *picking* de fases, como los autores mencionados, entre otros, han establecido, y cuyos trabajos han resultado exitosos en dicho objetivo ((Dai and MacBeth, 1997), (Zhao and Takano, 1999)), con la desventaja de posibles mayores costos computacionales y de tiempo de procesamiento, al incrementar el volumen de datos y, en consecuencia, la complejidad de la red.

Un aspecto importante es la posibilidad de mejorar la estructura y funcionamiento interno de la red MLP utilizada. Los resultados muestran que añadir más capas ocultas no mejora significativamente el desempeño de los modelos, mientras que también ha sido demostrado que la cantidad de neuronas ocultas debe ser alta en lo posible, debido al factor de complejidad de la red. Por otro lado, un aspecto no explorado en esta tesis es la utilización de una neurona de salida única, lo cual podría hacer más manejable a la red y sus productos, debido a la respuesta única que entregaría la red, pudiendo abordarla mediante probabilidad o valores umbrales para la identificación de los eventos. En el aspecto interno del MLP, los valores de las funciones Sigmoide en cuanto a error respecta, se mostraron bastante altos en comparación a las otras funciones, lo cual puede ser debido a que sus curvas de error aún no alcanzaban sus mínimos absolutos. Este aspecto podría ser solucionado incorporando *Early Stopping* [1] (Géron, 2017) en reemplazo a una cantidad de iteraciones finita, lo cual aseguraría valores mínimos de error, pero supondría un número de iteraciones irregular entre modelos, lo que complicaría comparaciones múltiples, como fue hecho en este trabajo.

Como se observó en los resultados, no se logró encontrar una gran diferencia entre el desempeño de modelos con o sin estandarización de los datos. Si bien no se ha encontrado una explicación a tan poca incidencia de este proceso, se puede inferir que esto proviene de las características de los datos de entrada, dado que los 50 valores de cada muestra no presentan mucha diferencia entre

[1] Es un método de regularización de algoritmos de aprendizaje como *Gradient Descent*, cuyo objetivo es detener el proceso de entrenamiento en cuanto se alcanza el mínimo valor de error en el set de validación.

sí, lo cual no es el caso en la mayoría de estudios de redes neuronales, en que los patrones usados son una mezcla de parámetros que poseen rangos muy variables ((Curilem et al., 2009), (Ibs-von Seht, 2008)). Aún así, este tipo de pre-procesamiento es recomendable, pues el mejor modelo obtenido corresponde a uno con estandarización de sus datos, en donde los valores de sus neuronas de salidas eran muy distintivos.

Con respecto al mejor modelo obtenido, correspondiente al de 20 neuronas con función Sigmoide y estandarizado, se ha mostrado que sus resultados son muy buenos puestos a prueba en una situación de análisis continuo, pero también se encontró la desventaja de su alta tasa de detecciones erróneas o múltiples, lo cual fue solventado para los otros modelos mediante un pequeño trabajo de post-procesamiento, disminuyendo sustancialmente este fenómeno. Sin embargo, al parecer este modelo escapa a dicho proceso, por lo que sería necesario integrar un método más robusto para su eliminación, de ser posible, en un futuro trabajo.

Basado en lo discutido durante esta sección, existen muchas modificaciones o elementos que podrían integrarse a una posible futura investigación con *Machine-learning* aplicado a Sismología. Por ejemplo, es posible ampliar la red a otras estaciones para la detección simultánea de eventos, con una posterior catalogación/localización de eventos registrados por la misma, como también se podría trabajar en profundizar la red a funciones de *picking* de fases o clasificación, teniendo la densidad de datos suficiente para ello, o simplemente variar elementos ya usados en este trabajo y ver si son posibles mayores mejoras.

Entonces, dado lo mostrado a lo largo de este trabajo, la utilización de redes MLP se mostraría beneficiosa para redes en general, debido a la disminución de falsas detecciones, fácil implementación y mantención, entre otros factores mostrados. Ahora, para redes de mayor envergadura, con una densidad de datos entrantes mayor, se recomendaría la utilización de redes MLP modificado para entrenamiento en línea, o sea, una red que pueda entrenarse a sí misma de forma periódica, mejorando su desempeño a medida que llegan nuevos datos, permitiendo su actualización automática, además de solucionar problemas de tiempo en la detección de eventos y mantención de la red, claro que, sería necesario tener en consideración su costo computacional, el cual debería ser

reservado en parte para la red.

Dicho lo anterior, las redes neuronales, y en consecuencia *Machine-learning*, muestran un mundo amplio para explorar, con muchos elementos y metodologías aplicables a estudios sismológicos. El uso de este tipo de inteligencia artificial tiene el potencial de ser un gran apoyo para el área de Sismología, ciencias de la Tierra u otras ramas de la ciencia, tal como los resultados de esta tesis y otros autores a lo largo del tiempo lo comprueban, y así, posiblemente en un futuro cercano, estas metodologías se vuelvan más ampliamente reconocidas y utilizadas (Kong et al., 2019).

