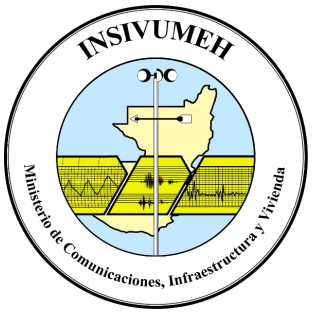


INSIVUMEH

Departamento de Investigación y servicios Geofísicos

PRÁCTICA FINAL SUPERVISADA: Análisis de Red Neuronal para detección de fases sísmicas

Julio Antonio Medina Chután 200412554
Escuela no Facultativa de Ciencias Físicas y Matemáticas
Universidad de San Carlos de Guatemala

Supervisada por:

Lic. Robin Yani

Índice general

Capítulo 1

Introducción General

Debido al rápido crecimiento de estaciones sísmicas en los últimos años el numero de datos de eventos sísmicos ha incrementado exponencialmente [5], este incremento se da tanto por el mayor numero de estaciones como también por la mayor capacidad de almacenamiento, procesamiento y distribución de los datos. Este incremento de datos sugiere el uso de herramientas de procesamiento automático para las distintas tareas de análisis que se realizan en las agencias sismológicas. Una de las tareas fundamentales en el análisis de sismogramas es la identificación de fases sísmicas, es decir identificar manualmente a partir de la forma de onda del evento sísmico el tiempo de llegada de las ondas sísmicas, tales como la onda primaria P y la secundaria S, estos tiempos de llegada son necesarios para determinar la localización de los hipocentros de los eventos sísmicos así como el tiempo en que se originó el mismo.
Numerosos métodos de detección automática han sido desarrollados a través de los años, siendo el mas común el método de STA/LTA en donde se obtiene la razón de la energía en un periodo corto entre la energía en un periodo largo, cualquier valor arriba de un umbral predeterminado se interpreta como una llegada de fase sísmica, una de la debilidades de este método es que no se puede determinar la diferencia entre las ondas P y S. Este método ha sido modificado y en algunas versiones se incluye a una función envolvente y un umbral dinámico de señal(Baer, Kradolfer, 1987). Hay muchos mas acercamientos al problema de detección automática como el uso de estadística de ordenes superiores, métodos que usan polarización de ondas y otros; pero a pesar de esto, los métodos automáticos no pueden alcanzar el desempeño que un analista experimentado posee. Esto se atribuye a la naturaleza compleja tanto de las fuentes de ondas sísmicas y de la propagación por el medio, variaciones en la atenuación de la onda, interferencia de ruido, mecanismo de la fuente, partición de energía en las interfaces, etc., todo estos factores afectan la forma de onda observada por el sismometro. Con la exitosa utilización de la inteligencia artificial en distintos problemas de alta complejidad se quiere investigar como los desarrollos teóricos de la inteligencia artificial, específicamente el aprendizaje automático en su forma de redes neuronales puede asistir al análisis sísmico de señales.

1.1. Aprendizaje automático en sismología

El uso de la inteligencia artificial en su forma de aprendizaje automático y redes neuronales se ha vuelto cada vez mas presente en la sismología con aplicaciones que van desde la identificación de señales sísmicas no identificadas con anterioridad, el reconocimiento de patrones en las señales sísmicas, hasta la extracción de características que pueden mejorar el entendimiento de los procesos físicos involucrados. Los algoritmos de aprendizaje automático se enfocan en aprender representaciones de los datos en vez de extraer características individuales, en una red neuronal múltiples capas de procesamiento proveen distintos niveles de abstracción para procesar un vector multidimensional. Un aspecto emocionante de las redes neuronales aplicadas a la sismología es el potencial de encontrar patrones no vistos con anterioridad en los datos registrados de eventos pasados. Desarrollos recientes del aprendizaje automático y de las redes neuronales en tareas como la vision computarizada y reconocimiento del habla [3], sugieren la inclusión de las técnicas de la inteligencia artificial como una herramienta de apoyo para el análisis sismológico, por ejemplo las redes neuronales convolucionales que son una forma de aprendizaje supervisado han tenido resultados excepcionales al clasificar entradas multidimensionales como archivos de video, audio e imágenes(LeCun, Bengio, 1995; Kriszhevsky 2012). Algunas de las aplicaciones del aprendizaje automático en la sismología son :

- Detección de terremotos y detección de fases sísmicas.
- Sistema de alerta temprana de terremotos(*EEW, Earthquake Early Warning*).
- Predicción del movimiento del suelo(*Ground-motion prediction*).
- Tomografía sísmica.
- Geodesia de terremotos.

En una red neuronal las características son extraídas por medio de la optimización de los parámetros internos de la red a través de un método de optimización computacional. Las propiedades intrínsecas de las redes neuronales las hacen idóneas para la clasificación de señales naturales, en especial las redes neuronales convolucionales. Una ventaja de la redes neuronales para su utilización en sismología es que la disponibilidad de datos previamente etiquetados correctamente es muy grande. Por ejemplo para el caso de identificación de fases sísmicas, en el INSIVUMEH se tiene un catalogo de eventos sísmicos con fases identificadas que crece diariamente, estos datos son los datos con los que se entrena a la red. En los últimos años se ha visto la adopción de algunos investigadores en los métodos de aprendizaje automático, por ejemplo ver [6], [3] y [5]. Resultados preliminares indican que los métodos de aprendizaje automático pueden alcanzar y de hecho sobrepasar el desempeño en clasificación de fases sísmicas que tienen los analistas.

Estos resultados sugieren la elaboración de una red neuronal para la tarea de identificación de fases sísmicas en el INSIVUMEH con el propósito de evaluar su desempeño y determinar que arquitectura de red neuronal es la adecuada.

1.2. Visión general del aprendizaje automático

En esencia las redes neuronales aprenden de los datos usando teoría de la probabilidad. Los algoritmos de aprendizaje automático pueden ser agrupados en dos categorías principales: aprendizaje supervisado, aprendizaje no supervisado, ver figura 1.1. En el aprendizaje supervisado se proveen etiquetas objetivo y en el no supervisado no se da este tipo de información. Las etiquetas objetivo son las salidas que deseamos que nuestro algoritmo devuelva, por ejemplo si se esta trabajando en un clasificador binario de imágenes para determinar si en las imágenes hay perros o no, entonces una foto con algún perro tendría una etiqueta con el numero $t_k = 1$ indicando la presencia de un perro en la imagen, por el otro lado cualquier imagen que no contenga perro tendría la etiqueta $t_k = 0$ indicando la ausencia de perros en las imágenes. El aprendizaje automático supervisado que abarca el modelado predictivo y que trabaja con conjunto de datos etiquetados se puede subdividir en algoritmos de regresión y clasificación, dependiendo de la naturaleza de las etiquetas, para etiquetas de tipo categórico se usan algoritmos de clasificación y para etiquetas de tipo cuantitativo se utilizan algoritmos de regresión. Los algoritmos de de aprendizaje no supervisado pueden clasificarse en algoritmos de agrupamiento y de reducción de dimensionalidad, dependiendo si se quiere agrupar los datos en categorías basándose en la similitud de los datos, o simplemente reducir la dimension de los datos es decir convertir a un vector de dimension N a uno de dimension M, donde $M < N$. Hay otros algoritmos de aprendizaje automático mas sofisticados, tal es el caso del aprendizaje semi-supervisado y los algoritmos de aprendizaje de reforzamiento. Hay una gran

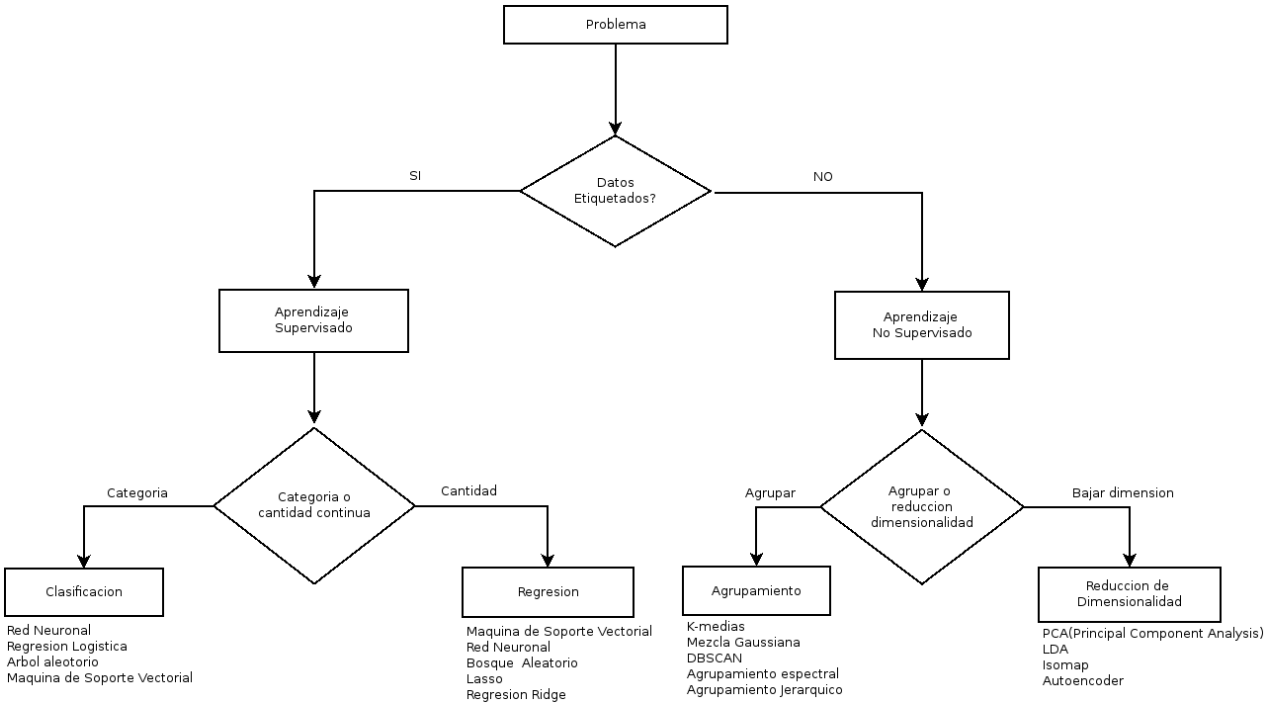


Figura 1.1: Diagrama que muestra los diferentes tipos de algoritmos de aprendizaje automático.

diversidad en la implementación de algoritmos de aprendizaje automático, pero la mayoría de ellos tienden a seguir un flujo de trabajo bastante estándar ver diagrama 1.2. En el paso 1 los datos son datos con colectados y se particionan los datos en conjuntos de entrenamiento y prueba, un aspecto clave del aprendizaje automático es que el modelo se entrena con un subconjunto aleatorio del la totalidad de los datos, ademas se tiene otro subconjunto independiente para validar y verificar el modelo. En el paso 2, se preprocesan los datos, esto incluye la limpieza de los datos, formateo de datos, normalización de datos, regularización de datos, calculo de características, re-muestreo. Lo que se pretende en esta etapa es tener un conjunto de datos consistente y que tenga propiedades que aceleren el aprendizaje, por ejemplo la normalización de los datos es crucial para que el algoritmo de optimización de parámetros converja mas rápidamente. Otro aspecto clave es la regularización de los datos, por ejemplo: las entradas de una red neuronal se dan como un vector multidimensional, si nuestros datos en su forma original tienen distintos tamaños dimensionales se tendría que crear rutinas para que todos los datos de nuestro conjunto de entrenamiento tengan la misma dimension. En el paso 3 se entrena al modelo, esto incluye aplicar métodos de optimización numérica para encontrar los parámetros internos que tenga la arquitectura del modelo a entrenar. Otro aspecto determinante es la selección del modelo, esto por lo regular se hace dependiendo del tipo de problema que se quiera resolver y de la naturaleza de los datos de entrenamiento. En el paso 4 se evaluá el desempeño del modelo utilizando el conjunto de datos de prueba, esta evaluación se hace por lo regular con métricas que comúnmente dependen de la función de error del modelo y del numero de predicciones acertadas. Como etapa final en el paso 5 se pone al modelo en producción haciendo predicciones en datos nuevos que el algoritmo no ha visto para entrenar o para validar.

Figura 1.2: Diagrama genérico que muestra el flujo de trabajo de los algoritmos de aprendizaje automático.

Capítulo 2

Objetivos

- Analizar los sismogramas con fases S y P picadas correctamente.
- Revisar documentación del software SEISAN.
- Revisar documentación del API ObsPy.
- Revisar datos del catalogo de sismogramas disponible.
- Desarrollar rutinas para la adquisición de datos.
- Normalizar los datos de sismogramas.
- Regularizar los datos de sismogramas.
- Investigar el estado actual del aprendizaje automático aplicado a la sismología.
- Crear conjunto de datos para distintas tareas de investigación.
- Investigar el desarrollo de una red neuronal que recibe como entrada los datos de los sismogramas.
- Determinar que herramienta(*Octave, TensorFlow*) es la adecuada para desarrollar una red neuronal.
- Desarrollar una red neuronal de prueba, para evaluar el comportamiento que tiene para identificar fases sísmicas.
- Hacer pruebas con distintos conjuntos de datos para entrenar a la red neuronal e identificar que modelos tienen éxito al identificar fases sísmicas con un conjunto de datos independiente que no haya sido utilizado para entrenar a la red.
- Identificar estrategias a seguir para mejorar el desempeño de la red neuronal de prueba.
- Investigar la utilización de una red neuronal convolucional parar la identificación de fases sísmicas.

Capítulo 3

Marco Teórico

3.1. Sismología

La sismología es el estudio de ondas elásticas o de sonido en la tierra solida. Las ondas sísmicas se generan en una fuente que puede ser natural, como un terremoto, o artificial como un explosión. Las ondas resultantes se propagan a través del medio que en este caso es una porción de la tierra y son registradas en un receptor, como se ve en en la figura 3.1.

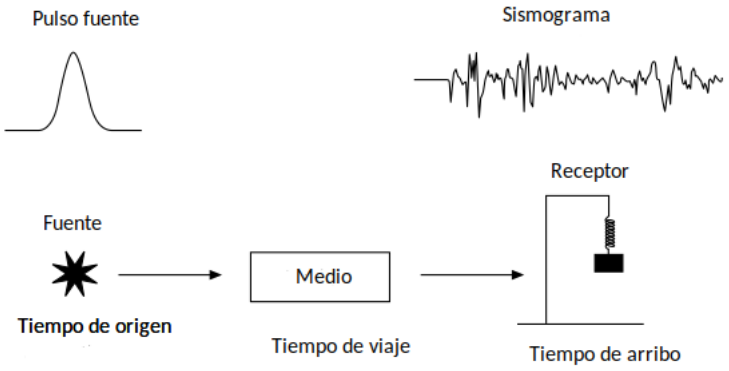


Figura 3.1: Diagrama esquemático de experimento sísmico

Un sismograma es el registro del movimiento terrestre capturado por un receptor llamado sismómetro, inventado en 1842 por el físico escoces James Forbes. Los sismogramas contienen información de la fuente y del medio por el cual se propagan las ondas resultantes. Esta información puede tomar varias formas. Las ondas proveen información de la locación y naturaleza de la fuente que las genero. Si se conoce el tiempo de origen de cuando las ondas dejaron la fuente, entonces el tiempo de llegada registrado en el receptor nos da también el tiempo de viaje de la onda requerido para pasar a través del medio y por lo tanto se obtiene también información de la velocidad a la que viaja la onda y de esta manera se infieren propiedades físicas del medio. Aun mas, debido a que la amplitud y la forma de la onda que dejaron la fuente se ven afectadas por la propagación a través

del medio, la señales observadas en los sismogramas proveen información adicional del medio.
La sismología es la herramienta principal para el estudio del interior terrestre ya que la accesibilidad al interior de la tierra es bastante limitado. De esta manera la sismología ayuda a mapear el interior terrestre y estudiar la distribución de sus propiedades físicas. También se utiliza la sismología como método principal para el estudio de terremotos, la mayoría de la información de las fallas durante un terremoto se determina a través del sismograma resultante. Los análisis de sismogramas también hace posible investigar los procesos físicos que ocurren previamente, durante y después de los terremotos en las fallas, tales estudios son útiles en evaluar las amenazas que los terremotos plantean a la sociedad.

3.1.1. Ondas sísmicas

Las ondas elásticas que son generadas por un terremoto pueden ser registradas local o globalmente por los sismometros. El principio del funcionamiento de los sismometros puede verse en la figura 3.2 que muestra un análogo mecánico del sismometro, donde hay una traslación directa del movimiento de la tierra(en la mayoría de los casos muy pequeño para ser percibido) al movimiento de la tierra amplificado plasmado por el sismometro. Hoy en dia casi todos los sismometros en funcionamiento están basados en registros digitales donde el movimiento amplificado de la tierra es registrado como una serie de números que pueden graficarse para que se vean como un sismograma tradicional.
Los tiempos de arribo de grupos de ondas cuya amplitud sea considerablemente mayor que el ruido de fondo se conocen como fases sísmicas. Teniendo los registros de eventos sísmicos en los sismogramas, es crucial identificar a las fases sísmicas(vea figura 3.3). Hay varios tipos de fases sísmicas que pueden ser distinguidas en un sismograma, la distinción se da en base a la naturaleza de las ondas sísmicas y de la estructura terrestre.

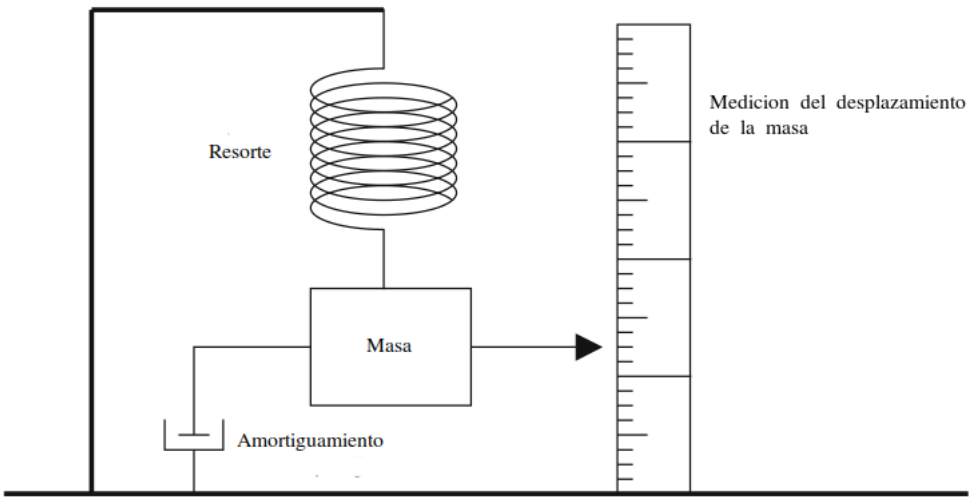


Figura 3.2: Diagrama esquemático de experimento sísmico

Los principales tipos de ondas sísmicas son

- Ondas P: Ondas compresionales, la letra P es para denotar que son las ondas principales ya que estas son las primeras en llegar. Una velocidad típica en la corteza superior(profundidad de menos de 15 km) es de 6 km/s.
- Ondas S: Ondas de corte, la letra S es para denotar que se tratan de ondas secundarias ya que arriban después de las ondas P. Una velocidad típica en la corteza superior es de 3,5 km/s.
- Ondas Superficiales: Como su nombre lo indica viajan en la superficie terrestre. Se producen de varias maneras, las mas importantes son: las ondas Love, que se dan por la reflexión multiplicada de ondas S superpuestas y se tienen también a las ondas Rayleigh que son combinaciones de ondas P y S. Velocidades típicas para estas ondas superficiales pueden ser de $3{,}5 - 4{,}5$ km/s. Estas ondas siempre llegan después que las ondas S.

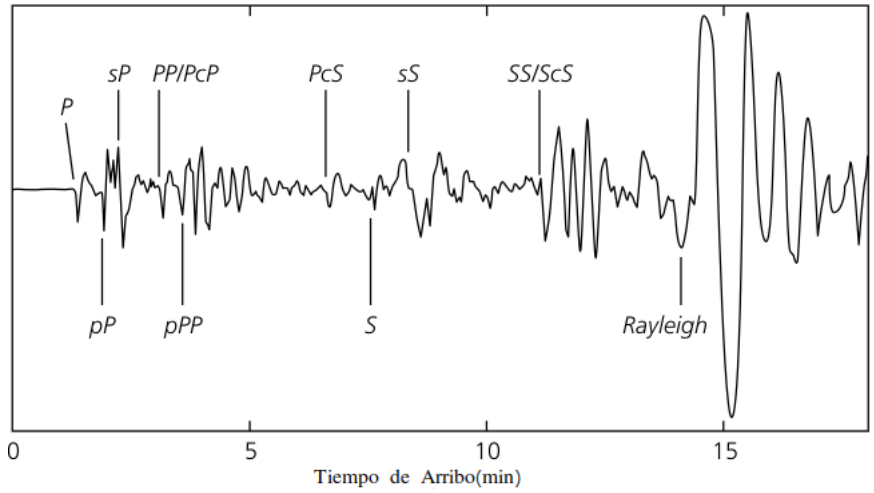


Figura 3.3: Ejemplo de identificación de fases sísmicas en un sismograma.

3.2. Redes Neuronales

El termino 'Red Neuronal' tiene sus orígenes en intentos de hallar representaciones matemáticas que pudieran modelar el procesamiento de información en sistemas biológicos(McCulloh & Pitts, 1943; Widrow & Hoff, 1960; Rumelhart, 1960), de hecho el termino ha sido utilizado ampliamente para describir a un gran numero de modelos dando lugar a confusiones sobre su plausibilidad biológica. Aquí se quiere tener un enfoque de las redes neuronales como modelos eficientes para el reconocimiento de patrones estadísticos.