7. Conclusión

Mediante la utilización de *Machine-learning*, se logró desarrollar una red neuronal artificial que puede detectar eventos sísmicos a partir de los sismogramas, basándose en datos de STA/LTA obtenidos de los registros sismológicos.

El uso propuesto de una red MLP probó ser efectivo para la tarea de detección/clasificación de señales sísmicas, con modelos en su mayoría competentes frente a la metodología del STA/LTA, mientras que otros inclusive mostraron resultados que sobrepasaban el rendimiento de dicho método, alcanzando hasta un 11.8 % de sus detecciones y falsos positivos. Además, la red MLP resultó fácil de manejar, aprender y aplicar, demostrando ser una herramienta poderosa y versátil para este trabajo, y otras aplicaciones en el futuro.

Si bien se logró obtener modelos exitosos y eficientes, la metodología utilizada para su discriminación se mostró insuficiente en algunos aspectos, como la comparación y selección de modelos en base a error y exactitud. Por otra parte, el análisis de falsos positivos y estadístico fue más directo y práctico, dando mejores resultados.

Por otro lado, a pesar que la arquitectura utilizada no presentó inconveniente alguno, esta muestra varias posibilidades a explorar en el futuro, como los patrones a utilizar, las capas ocultas, neuronas de salida y la inclusión de elementos como *Early Stopping* y/o post-procesamiento, para dar un ejemplo. A su vez, la estandarización de datos presentó resultados poco concluyentes, pero al observar que el mejor modelo obtenido es estandarizado, se cree este aspecto recomendable.

Finalmente, dado el éxito obtenido en la implementación del MLP para la estación COPA, se abre la posibilidad tanto para su desarrollo en múltiples estaciones, como para la ampliación de sus funciones a *picking* de fases o detección en tiempo real. Estos aspectos y otros mostrados a lo largo de esta investigación, dan cuenta del amplio mundo a explorar de *Machine-learning* y el potencial de

Redes neuronales. Así, un trabajo en conjunto de esta rama de la Inteligencia artificial podría ser un gran apoyo para la detección de sismos y la Sismología en general.

Bibliografía

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., et al. (2015). Tensorflow: Large-scale machine learning on heterogeneous systems.

Beyreuther, M., Barsch, R., Krischer, L., Megies, T., Behr, Y., and Wassermann, J. (2010). Obspy: A python toolbox for seismology. *Seismological Research Letters*, 81(3):530–533.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag.

Curilem, G., Vergara, J., Fuentealba, G., Acuña, G., and Chacón, M. (2009). Classification of seismic signals at villarrica volcano (chile) using neural networks and genetic algorithms. *Journal of volcanology and geothermal research*, 180(1):1–8.

Curilem, M., Vergara, J., San Martin, C., Fuentealba, G., Cardona, C., Huenupan, F., Chacón, M., Khan, M. S., Hussein, W., and Yoma, N. B. (2014). Pattern recognition applied to seismic signals of the llaima volcano (chile): An analysis of the events' features. *Journal of volcanology and geothermal research*, 282:134–147.

Dai, H. and MacBeth, C. (1995). Automatic picking of seismic arrivals in local earthquake data using an artificial neural network. *Geophysical journal international*, 120(3):758–774.

Dai, H. and MacBeth, C. (1997). The application of back-propagation neural network to automatic picking seismic arrivals from single-component recordings. *Journal of Geophysical Research: Solid Earth*, 102(B7):15105–15113.

Del Pezzo, E., Esposito, A., Giudicepietro, F., Marinaro, M., Martini, M., and Scarpetta, S. (2003). Discrimination of earthquakes and underwater explosions using neural networks. *Bulletin of the Seismological Society of America*, 93(1):215–223.

Evans, J. R. and Allen, S. S. (1983). A teleseism-specific detection algorithm for single short-period traces. *Bulletin of the Seismological Society of America*, 73(4):1173–1186.

Gardner, M. W. and Dorling, S. (1998). Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences. *Atmospheric environment*, 32(14-15):2627–2636.

Géron, A. (2017). *Hands-on machine learning with Scikit-Learn and TensorFlow: Concepts, tools, and techniques to build intelligent systems*. O'Reilly Media.

Ibáñez, J. M., Benítez, C., Gutiérrez, L. A., Cortés, G., García-Yeguas, A., and Alguacil, G. (2009). The classification of seismo-volcanic signals using hidden markov models as applied to the stromboli and etna volcanoes. *Journal of Volcanology and Geothermal Research*, 187(3-4):218–226.

Ibs-von Seht, M. (2008). Detection and identification of seismic signals recorded at krakatau volcano (indonesia) using artificial neural networks. *Journal of Volcanology and Geothermal Research*, 176(4):448–456.

Jordan, M. I. and Mitchell, T. M. (2015). Machine learning: Trends, perspectives, and prospects. *Science*, 349(6245):255–260.

Kong, Q., Trugman, D. T., Ross, Z. E., Bianco, M. J., Meade, B. J., and Gerstoft, P. (2019). Machine learning in seismology: Turning data into insights. *Seismological Research Letters*, 90(1):3–14.