3.2.1. El algoritmo del perceptron

Este modelo ocupa un lugar importante en la historia de los algoritmos de reconocimiento de patrones desarrollado por Rosenblatt(1962). Corresponde a un modelo de dos

clases en el que el vector de entrada $\mathbf{x}$ es primeramente transformado usando una función no lineal fija que da como resultado un vector de características $\phi(\mathbf{x})$ que después es usado para construir un modelo lineal generalizado de la forma

$$y(\mathbf{x}) = f\big(\mathbf{w}^T\phi(\mathbf{x})\big) \tag{3.1}$$

donde la función no lineal de activación $f(\cdot)$ esta dada por la función step de la forma

$$f(a) = \begin{cases} +1, & a \geq 0 \\ -1, & a < 0 \end{cases} \tag{3.2}$$

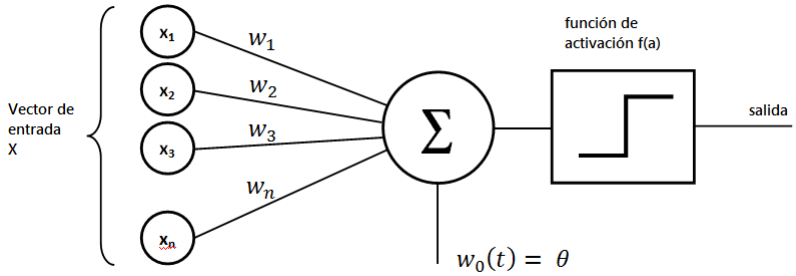


Figura 3.4: Diagrama del perceptron

3.2.2. Funciones de Propagación de la Red(*Feed-forward Network Functions*)

Los modelos para regresión y clasificación están basados en combinaciones lineales de funciones base fijas no lineales $\phi_j(\mathbf{x})$ y toman la forma

$$y(\mathbf{x}, \mathbf{w}) = f\left(\sum_{j=1}^{M} w_j\phi_j(\mathbf{x})\right) \tag{3.3}$$

donde $f(\cdot)$ es una función de activación no lineal en el caso de clasificación y la identidad para el caso de regresión. El objetivo es extender este modelo al hacer que las funciones base $\phi_j(\mathbf{x})$ dependan de parámetros y después permitir que estos parámetros y los coeficientes $\{w_j\}$ sean ajustados durante el entrenamiento. Es claro que hay muchas maneras de construir funciones bases parametrizables no lineales. Las redes neuronales usan funciones base que siguen la misma forma que 3.3 de tal manera que cada función base es en si misma una función no lineal de la combinación lineal de las entradas, donde los coeficientes en la combinación lineal son parámetros adaptables.
Esto lleva al modelo básico de red neuronal, que puede ser descrito como una serie de transformaciones funcionales. Primero se construyen M combinaciones lineales de las variables de entrada $x_1, \ldots, x_D$ de la forma

$$a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)} \tag{3.4}$$

donde $j = 1, \ldots, M$ y el superindice (1) indica que los parámetros corresponden a la primera 'capa' de la red. Los parámetros $w_{ji}^{(1)}$ son denominados pesos y los parámetros $w_{j0}^{(1)}$ como sesgos(*biases*). Las cantidades a_j son conocidas como activaciones, estas son transformadas después utilizando una función de activación no lineal y diferenciable $h(\cdot)$ que resulta en

$$z_j = h(a_j). \tag{3.5}$$

Estas cantidades corresponden a las salidas de las funciones base en 3.3, que en el contexto de las redes neuronales se les conoce como unidades ocultas. Generalmente se escogen funciones no lineales $h(\cdot)$ de tipo sigmoidal tales como la función logística sigmoid o la función tanh. Siguiendo de la ecuación 3.3 estos valores son nuevamente combinados linealmente para dar las unidades de activación de salida

$$a_k = \sum_{j=1}^{M} w_{kj}^{(2)} z_j + w_{k0}^{(2)} \tag{3.6}$$

donde $k = 1, \ldots, K$ y K es el numero total de salidas. Esta transformación corresponde a la segunda capa de la red y de nuevo $w_{kj}^{(2)}$ son los parámetros de peso de la segunda capa y $w_{k0}^{(2)}$ son los parámetros de sesgo(*biases*) de la segunda capa. Finalmente las unidades de activación de salida son transformadas usando una función de activación apropiada para dar un conjunto de salidas de la red y_k. La elección de la función de activación es determinada por la naturaleza de los datos y de la distribución asumida de las variables objetivo. Para problemas de regresión la función de activación es la identidad de tal manera que $y_k = a_k$. De igual manera para problemas de clasificación binaria, cada unidad de activación de salida es transformada usando una función logística sigmoidal tal que

$$y_k = \sigma(a_k) \tag{3.7}$$

donde

$$\sigma(a) = \frac{1}{1 + \exp(-a)}. \tag{3.8}$$

Finalmente para problemas de clasificación de múltiples clases se utiliza una función de activación softmax que tiene la forma

$$\begin{aligned} p(C_k|x) &= \frac{p(x|C_k)p(C_k)}{\sum_j p(x|C_j)p(C_j)} \\ &= \frac{\exp(a_k)}{\sum_j a_j} \end{aligned} \tag{3.9}$$

la elección de la función de activación de salida se discute en detalle en la siguiente sección. Se pueden combinar la etapas descritas anteriormente para dar como resultado la función global de la red que para el caso de funciones de activación sigmoidales de salida toma la forma

$$y_k(\mathbf{x}, \mathbf{w}) = \sigma\left(\sum_{j=1}^{M} w_{kj}^{(2)} h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)} \right) + w_{k0}^{(2)} \right) \tag{3.10}$$

donde el conjunto de todos los parámetros de pesos y sesgos han sido agrupados en el vector $\mathbf{w}$. De esta manera se puede ver como esta construcción de red neuronal no es mas que una simple función no lineal del conjunto de variables de entrada $\{x_i\}$ a otro conjunto de variables de salida $\{y_k\}$ controlada por el vector $\mathbf{w}$ de parámetros ajustables.
Esta función puede ser representada gráficamente en la forma del diagrama mostrado en la figura 3.5. El proceso de evaluar la expresión 3.10 puede ser interpretado como una propagación directa(*forward propagation*) de información a través de la red.

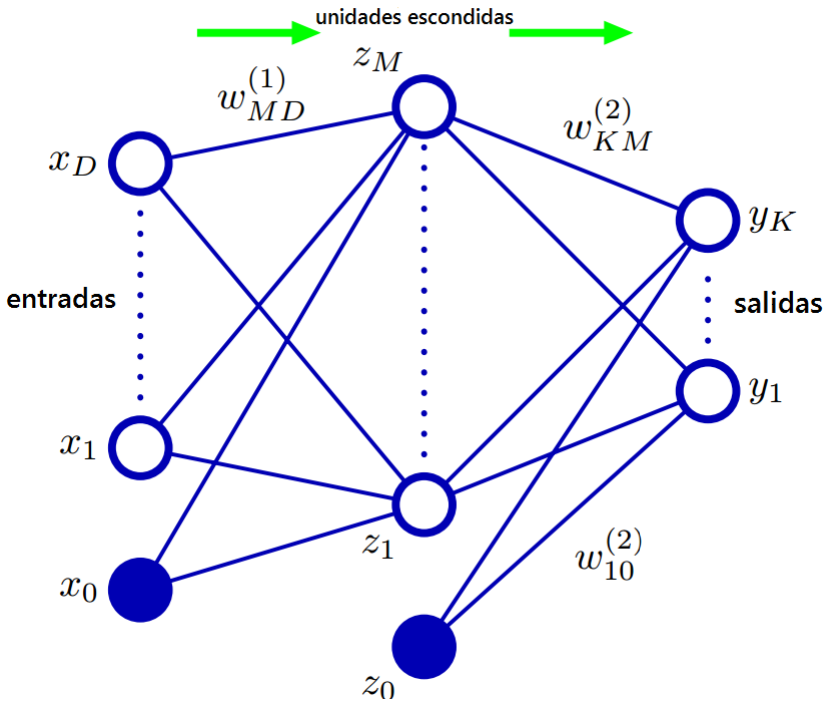


Figura 3.5: Diagrama de red para una red neuronal de dos capas correspondiente a la expresión 3.10. Las variables de entrada, ocultas y de salida son representadas por nodos y los parámetros de peso están representados por conexiones entre los nodos donde los parámetros de sesgo son denotados por conexiones que vienen de variables adicionales de entrada x_0 y escondida z_0. Las flechas verdes indican la dirección en la que la información fluye durante la propagación(*forward propagation*).

Los parámetros de sesgo en la ecuación 3.4 pueden ser absorbidos en un conjunto de parámetros de peso al definir una variable de entrada adicional cuyo valor se fija en $x_0 = 1$ para que la expresión 3.4 tome la forma

$$a_j = \sum_{i=0}^{D} w_{ji}^{(1)} x_i. \tag{3.11}$$

Similarmente se pueden absorber los sesgos de la segunda capa en los pesos de esta capa para que la función global de la red se convierta en

$$y_k(\mathbf{x}, \mathbf{w}) = \sigma\left(\sum_{j=0}^{M} w_{kj}^{(2)} h\left(\sum_{i=0}^{D} w_{ji}^{(1)} x_i \right) \right). \tag{3.12}$$

Como puede verse en la figura 3.5, la red neuronal se compone de dos etapas de procesamiento, teniendo en cada etapa una semejanza con el modelo del percenptron y por esta razón se conoce a las redes neuronales como perceptron de múltiples capas(*multilayer percepton, MLP*). Una diferencia clave con el perceptron es que la red neuronal usa funciones continuas no lineales en las unidades ocultas mientras que el perceptron usa la función step 3.1, esto significa que la función de la red neuronal es diferenciable con respecto a los parámetros de la red $\mathbf{w}$, un aspecto que juega un rol central en el proceso de entrenamiento de la red.
Si las funciones de activación de todas la unidades ocultas en la red son lineales entonces para tal red siempre se puede encontrar una red equivalente sin unidades ocultas, esto es consecuencia del hecho que una composición de transformaciones lineales sucesivas es en sí misma una transformación lineal, sin embargo si el numero de unidades ocultas es menor al numero de entradas o salidas entonces las transformaciones que la red puede generar no son las transformaciones lineales mas generales posibles de las entradas a las salidas, esto se debe a que se pierde información en la reducción de la dimensionalidad de la unidades ocultas.
La arquitectura de red mostrada en la figura 3.5 es la mas usada comúnmente en la practica. Esta arquitectura puede ser fácilmente generalizada por ejemplo considerando capas adicionales de procesamiento cada una consistiendo en una combinación lineal ponderada de la forma de la expresión 3.6 seguido por la transformación utilizando una función de activación no lineal de cada unidad en la capa adicional.

Simetrías del espacio de pesos

Una propiedad importante de las redes de propagación directa es que hay múltiples elecciones del vector de pesos $\mathbf{w}$ que resultan en el mismo mapeo funcional de entradas a las salidas de la red. Por ejemplo al considerar una red de dos capas como se ve en la figura 3.5 con M unidades ocultas que tienen a la función tanh como función de activación y hay una conexión completa entre las capas, si se cambia el signo de los parámetros de peso y sesgo que están conectados a una unidad arbitraria para alguna entrada de la red en especifico, entonces el signo de la unidad sera revertido, debido a que la función tanh es una función impar tal que $\tanh(-a) = -\tanh(a)$, esta transformación puede ser compensada exactamente al revertir el signo del vector de pesos de la conexiones salientes

de dicha unidad. Lo que se ve de este ejemplo se ve que al cambiar el signo de un grupo particular de parámetros de pesos y sesgos se conserva el mapeo funcional de entradas a salidas de la red, de esta manera se han encontrado 2 vectores de pesos distintos que dan lugar al mismo mapeo funcional. Si se tienen M unidades ocultas habrían M simetrías de inversión del signo, por lo que habrían 2^M vectores de peso equivalentes.
De la misma manera si se intercambian los pesos de las conexiones entrantes y salientes de una unidad en especifico con los pesos y sesgos de otra unidad, este intercambio deja el mapeo funcional de la red intacto pero se tiene un vector de pesos distinto. Para M unidades ocultas cualquier vector de pesos pertenecería a un conjunto de $M!$ vectores de pesos equivalentes. En totalidad la red tendría un factor de simetrias en el espacio de pesos de $M!2^M$. La existencia de estas simetrías no es una propiedad particular de la funcion tanh, sino que aplica a una gran variedad de funciones de activación.

3.2.3. Entrenamiento de la Red

Se ha visto que las redes neuronales como una clase general de funciones paramétricas no lineales de un vector $\mathbf{x}$ de variables de entrada a un vector $\mathbf{y}$ de variables de salida. Un acercamiento simple al problema de determinar los parámetros de la red es hacer una analogía con la regresión de curvas polinomiales y de esta manera minimizar la función suma de cuadrados de la diferencia de los errores. Dado un conjunto de entrenamiento de vectores de entrada $\{\mathbf{x}_n\}$, donde $n = 1, \ldots, N$, junto con el conjunto correspondiente de vectores objetivos $\mathbf{t}_n$, se quiere minimizar la función de error

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\left\|\mathbf{y}(\mathbf{x}_n, \mathbf{w}) - \mathbf{t}_n\right\|^2. \tag{3.13}$$

Se empieza discutiendo problemas de regresión y se consideran problemas de una variable objetivo t que puede tomar cualquier valor real.

$$p(t|\mathbf{x}, \mathbf{w}) = \mathcal{N}(t|y(\mathbf{x}, \mathbf{w}), \beta^{-1}) \tag{3.14}$$

donde $\mathcal{N}(\cdot)$ es la distribucion Gaussiana y β es la precision(inverso de la varianza) del ruido Gaussiano. Esta es una asunción restrictiva, para ver un metodo de generalizar esto se recomienda [1]. Para la condición impuesta por la distribución dada por 3.14 es suficiente tomar a la función de activación de salida como la identidad ya que tal red puede aproximar cualquier función lineal de $\mathbf{x}$ a y. Dado un conjunto de datos de N observaciones independientes e idénticamente distribuidas $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$, junto con sus valores objetivos correspondientes $\mathbf{t} = \{t_1, \ldots, t_N\}$, se puede construir la función de verosimilitud(*likelihood*) correspondiente

$$p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \beta) = \prod_{n=1}^{N} p(t_n|\mathbf{x}_n, \mathbf{w}, \beta) \tag{3.15}$$

. Tomando el logaritmo negativo de la expresión anterior 3.15 se obtiene la función de error

$$\frac{\beta}{2}\sum_{n=1}^{N}\{y(\mathbf{x}_n, \mathbf{w}) - t_n\}^2 - \frac{N}{2}\ln\beta + \frac{N}{2}\ln(2\pi) \tag{3.16}$$

que puede ser utilizada para aprender los parámetros $\mathbf{w}$ y β como una optimización trivial(maximizar la verosimilitud), utilizando métodos como descenso del gradiente u otros. Es usual que en la literatura de redes neuronales se considere minimizar la función de error en vez de maximizar el logaritmo de la verosimilitud(*likelihood*), por lo que es conveniente seguir esta convención. Se considera primero la determinación de $\mathbf{w}$, maximizar la función de verosimilitud es equivalente a minimizar la función de error de suma de cuadrados dada por

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{y(\mathbf{x}_n, \mathbf{w}) - t_n\}^2 \tag{3.17}$$

donde se han descartado las constantes aditivas y multiplicativas. El valor de $\mathbf{w}$ que se encuentra minimizando $E(\mathbf{w})$ de denota como $\mathbf{w}_{ML}$ ya que corresponde a la solución de verosimilitud máxima(*maximum likelihood*). En practica el comportamiento no lineal de la función de salida $y(\mathbf{x}_n, \mathbf{w})$ causa que la función de error $E(\mathbf{w})$ sea cóncava por lo que en practica máximos locales de la función de verosimilitud pueden ser determinados al encontrar mínimos locales de la función de error como se discutió anteriormente.
Habiendo encontrado $\mathbf{w}_{ML}$, el valor de β puede determinarse al minimizar el logaritmo negativo de la función de verosimilitud que da como resultado

$$\frac{1}{\beta_{ML}} = \frac{1}{NK}\sum_{n=1}^{N}\{y(\mathbf{x}_n, \mathbf{w}_{ML}) - \mathbf{t}_n\}^2. \tag{3.18}$$

Nótese que la expresión anterior puede ser evaluada cuando la optimización iterativa para encontrar $\mathbf{w}_{ML}$ haya sido completada. Si se tienen múltiples variable objetivo y se asume que se tienen distribuciones condicionales independientes para $\mathbf{x}$ y $\mathbf{w}$ con ruido de precision para β entonces la distribución condicional de los valores objetivo esta dada por

$$p(\mathbf{t}|\mathbf{x}, \mathbf{w}) = \mathcal{N}(\mathbf{t}|\mathbf{y}(\mathbf{x}, \mathbf{w}), \beta^{-1}\mathbf{I}). \tag{3.19}$$

siguiendo los mismos argumentos que se usaron para el caso de una variable objetivo se tiene que los parámetros de peso para tener una verosimilitud máxima se determinan al minimizar la función de error de suma de cuadrados(3.13). El ruido de la precision viene dado por

$$\frac{1}{\beta_{ML}} = \frac{1}{NK}\sum_{n=1}^{N}\left\|y(\mathbf{x}_n, \mathbf{w}_{ML}) - \mathbf{t}_n\right\|^2 \tag{3.20}$$

donde K es el numero de variables objetivo. La asunción de independencia de la observaciones puede dejarse a costo de un problema de optimización un poco mas complejo.
Para el caso de regresión se puede considerar que la red tienen una función de activación de salida que es la identidad de tal manera que $y_k = a_k$. La función de error correspondiente tiene la propiedad que

$$\frac{\partial E}{\partial a_k} = y_k - t_k \tag{3.21}$$

propiedad que es importante cuando se discuta retro-propagación en la sección 3.2.5.
Si ahora se considera el caso de clasificación binaria en el que solo se tiene una variable

objetivo t tal que $t = 1$ denota la 'clase' $\mathcal{C}_1$ y $t = 0$ denota a la clase $\mathcal{C}_2$, se considera una red que tiene una sola función de activación de salida que es la sigmoidal logística

$$y = \sigma(a) \equiv \frac{1}{1 + exp(-a)} \tag{3.22}$$

tal que $0 \leq y(\mathbf{x}, \mathbf{w}) \leq 1$. Se puede interpretar a $y(\mathbf{x}, \mathbf{w})$ como la probabilidad condicional $p(\mathcal{C}_1|\mathbf{x})$, donde se cumple que $p(\mathcal{C}_2|\mathbf{x})$ esta dada por $1-y(\mathbf{x}, \mathbf{w})$. La distribución condicional de los objetivos dados de entrada es entonces una distribución de Bernoulli de la forma

$$p(t|\mathbf{x}, \mathbf{w}) = y(\mathbf{x}, \mathbf{w})^t \{1 - y(\mathbf{x}, \mathbf{w})\}^{1-t}. \tag{3.23}$$

Si se considera un conjunto de entrenamiento de observaciones independientes entonces la función de error que esta dada por el logaritmo negativo de la verosimilitud, es la función de error de entropía cruzada(*cross-entropy*) de la forma

$$E(\mathbf{w}) = -\sum_{n=1}^{N} \{t_n \ln y_n + (1 + t_n) \ln(1 - y_n)\} \tag{3.24}$$

donde y_n denota $y(\mathbf{x}_n, \mathbf{w})$, en este caso no hay análogo a la precisión del ruido β ya que se asume que los valores objetivos están etiquetados correctamente, sin embargo este modelo se puede extender para permitir errores en las etiquetas. Simard([4]) halló que al usar la función de error de entropía cruzada se tiene un entrenamiento de la red mas rápido que en el caso de la función de error de suma de cuadrados de la diferencia de los errores para problemas de clasificación.
Si se tienen K clasificaciones binarias separadas para realizar, entonces se puede utilizar una red que tiene K salidas, cada una de las cuales tiene como función de activación a la función logística sigmoidal(3.22). Asociada a cada salida se tiene una etiqueta de clase binaria $t_k \in \{0, 1\}$, donde $k = 1, \ldots, K$. Si se asume que las etiquetas de clase son independientes, dado el vector de entrada $\mathbf{x}$, entonces la distribución condicional de la variable objetivo es

$$p(\mathbf{t}|\mathbf{x}, \mathbf{w}) = \prod_{k=1}^{K} y_k(\mathbf{x}, \mathbf{w})^{t_k} [1 - y_k(\mathbf{x}, \mathbf{w})]^{1-t_k}. \tag{3.25}$$

Tomando el logaritmo negativo de la función de verosimilitud correspondiente da la siguiente función de error

$$E(\mathbf{w}) = -\sum_{n=1}^{N} \sum_{k=1}^{K} \{t_{nk} \ln y_{nk} + (1 - t_{nk}) \ln(1 - y_{nk})\} \tag{3.26}$$

donde y_{nk} denota $y(\mathbf{x}_n, \mathbf{w})$, de nuevo la derivada de la función de error con respecto a la activación para una unidad de salida en particular toma la forma 3.21 justo como en el caso de regresión.
Finalmente se considera el problema estándar de clasificación de clases múltiples en el que cada entrada es asignada a una de K clases mutuamente excluyentes. Las variables

objetivos binarias $t_k \in \{0, 1\}$ tiene una tienen una codificación de esquema de $1-\text{de}-K$ (*1-of K coding scheme*), en el que $\mathbf{t}$ es un vector de dimension K tal que si la clase es $\mathcal{C}_j$ entonces todos los elementos t_k del vector $\mathbf{t}$ son cero excepto t_j que toma el valor de 1. Por ejemplo si se tienen $K = 5$ clases, entonces si se tiene un patrón(vector de entrada) que corresponde a la clase 2 tendría un vector objetivo $\mathbf{t} = (0, 1, 0, 0, 0)^T$. Las salidas de la red son interpretadas como $y_k(\mathbf{x}, \mathbf{w}) = p(t_k = 1|\mathbf{x})$ que conlleva a la siguiente función de error

$$E(\mathbf{w}) = -\sum_{n=1}^{N}\sum_{k=1}^{K} t_{kn} \ln y_k(\mathbf{x}_n, \mathbf{w}). \tag{3.27}$$

Siguiendo la discusión de la sección (4.3.4) de [1], se tiene que la función de unidad de salida de activación que corresponde al vinculo canónico esta dado por la función softmax

$$y_k(\mathbf{x}, \mathbf{w}) = \frac{\exp(a_k(\mathbf{x}, \mathbf{w}))}{\sum_j \exp(a_j(\mathbf{x}, \mathbf{w}))} \tag{3.28}$$

que satisface $0 \leq y_k \leq 1$ y $\sum_k y_k = 1$. De nuevo la derivada de 3.28 con respecto a las activaciones para una unidad de salida en particular toma la forma conocida de 3.21.
Resumiendo, hay una elección natural tanto de la función de activación de salida para las unidades de la red como para la función de error de correspondencia que va de acuerdo al tipo de problema a resolver. Para regresión se utilizan funciones de salida lineales y la función de error de suma de cuadrados, para clasificaciones binarias(múltiples independientes) se usa la funcione logística sigmoidal como función de activación de salida para las unidades de la red y la función de entropía cruzada como función de error. Para problemas de clasificación que involucran a dos clases se puede usar una solo función sigmoidal de salida o alternativamente se pueden usar dos salidas utilizando la función softmax como función de activación.