Kortström, J., Uski, M., and Tiira, T. (2016). Automatic classification of seismic events within a regional seismograph network. *Computers & Geosciences*, 87:22–30.

Krischer, L., Megies, T., Barsch, R., Beyreuther, M., Lecocq, T., Caudron, C., and Wassermann, J. (2015). Obspy: A bridge for seismology into the scientific python ecosystem. *Computational Science & Discovery*, 8(1):014003.

Megies, T., Beyreuther, M., Barsch, R., Krischer, L., and Wassermann, J. (2011). Obspy–what can it do for data centers and observatories? *Annals of Geophysics*, 54(1):47–58.

Mitchell, T. M. et al. (1997). *Machine learning*. McGraw-hill New York.

Murphy, K. P. (2012). *Machine learning: a probabilistic perspective*. MIT press.

Nwankpa, C., Ijomah, W., Gachagan, A., and Marshall, S. (2018). Activation functions: Comparison of trends in practice and research for deep learning. *arXiv preprint arXiv:1811.03378*.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al. (2011). Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830.

Ruano, A. E., Madureira, G., Barros, O., Khosravani, H. R., Ruano, M. G., and Ferreira, P. M. (2013). A support vector machine seismic detector for early-warning applications. *IFAC Proceedings Volumes*, 46(20):405–410.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *nature*, 323(6088):533–536.

Sharma, B., Kumar, A., and Murthy, V. (2010). Evaluation of seismic events detection algorithms. *Journal of the Geological Society of India*, 75(3):533–538.

Shearer, P. M. (2019). *Introduction to seismology*. Cambridge university press.

Shinde, P. P. and Shah, S. (2018). A review of machine learning and deep learning applications. In *2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA)*, pages 1–6. IEEE.

Tiira, T. (1999). Detecting teleseismic events using artificial neural networks. *Computers & Geosciences*, 25(8):929–938.

Trnkoczy, A. (2009). Understanding and parameter setting of sta/lta trigger algorithm. In *New Manual of Seismological Observatory Practice (NMSOP)*, pages 1–20. Deutsches GeoForschungsZentrum GFZ.

Wang, J. and Teng, T.-L. (1995). Artificial neural network-based seismic detector. *Bulletin of the Seismological Society of America*, 85(1):308–319.

Wang, J. and Teng, T.-l. (1997). Identification and picking of s phase using an artificial neural network. *Bulletin of the Seismological Society of America*, 87(5):1140–1149.

Wiszniowski, J., Plesiewicz, B. M., and Trojanowski, J. (2014). Application of real time recurrent neural network for detection of small natural earthquakes in poland. *Acta Geophysica*, 62(3):469–485.

Withers, M., Aster, R., Young, C., Beiriger, J., Harris, M., Moore, S., and Trujillo, J. (1998). A comparison of select trigger algorithms for automated global seismic phase and event detection. *Bulletin of the Seismological Society of America*, 88(1):95–106.

Zhao, Y. and Takano, K. (1999). An artificial neural network approach for broadband seismic phase picking. *Bulletin of the Seismological Society of America*, 89(3):670–680.

A. Apéndice

A.1. *Backpropagation*

En el algoritmo de *Backpropagation* para cada instancia de entrenamiento se calcula una predicción con la red, se mide el error de este con el valor objetivo, para luego ir en reversa en donde en cada capa se mide la contribución de las distintas neuronas al resultado y error anteriormente medido, actualizando los pesos en cada conexión con el objetivo de minimizar el error, hasta llegar a la capa de neuronas de entrada (Rumelhart et al., 1986).

A.2. Tensorflow y Scikit

Como base y quien se encarga de la mayoría de las funciones corresponde a *Tensorflow*, mientras que Scikit es utilizado para herramientas específicas debido que posee una estructura mucho más simple en el uso de sus rutinas, por lo que es fácil de implementar.

Tensorflow corresponde a una librería/software de acceso libre ideal para computación numérica, además de que se encuentra optimizado y desarrollado para trabajos de Machine-learning de pequeña hasta gran escala (redes simples para principiantes hasta grandes redes profundas de numerosas capas que requieren alta capacidad computacional). Su funcionamiento es distinto a Scikit y a lo que se acostumbra en Python, pero bastante simple de entender. Primero se define lo que es básicamente un flujo de trabajo, o sea, un gráfico de computación y sus distintas partes, nodos y operaciones matemáticas, para que luego Tensorflow tome el gráfico y lo corra de principio a fin, realizando las operaciones necesarias de forma rápida y eficiente (Abadi et al., 2015).

En comparación a Tensorflow, Scikit es mucho más simple en su estructura, debido a que su enfoque es proporcionar las herramientas necesarias a quienes empiezan a trabajar en Machine-learning, sin perder la eficiencia y desempeño de

otros software de gran nivel. Al ser una librería de Python junto con Tensorflow, sus herramientas y rutinas pueden combinarse, además de hacer el cambio a aplicaciones de alto nivel mucho más fácil, pues se integran fácilmente ambas librerías (Pedregosa et al., 2011).