3.2.4. Optimización de parámetros

Ahora se analiza la tarea de hallar al vector de pesos $\mathbf{w}$ que minimiza al función de error elegida $E(\mathbf{w})$. En este punto es útil tener una imagen geométrica de la función de error que puede ser vista como una superficie que reside en el espacio de pesos como se muestra en la figura 3.6. Primero hay que notar que si se toma un pequeño paso en el espacio de pesos desde $\mathbf{w}$ hasta $\mathbf{w} + \sigma\mathbf{w}$ entonces el cambio en la función de error es $\sigma E \simeq \sigma\mathbf{w}^T\nabla E(\mathbf{w})$, donde el vector $\nabla E(\mathbf{w})$ apunta en la dirección de máxima tasa de incremento de la función de error. Debido a que la función de error $E(\mathbf{w})$ es una función suave y continua de $\mathbf{w}$, su valor mas pequeño ocurrirá en un punto en el espacio de pesos tal que el gradiente de la función de error se desvanece, dando como resultado

$$\nabla E(\mathbf{w}) = 0. \tag{3.29}$$

Si 3.29 no se cumple se puede tomar un pequeño paso en la dirección de $-\nabla E(\mathbf{w})$ y de esta manera reducir el error. Los punto en los que el gradiente se hace cero se llaman puntos estacionarios y pueden ser clasificados como mínimos, máximos o puntos de ensilladura.

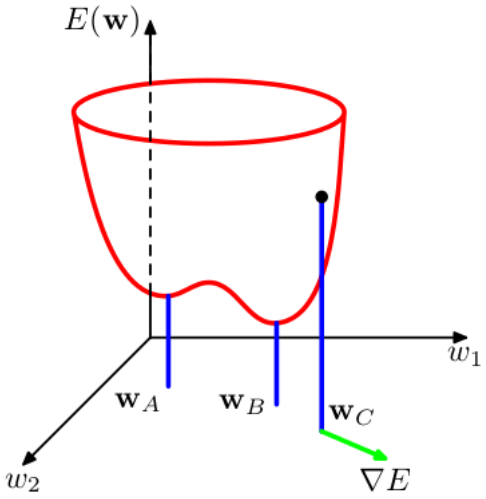


Figura 3.6: Representación geométrica de la función de error $E(\mathbf{w})$ como una superficie que reside en un espacio de pesos tridimensional. El punto $\mathbf{w}_A$ es un mínimo local y el punto $\mathbf{w}_B$ es un mínimo global. A cualquier punto $\mathbf{w}_C$, el gradiente de la función de error esta dado por el vector $\nabla E(\mathbf{w})$.

El objetivo es hallar el vector $\mathbf{w}$ tal que $E(\mathbf{w})$ tome su valor mas pequeño posible, sin embargo la función de error típicamente tiene una alta dependencia no lineal de los parámetros de peso y sesgo y por lo tanto habrán muchos puntos en el espacio de pesos en los que el gradiente se desvanece o es numéricamente muy pequeño. De hecho de la discusión en el sección 3.2.2 se ve que para cualquier punto w que sea un mínimo local habrán mas puntos en el espacio de pesos tal que se comportan como puntos estacionarios mínimos equivalentes; por ejemplo en una red neuronal de dos capas como la que se muestra en la figura 3.5 con M unidades ocultas, cada punto en el espacio de pesos es un miembro de una familia de $M!2^M$ puntos equivalentes.
Aun mas se tiene que típicamente habrán múltiples puntos estacionarios inequivalentes y en particular múltiples mínimos inequivalentes. El mínimo que corresponde al valor mas pequeño de la función de error para cualquier vector de pesos es conocido como mínimo global, cualquier otro mínimo con valores mayores de la función de error son conocidos como mínimos locales. Para una aplicación exitosa de redes neuronales puede que no sea necesario encontrar un mínimo global(en general no se sabrá si un mínimo global ha sido hallado, hay que recordar que el espacio de pesos es un espacio multidimensional) pero puede que resulte necesario comparar varios mínimos locales con el objetivo de hallar una solución suficientemente buena.
Debido a que claramente no hay esperanza de hallar una solución analítica de la ecuaciones $\nabla E(\mathbf{w}) = 0$ se recurre a métodos numéricos iterativos. La optimización de funciones continuas no lineales es un problema ampliamente estudiado y existe una literatura extensa para aprender métodos de solución eficientes. La mayoría de técnicas involucran

escoger un valor inicial $\mathbf{w}^{(0)}$ para el vector de pesos y después moverse en el espacio de pesos en una sucesión de pasos de la forma

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} + \Delta\mathbf{w}^{(\tau)} \tag{3.30}$$

donde τ identifica el paso de iteración. Diferentes algoritmos utilizan distintas elecciones para el vector de actualización de pesos $\Delta\mathbf{w}^{(\tau)}$. Muchos de estos algoritmos usa la información del gradiente y por lo tanto requieren que después de cada actualización se evalué el valor de $\nabla E(\mathbf{w})$ con el nuevo vector de pesos $\mathbf{w}^{(\tau+1)}$. Para entender la importancia de la información del gradiente es útil considerar una aproximación local de la función de error basada en la expansión de Taylor.

Aproximación local cuadrática

Se puede obtener intuición del problema de optimización y de la varias técnicas para resolverlo al considerar una aproximación local cuadrática de la función de error.
Si se considera una expansión de Taylor de $E(\mathbf{w})$ alrededor de un punto $\hat{\mathbf{w}}$ en el espacio de pesos de la siguiente forma

$$E(\mathbf{w}) \simeq E(\hat{\mathbf{w}}) + (\mathbf{w} - \hat{\mathbf{w}})^T\mathbf{b} + \frac{1}{2}(\mathbf{w} - \hat{\mathbf{w}})^T\mathbf{H}(\mathbf{w} - \hat{\mathbf{w}}) \tag{3.31}$$

donde los términos cúbicos y de orden superior han sido ignorados, $\mathbf{b}$ se define como el gradiente de E evaluado en $\hat{\mathbf{w}}$

$$\mathbf{b} \equiv \nabla E|_{\mathbf{w}=\hat{\mathbf{w}}} \tag{3.32}$$

y la matriz Hessiana(*Hessian Matrix*) $\mathbf{H} = \nabla\nabla E$ tiene elementos

$$(\mathbf{H})_{ij} \equiv \frac{\partial E}{\partial w_i \partial w_j}\bigg|_{\mathbf{w}=\hat{\mathbf{w}}}. \tag{3.33}$$

De 3.31, se tiene que la aproximación del gradiente esta dada por

$$\nabla E \simeq \mathbf{b} + \mathbf{H}(\mathbf{w} - \hat{\mathbf{w}}) \tag{3.34}$$

que no es mas que una expansión de Taylor. Para puntos $\mathbf{w}$ que estén suficientemente cerca de $\hat{\mathbf{w}}$ se tiene que las expresiones 3.31 y 3.34 darán aproximaciones razonables para la función de error y su gradiente.
Considerando el caso particular de una aproximación cuadrática local alrededor de un punto $\mathbf{w}^*$ que es un minino de la función de error. En este caso no hay termino lineal ya que $\nabla E = 0$ en $\mathbf{w}^*$ y 3.31 se convierte en

$$E(\mathbf{w}) = E(\mathbf{w}^*) + \frac{1}{2}(\mathbf{w} - \mathbf{w}^*)^T\mathbf{H}(\mathbf{w} - \mathbf{w}^*) \tag{3.35}$$

done el Hessian $\mathbf{H}$ es evaluado en $\mathbf{w}^*$, para tener una interpretación geométrica de esto se considera la ecuación de eigen-valores (valores propios) de la matriz Hessiana

$$\mathbf{H}\mathbf{u}_i = \lambda_i\mathbf{u}_i \tag{3.36}$$

donde los eigen vectores $\mathbf{u}_i$ forman un conjunto ortonormal completo

$$\mathbf{u}_i^T \mathbf{u}_j = \delta_{ij}. \tag{3.37}$$

Ahora se expande $(\mathbf{w} - \mathbf{w}^*)$ como una combinación de los eigen vectores de la forma

$$\mathbf{w} - \mathbf{w}^* = \sum_i \alpha_i \mathbf{u}_i. \tag{3.38}$$

Esto puede interpretarse como una transformación del sistema de coordenadas en el que el origen es trasladado al punto $\mathbf{w}^*$ y los ejes son rotados para coincidir con la alineación de los eigen vectores(a través de la matriz ortogonal cuyas columna son $\mathbf{u}_i$)

$$E(\mathbf{w}) = E(\mathbf{w}^*) + \frac{1}{2} \sum_i \lambda_i \alpha_i^2. \tag{3.39}$$

Se dice que una matriz $\mathbf{H}$ es definida positiva(*positive definite*) si y solo si

$$\mathbf{v}^T \mathbf{H} \mathbf{v} > 0 \quad \text{para todo } \mathbf{v}. \tag{3.40}$$

Debido a que los eigen-vectores $\{\mathbf{u}_i\}$ forman un conjunto completo, un vector arbitrario $\mathbf{v}$ puede escribirse de la siguiente manera

$$\mathbf{v} = \sum_i c_i \mathbf{u}_i. \tag{3.41}$$

De 3.36 y 3.37 se tiene que

$$\mathbf{v}^T \mathbf{H} \mathbf{v} = \sum_i c_i^2 \lambda_i \tag{3.42}$$

entonces $\mathbf{H}$ sera definida positiva si y solo si, todos sus eigen-valores son positivos. En el nuevo sistema de coordenadas cuyos vectores base están dados por los eigen vectores $\{\mathbf{u}_i\}$, los contornos de E constante son elipses centradas en el origen como se ve en la figura 3.7. Para un espacio de pesos unidimensional, un punto estacionario w^* sera un mínimo si

$$\left. \frac{\partial^2 E}{\partial w^2} \right|_{w^*} > 0. \tag{3.43}$$

El resultado correspondiente a D dimensiones es que la matriz Hessiana evaluada en $\mathbf{w}^*$ sea definida positiva.

Uso de la información del gradiente

Como se vera mas a fondo en la sección 3.2.5 es posible evaluar el gradiente de la función de error eficientemente por medio del procedimiento de retro propagación. El uso de esta información del gradiente resulta en mejoras significativas en la velocidad a la cual se localiza el mínimo de la función de error.
En la aproximación cuadrática de la función de error dada en 3.31, la superficie de la función de error esta especificada por las cantidades $\mathbf{b}$ y $\mathbf{H}$ que contiene un total de $W(W+3)/2$ de elementos independientes(debido a que la matriz $\mathbf{H}$ es simétrica), donde W es la

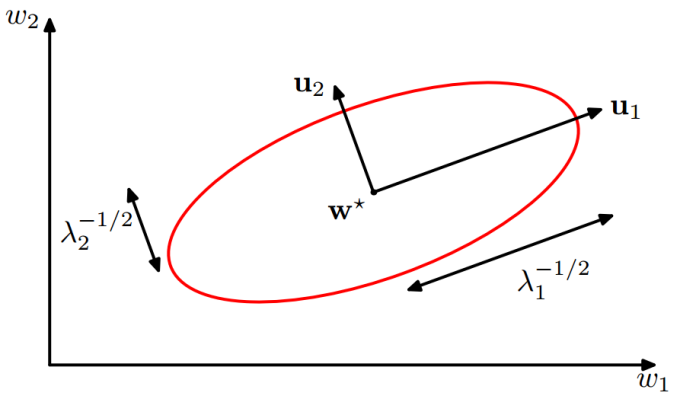


Figura 3.7: En la vecindad de un mínimo $\mathbf{w}^*$ la función de error puede ser aproximada por una expansión de Taylor hasta el termino cuadrático. Los contornos de error constante son elipses cuyos ejes están alineados con los eigen-vectores $\mathbf{u}_i$ de la matriz Hessiana, con longitudes que son inversamente proporcional a la raíz cuadrada de los eigen-vectores

dimensionalidad de $\mathbf{w}$(i.e. el numero total de parámetros configurables de la red neuronal). La locación de los mínimos de esta aproximación cuadrática depende entonces de $O(W^2)$ parámetros y no se debería de esperar localizar un mínimo hasta haber reunido $O(W^2)$ piezas de información independientes. Si no se hace uso de la información del gradiente, se espera realizar $O(W^2)$ evaluaciones funcionales cada una de las cuales requiere $O(W)$ pasos, por lo que el esfuerzo computacional requerido es del orden $O(W^3)$.
Comparando esto con algún algoritmo que utilice la información del gradiente, debido a que la evaluación de ∇E involucra W elementos de información , se espera hallar el mínimo de la función en $O(W)$ evaluaciones del gradiente. Al utilizar retro-propagación de errores se tiene que cada evaluación toma solo $O(W)$ pasos por lo que el mínimo puede ser encontrado en $O(W^2)$ pasos. Por estos motivos el uso de la información del gradiente forma la base de algoritmos pragmáticos para el entrenamiento de redes neuronales.

Descenso del gradiente

El enfoque mas simple para usar la información del gradiente es escoger la actualización de pesos que se da en 3.30 tal que comprenda un pequeño paso en la dirección del negativo del gradiente de tal manera que

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E(\mathbf{w}^{(\tau)}) \tag{3.44}$$

donde el parámetro η se conoce como tasa de aprendizaje. Después de cada actualización el gradiente es revaluado para obtener el nuevo vector de pesos y después se repite el proceso; hay que notar que la función de error esta definida con respecto al conjunto de datos de entrenamiento por lo que cada paso requiere que todo el conjunto de entrenamiento sea re-procesado con el objetivo de calcular ∇E. Las técnicas que utilizan todos los datos de una vez se conocen como métodos de lote(*batch methods*). En cada paso el vector de pesos es movido en la dirección de la máxima tasa de decrecimiento de la función de error, por esto se conoce a este método como descenso del gradiente o el descenso mas

inclinado(*steepest descent*). Aunque parezca que este acercamiento parezca intuitivamente razonable de hecho es un algoritmo poco eficiente como se discute en [2].
Para optimización de lote, hay métodos mas eficientes, tales como el método de gradientes conjugados o el método de cuasi-Newton, que son mucho mas robustos y mas rápidos que el simple método de descenso del gradiente. A diferencia del método de descenso del gradiente estos algoritmos tienen la propiedad que la función de error siempre decrece en cada iteración a menos que el vector de pesos haya llegado a un mínimo local o global.
Para hallar un mínimo de la función de error suficientemente bueno puede que sea necesario correr el algoritmo de descenso de gradiente múltiples veces cada vez con una elección distinta y aleatoria del punto inicial y después comparar el desempeño resultante en un conjunto de datos de validación independiente.

3.2.5. Retro-Propagación de errores(*Error Backprogagation*)

El termino retro-propagación se utiliza en la literatura de redes neuronales para denotar varios conceptos, por ejemplo la arquitectura del perceptron de capas múltiples es a veces llamada red de retro-propagación, el termino también se utiliza para describir el entrenamiento de un perceptron de capas múltiples que usa el descenso del gradiente con una función de error de suma de cuadrados, para aclarar esta terminología es útil considerar la naturaleza del proceso de entrenamiento mas cuidadosamente. La mayoría de algoritmos de entrenamiento involucran un procedimiento iterativo en el que el objetivo es minimizar a una función de error con ajustes realizados al vector de pesos en una secuencia de pasos. En cada paso se pueden distinguir dos etapas, en la primera etapa las derivadas de la función de error con respecto a los pesos son evaluadas. Como se vera a continuación la contribución importante del método de retro-propagación es proveer un método computacionalmente eficiente para evaluar las derivadas mencionadas. Debido a que en esta etapa los errores se propagan en la dirección opuesta a la que se ve en la figura 3.5, en otras palabras en dirección opuesta al procesamiento de la red, se utiliza el termino retro-propagación, específicamente se utiliza para describir la evaluación de la derivadas. En la segunda etapa se utilizan la derivadas para calcular los ajustes que se hacen al vector de pesos. Este método de propagación de errores en sentido contrario de la red para evaluar derivadas puede ser aplicado a funciones de error distintas a la suma de cuadrados y a derivadas de orden superior tales como las matrices Jacobiana y Hessiana.

Evaluación de derivadas de la función de error

La mayoría de funciones de error de interés practico, por ejemplo aquellas definidas por la verosimilitud máxima de conjunto de datos independientes idénticamente distribuidos, comprende una suma de términos uno por cada punto en el conjunto de entrenamiento tal que

$$E(\mathbf{w}) = \sum_{n=1}^{N} E_n(\mathbf{w}). \tag{3.45}$$

Se considera el problema de evaluar $\nabla E_n(\mathbf{w})$ para uno de dichos términos en la función de error, esto puede usarse directamente para una optimización secuencial o los resultados pueden irse acumulando para todo el conjunto de entrenamiento en el caso de métodos

de lote.
Se considera un modelo lineal simple en el que las salidas y_k son combinaciones lineales de las entradas x_i tal que

$$y_k = \sum_i w_{ki} x_i \tag{3.46}$$

que tienen como función de error para una entrada n, de la siguiente manera

$$E_n = \frac{1}{2} \sum_k (y_{nk} - t_{nk})^2 \tag{3.47}$$

donde $y_{nk} = y(\mathbf{x}_n, \mathbf{w})$, el gradiente de esta función de error con respecto a un peso w_{ji} esta dado por

$$\frac{\partial E_n}{\partial w_{ji}} = (y_{nj} - t_{nj}) x_{ni} \tag{3.48}$$

que puede ser interpretado como una computación 'local' que involucra el producto de una 'señal de error' $y_{nj} - t_{nj}$ asociada con el extremo de salida del enlace w_{ji} y la variable x_{ni}.
En una red general de propagación directa(*feed forward network*) cada unidad calcula una suma ponderada de sus entradas como

$$a_j = \sum_i w_{ji} z_i \tag{3.49}$$

donde z_i es la activación de una unidad o una entrada que envía una conexión con la unidad j y w_{ji} es el peso asociado con dicha conexión. En la sección 3.2.2 se vio los parámetros de sesgo pueden ser incluidos en la suma al introducir una unidad extra o entrada extra que se fija en 1. La suma en 3.49 es transformada por una función no lineal de activación $h(\cdot)$ que da la activación z_j para la unidad j de la forma

$$z_j = h(a_j) \tag{3.50}$$

hay que notar que una o mas variables en la expresión 3.49 puede ser una entrada y similarmente la unidad j en 3.50 puede ser una salida.
Considerando ahora la evaluación de la derivada E_n con respecto al peso w_{ji}, las salidas de las unidades va a depender del vector de entrada con sub-indice n, sin embargo no se incluirá la n en las variables de la red con el motivo de tener una notación mas clara. Se nota que E_n depende del peso w_{ji} solo a través de la entrada a_j de la unidad j. Se puede entonces aplicar la regla de la cadena para derivadas parciales para obtener

$$\frac{\partial E_n}{\partial w_{ij}} = \frac{\partial E_n}{\partial a_j} \frac{\partial a_j}{\partial w_{ji}}. \tag{3.51}$$

Se introduce la siguiente notación

$$\delta_j \equiv \frac{\partial E_n}{\partial a_j} \tag{3.52}$$

donde los términos δ son frecuentemente referidos como errores por razones que se harán evidentes. Usando 3.49 se puede escribir

$$\frac{\partial a_j}{\partial w_{ji}} = z_i. \tag{3.53}$$

Substituyendo 3.52 y 3.53 en 3.51 se obtiene

$$\frac{\partial E_n}{\partial w_{ji}} = \delta_j z_i. \tag{3.54}$$

La ecuación 3.54 nos dice que la derivada requerida se obtiene simplemente al multiplicar el valor de δ para la unidad en el extremo de salida del peso por el valor de z para la unidad en el extremo de entrada del peso. Se nota que esto toma la forma simple del modelo lineal discutido al inicio de esta sección. Entonces para evaluar la derivada solo se necesita calcular el valor de δ_j para cada unidad escondida en la red y después aplicar 3.54.
Como se vio anteriormente para las unidades de salida se tiene

$$\delta_k = y_k - t_k. \tag{3.55}$$

para evaluar los errores δ de las unidades ocultas se hace uso de nuevo de la regla de la cadena

$$\delta_j = \frac{\partial E_n}{\partial a_j} = \sum_k \frac{\partial E_n}{\partial a_k}\frac{\partial a_k}{\partial a_j} \tag{3.56}$$

donde la suma se hace para todas las unidades k a las cuales la unidad j hace una conexión, el arreglo de unidades y pesos es ilustrado en la figura 3.8.

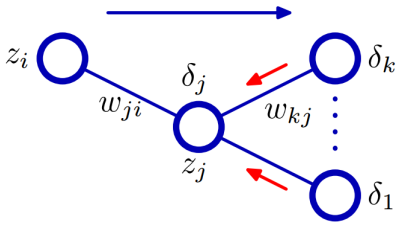


Figura 3.8: Ilustración del calculo de δ_j para unidades ocultas j por medio de la retro-propagación de errores δ de la unidades k a las cuales la unidad j hace conexiones. La flecha azul representa la dirección de propagación de información durante la propagación directa(*feed forward propagation*) y las flechas rojas representan la dirección de retro-propagación de la información de los errores δ.

Al escribir la ecuación 3.56 se hizo uso del hecho que variaciones en a_j dan lugar a variaciones en la función de error solo a través de variaciones de las variables a_k. Si ahora se substituye la definición dada de δ dada por 3.52 en la expresión 3.56 y se hace uso de la formulas 3.49 y 3.50 se obtiene la formula de retro-propagación

$$\delta_j = h'(a_j)\sum_k w_{kj}\delta_k \tag{3.57}$$

que dice que el valor de δ para una unidad escondida en particular se puede obtener al propagar los errores δ en sentido contrario de unidades que están mas adelante en la red, como se ilustra en la figura 3.5. El procedimiento de retro-propagación se puede resumir de la siguiente manera:

1. Alimentar a la red con un vector de entrada y propagarlo directamente usando las expresiones 3.49 y 3.50 para hallar las activaciones de todas las unidades ocultas y de salida

2. Evaluar los errores δ_k para todas las unidades de salida usando 3.55.

3. Retro-propagar los errores δ_k usando 3.57 para obtener δ_j para cada unidad escondida de la red.

4. Usar 3.54 para evaluar las derivadas necesarias.

3.2.6. Redes Convolucionales

Un acercamiento para crear modelos que son invariantes a ciertas transformaciones de las entradas es construir la propiedades de invarianza en la estructura de la red neuronal, esta es la base de las redes neuronales convolucionales. Una propiedad importante de datos de entrada como imágenes o series de tiempo es el hecho que los pixeles o puntos de la serie de tiempo que están cercanos unos con otros muestran una correlación mucho mas alta que lo que harían pixeles o puntos distantes; muchos de los algoritmos modernos utilizados en el reconocimiento computarizado de imágenes(*Computer Vision*) sacan provecho de esta propiedad al extraer características locales que dependen solo de sub-regiones pequeñas de la imagen original. La información de las características extraídas puede ser unida después en etapas posteriores de procesamiento para detectar características de orden superior y ultimadamente dar información de la imagen como un todo. La estructura de una red convolucional se puede ver en la figura 3.9.

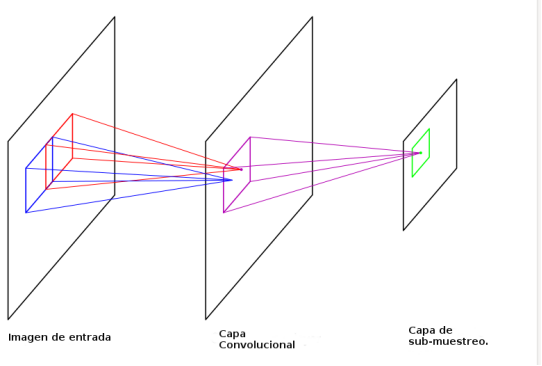


Figura 3.9: Diagrama que ilustra parte de una red convolucional, se muestra una capa convolucional de unidades de activación seguida por una capa de unidades de sub-muestreo. Múltiples pares sucesivos de estas capas pueden ser utilizadas.

En la capa de convolución las unidades están organizadas en planos, cada uno de los cuales se llama mapa de características(*feature map*). Cada una de las unidades en el mapa de características toma entradas de una pequeña sub-región de la imagen y todas la unidades en un mapa de características están restringidas a compartir los mismos valores para los parámetros de peso. Por ejemplo un mapa de características que consista en 10

unidades arregladas en una matriz de 10×10, con cada unidad tomando entradas de una sub-región de 5×5 de la imagen, en este caso el mapa de características en total tendría 25 parámetros de peso ajustables y un parámetro adicional de sesgo. Los valores de entrada provenientes de una sub-región son combinados linealmente utilizando los parámetros de peso y sesgo y el resultado es transformado usando una función no lineal de activación como en la expresión 3.3. Si se consideran a las unidades de un mapa de características como detectores de características entonces se tiene que todas la unidades en el mapa detectan el mismo patrón pero en distintas locaciones de la imagen de entrada.
Las salidas de las unidades de convolución forman las entradas de la capa de sub-muestreo de la red. Por cada mapeo de características en la capa convolucional hay un plano de unidades en la capa de sub-muestreo y cada entrada toma entradas de un campo receptivo relativamente pequeño en el mapa de características correspondiente de la capa de convolución. Las unidades de sub-muestreo realizan una depuración de sus entradas, por ejemplo cada unidad de sub-muestreo toma entradas de una región de 2×2 del mapa de características y calcula el promedio de estos valores multiplicado por el parámetro adaptable de peso y sumado al parámetro adaptable de sesgo y por ultimo transformado utilizando alguna función de activación no lineal.
En una aplicación practica de la arquitectura de redes convolucionales pueden haber varios pares de capas de convolución y sub-muestreo.

Capítulo 4

Metodología de Implementación

4.1. Adquisición de datos

Para la adquisición de los datos se han utilizado los S-files y los Wav-files que conforman la base de datos del software SEISAN. En las siguientes secciones se dan mas detalles de este tipo de archivos. La estrategia es básicamente iterar a través de los S-files que contienen la información de las picadas de fases sísmicas echas por los analistas del INSIVUMEH. De la información de los S-files se identifican las formas de onda que residen el los Wav-files, estas formas de onda son básicamente los sismogramas de los eventos sísmicos. Las amplitudes de los sismogramas que residen en los Wav-files, son las entradas de la red neuronal y los tiempos de llegada de fases se utilizan para generar etiquetas del tipo continuo para cada vector de entrada. Los S-files y Wav-files forman parte del catalogo de eventos sísmicos del INSIVUMEH.

4.1.1. S-files

Los S-files son archivos de texto y son la unidad básica de almacenamiento del software SEISAN, estos contienen información de las lecturas de las fases sísmicas, tiempos de arribo, periodo, azimutal, velocidad aparente, magnitudes del evento, localización de hipocentros, la localización del archivo de forma de onda(Wav file) y otra información importante para el análisis sismológico. Estos archivos vienen en el formato Nordico, para ver la definición de los archivos Nordicos se puede ver el Apendice A de [14]. Para acceder a la información de estos archivos se utilizaron herramientas de la API de Python, *ObsPy* pero también se utilizaron rutinas de acceso de directo a los archivos. Ver la figura 4.1 para ver un ejemplo de un S-file.

4.1.2. Wav-files

Estos son los archivos que contienen las formas de ondas para un evento sísmico. SEISAN soporta varios formatos estos incluyen SEISAN, GSE2.0, SEED/MINISEED, GURALP, Helmberger, binarios SAC y ASCII SAC. Para acceder a la información de estos archivos se utilizaron herramientas de la API de Python *ObsPy* que se describirá mas adelante. Ver la figura para visualizar al archivo Wav.

```
2019 11 4 0613 44.7 L  14.047 -89.798  2.8  GUA  5 0.7 2.3LGUA 3.2CGUA       1
GAP=205        1.62      12.1    30.9 33.8  0.2277E+03 -0.8256E+03 -0.3804E+03E
2019 11 4 0613 44.695  14.04737  -89.79838    2.830  0.674                    H
ACTION:UPD 19-12-16 17:02 OP:laa  STATUS:               ID:20191104061260 L   I
OLDACT:ARG 19-11-12 20:52 OP:seis STATUS:               ID:20191104061260 L   3
2019-11-04T06:13:00.000000Z.MSEED                                             6
STAT SP IPHASW D HRMM SECON CODA AMPLIT PERI AZIMU VELO AIN AR TRES W  DIS CAZ7
NUBE HZ IP        61348.605   61                          97   -0.0810 16.2 173
NUBE HE ES   1    61351.380                               97   -0.42 7 16.2 173
JUAM EZ EP        61351.437                               56   -0.1610 33.4  17
JUAM EN IS   1    61356.531                               56   -0.45 7 33.4  17
JUAM EN  IAML     61356.791       404.2 0.22                        33.4  17
MTO3 HZ EP        61356.369   29                          56   -0.4510 61.2  50
CHIE HZ IP        61358.730                               47    0.1110 75.1  41
CHIE HN ES   1    614 8.707                               47   -0.78 7 75.1  41
SARH EZ EP        614 9.256                               47    1.5210  138  54
```

Figura 4.1: Ejemplo de un S-file

4.1.3. Datos

Como se menciono anteriormente se han utilizado a los Wav-files que contienen la información de las amplitudes de los eventos sísmicos. Estas amplitudes se utilizan para crear el vector de entrada de la red. Para crear las etiquetas se han utilizado los S-files que contienen la información del tiempo de llegada de las fases P y S. Después de haber accedido a la información usando herramientas de *ObsPy* se usan listas de Python para tabular los datos y después se utilizan a las API'S de Python: *pickle* y *numpy*, para guardar los datos. Estos datos se utilizan después para entrenar y validar el modelo de la red neuronal. Los datos consisten entonces de un archivo S-file por cada evento sísmico y un archivo Wav-file que contiene las formas de onda para dicho evento. En los Wav-files se tienen varias formas de onda, cada una corresponde a una estación sísmica y a un canal(eje de movimiento): Vertical, Norte-Sur, Este-Oeste.

Datos de entrenamiento

Los datos de entrenamiento son datos del catalogo de eventos sísmicos del INSIVUMEH correspondientes al año 2019, se utilizaron eventos para los cuales existen picadas de fases sísmicas hechas por los analistas del departamento de Geofísica del INSIVUMEH. A continuación se presenta el formato original de los sismogramas del catalogo utilizado para que se pueda apreciar la naturaleza de los datos con los que se entrenó a la red neuronal.

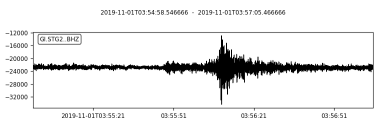


Figura 4.2: Forma de onda en su versión original.

4.1.4. Colección de etiquetas

Las etiquetas fueron adquiridas de los S-files debido a que en estos archivos se encuentran los tiempos de arribo de cada fase sísmica para cada evento. Este tiempo de arribo se convierte a un numero entero entre $1 - 3000$ la elección de este numero se hace debido a que las entradas de la red neuronal se han fijado en 3000, de esta manera se puede identificar la posición relativa de la llegada con respecto al vector de entrada. Después se convierte en numero real entre $0 - 1$ ya que se utilizan entradas y etiquetas normalizadas con fines de convergencia.

4.1.5. Método de adquisición

El método de adquisición de datos consiste en tomar todos los S-files que están alojados en una carpeta y hacer una lista de estos para después iterar a través de todos los S-files en dicha carpeta. Después se lee al archivo con la API de Python, *ObsPy* utilizando la rutina, *read_event*

```
from obspy import read_events
Sfile_events=read_events(Sfiles_path+Sfile_name,format="NORDIC")
```

esta rutina se utiliza para leer los eventos sísmicos de un S-file. Se tuvo que corregir un bug de esta rutina que leía incorrectamente los segundos de los eventos sísmicos, en el código correspondiente se dan mas detalles. Esta rutina se usa para leer la metadata de los eventos, la metadata es manejada en una jerarquía de clases que esta modelada en base al estándar de facto en sismología, *QuakeML*. En la siguiente figura se puede ver un diagrama de la jerarquía de clases.

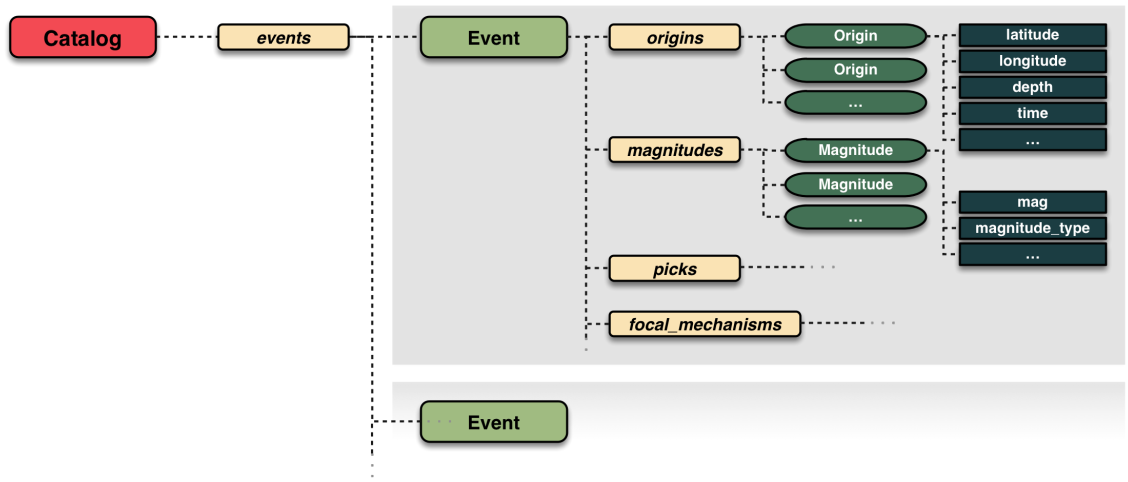


Figura 4.3: Jerarquía de clases manejada por *ObsPy*

Después que se tiene la información de las picadas, se identifica al Wav-file que contiene las formas de onda correspondientes al S-file, y se utliza la rutina de *ObsPy*, *read*

```
from obspy import read as readStream
Wav_stream=readStream(Wfiles_path+Wfile_name)
```

que devuelve un objeto del tipo *Stream*. Los *Stream* son objetos del tipo lista que contienen múltiples objetos del tipo *Trace* que contienen las amplitudes del evento sísmico en una serie de tiempo, headers y meta-data correspondiente. En la siguiente figura se puede ver la estructura de estos objetos.

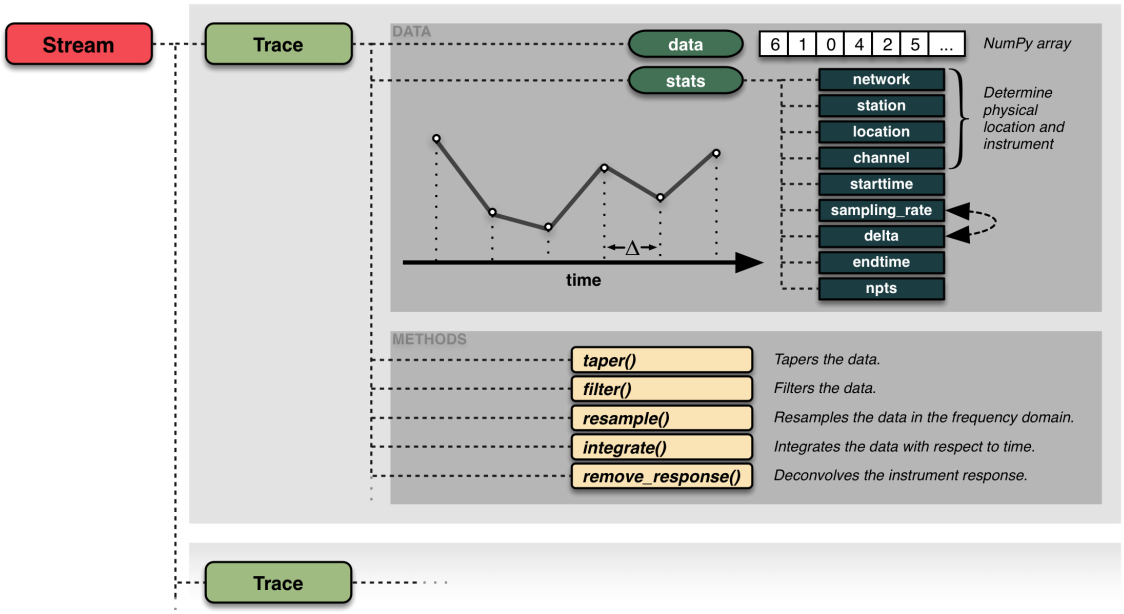


Figura 4.4: Estructura de objeto *Stream*

4.1.6. Algoritmo de adquisición

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Thu Feb 20 11:41:34 2020

@author: jmedina
"""


#Programa para adquisicion de datos de Sfiles y Wav files
#Para cada picada en el S-file se buscan los canales disponible y  se crea una muestra que consiste en
      la traza y los tiempos de picada S y P
#Se utlizan solo picadas para las cuales exista fase P y fase S para la estacion en el S file
#al final se guardan la lista resultante np_training_dataX y np_training_dataY en  archivos llamados
      training_dataX & training_dataY
#Version actualizada de NN_data_acquisition
#se salva la lista con la estructura de cada picada para recoleecion de resultados

from obspy import read as readStream#objeto para leer formas de onda, WAV files
#from obspy.core.event import read_events
from obspy import read_events#objeto para leer Eventos
import matplotlib.pyplot as plotTrace
import os #  objeto para crear lista de archivos en una carpeta
import math
#import tensorflow as tf
import numpy as np #compuracion cientifica
import pickle#objeyo para guardar archivos de objetos en txt
```

```
Sfiles_path='/home/jmedina/seismo/REA/GUA__/2019/11/'#'/home/jmedina/seismo/REA/ENJ__
    /2019/04/16-1456-38L.S201904'#/home/jmedina/seismo/REA/GUA__/2019/10/"#Directorio de S-files
Wfiles_path='/home/jmedina/seismo/WAV/GUA__/2019/11/'#"/home/jmedina/seismo/WAV/GUA_save/"#Directorio
    de Wav files
outputX_file='/home/jmedina/Documents/NeuralNetworks/data/training_dataX_11'#archivo para guardar datos
     de entrenamiento de entrada
outputY_file='/home/jmedina/Documents/NeuralNetworks/data/training_dataY_11'#archivo para guardar datos
    de entrenamiento de salida
component_training_dataY_file_path='/home/jmedina/Documents/NeuralNetworks/data/
    component_training_dataY_11.txt'#archivo que guarda meta-data del conjunto de entrenamiento

Sfiles_names=os.listdir(Sfiles_path)#se genera una lista de los archivos en el directorio
training_dataX=[]
training_dataY=[]
print(len(Sfiles_names))
event_id=0
for Sfile_name in Sfiles_names:#se itera a traves de los Sfiles
    if (Sfile_name.find('.out')==-1)and(Sfile_name.find('.mes')==-1)and(Sfile_name.find('.inp')==-1)and
        (Sfile_name.find('.wav')==-1)and (Sfile_name.find('CCIG')==-1):
        #print(Sfile_name)
        Sfile_lines=[]
        try:
            Sfile_events=read_events(Sfiles_path+Sfile_name,format="NORDIC")#se leen los eventos del S
                file
            f=open(Sfiles_path+Sfile_name,"r")#,encoding="ISO-8859-1")
            Sfile_lines=f.readlines()#lineas S file
            f.close()
        except:
            print('Problem reading S-file: '+Sfile_name)
        # print(Sfile_events[0])

        #try:
        # f=open(Sfiles_path+Sfile_name,"r")#,encoding="ISO-8859-1")
        # Sfile_lines=f.readlines()#lineas S file
        # f.close()
        #except:
            #print('Problem reading S-file: '+Sfile_name)
        if len(Sfile_lines):
            i=0
            for l in Sfile_lines:#se itera a traves de las lineas del archivo
                i+=1
                if l[len(l)-2]=="6":#se busca en las lineas de Sfile el identificador 6 que indica el
                    nombre del archivo WAV
                    Wfile_name=l[0:len(l)-3].strip()# nombre del WAV file, funcion strup quita espacios
                        en blanco
                if l[len(l)-2]=='7':# se busca en las lineas del S file el identificador 7 que indica
                    que empieza la lista de llegadas de fase
                    event_line_start=i
                    break
            pick_lines=Sfile_lines[event_line_start:len(Sfile_lines)-1]#lineas del Sfile con
                informacion de las picadas

            #Se arregla bug de obspy, no se leian correctamente los tiempos de llegada de las fases en
                el archivo NORDIC
            #El bug se daba ya que los segundos en los Sfiles del INSIVUMEH vienen con 3 cifras
                decimales, lo que causa problemas
            #con el comando read_events()
            i=0
            if len(Sfile_events):
                for pick in Sfile_events[0].picks:# se itera a traves de las picadas para el evento
                    leido por obspy
                    for pick_line in pick_lines:
                        seconds=pick_line[22:28]#posicion en la que estan los segundos

                        if (pick.waveform_id.station_code in pick_line)and (pick.waveform_id.
                            channel_code in pick_line):

                            #Sfile_events[0].picks[i].time.second=int(float(seconds))
                            Sfile_events[0].picks[i].time._set_second(int(float(seconds)))
                            break
                        # print(seconds)
                        # print(Sfile_events[0].picks[i].time.second)
                    i+=1

                try:
                    Wav_stream=readStream(Wfiles_path+Wfile_name)#se lee el archivo Wav correspondiente
                        al S-file
                    wav_found=1
                except:
                    print('Wav_File: '+Wfile_name+'_not_found_corresponding_to_Sfile: '+Sfile_name)
                    wav_found=0
                    #print()
                for pick in Sfile_events[0].picks:#se itera a traves de las picadas del evento
                    #if (pick.phase_hint=='S'):#and(pick.waveform_id.station_code=='ZAFR2'):
                    #    print(pick)
                    num_stations=0
                    stations=[]
                    if wav_found:
                        for wave_form in Wav_stream:
                            #se itera a traves de las formas de onda del evento
                            if ([wave_form.stats.station,wave_form.stats.channel] not in stations)and(
                                pick.waveform_id.station_code==wave_form.stats.station)and(wave_form.
                                stats.npts>=3000)and(pick.phase_hint in ['P','S']):#and(pick.
                                waveform_id.channel_code in wave_form.stats.channel):#and(wave_form.
```

```
                                    stats.npts>3000):#and(pick.waveform_id.station_code==wave_form.stats.
                                    station):#and (pick.waveform_id.phase_hint in ['P','S']):
                                num_stations+=1
                                aux=0
                                #aux_wave=wave_form
                                while (wave_form.stats.npts!=3000):#se asegura que el numero de datos
                                     sea 3000
                                    wave_form.resample(wave_form.stats.sampling_rate/(wave_form.stats.
                                         npts/(3000+aux)))
                                    aux+=0.1
                                #se normalizan los datos como, (X-mean)/(max-min) convencion de machine
                                     learning
                                normalized_data=(wave_form.data-wave_form.data.mean())/(wave_form.data.
                                    max()-wave_form.data.min())# se normalizan los datos para que el
                                     entrenamiento convega
                                training_dataX.append(normalized_data)
                                #datos del entrenamiento
                                #se crea la distribucion  de probabilidad
                                probability_dist=[]
                                #if pick.waveform_id.channel_code in wave_form.stats.channel:
                                t_phase_arrival=int(3000*(pick.time-wave_form.stats.starttime)/(
                                    wave_form.stats.endtime-wave_form.stats.starttime))
                                sigma=3000/(wave_form.stats.endtime-wave_form.stats.starttime)
                                for t in range(3000):
                                    probability_dist.append((1/(sigma*(2*math.pi)**0.5))*math.exp
                                        (-0.5*((t-t_phase_arrival)/sigma)**2.0))
                                #new_pick recibe una nueva muestra para los datos training_dataY
                                #new_pick consiste en :
                                 # 0.tipo de fase
                                 # 1.hora de llegada  de fase
                                 # 2.tiempo de llegada transformado
                                 # 3.id de forma de onda,
                                 # 4.tiempo de incio de traza
                                 # 5.tiempo de finalizacion de traza
                                 # 6.id estacion,
                                 # 7.id canal
                                 # 8.distribucion de probabilidad,
                                 # 9.identificador del evento

                                new_pick=[pick.phase_hint,pick.time,t_phase_arrival,pick.waveform_id,
                                    wave_form.stats.starttime,wave_form.stats.endtime,wave_form.stats.
                                    station, wave_form.stats.channel,probability_dist, event_id]
                                training_dataY.append(new_pick)#datos de la picada identificada
                                stations.append([wave_form.stats.station,wave_form.stats.channel])
                                #break
                                    #aux+=0.1
                            if num_stations==3:#si se encontraron 3 canales se termiana el ciclo
                                break
                event_id+=1

#rutina para crear conjunto de datos que tienen  picadas P y S, con el fin de crear un set de datos
     para el modelo a entrenar

stations=[]
events=[]
component_training_dataX=[]
component_training_dataY=[]
reduced_training_dataY=[]

for i in range(len(training_dataY)):
    for j in range(len(training_dataY)):
        if (training_dataY[i][9]==training_dataY[j][9]) and (training_dataY[i][6]==training_dataY[j
            ][6]) and (training_dataY[i][0]!=training_dataY[j][0]):
            if ([training_dataY[i][9],training_dataY[i][6],training_dataY[i][7]]not in stations):
                component_training_dataX.append(training_dataX[i])
                t_phase_arrival=int(3000*(training_dataY[j][1].timestamp-training_dataY[i][4].timestamp
                    )/(training_dataY[i][5].timestamp-training_dataY[i][4].timestamp))

                component_training_dataY.append([[training_dataY[i][0],training_dataY[i][1],
                    training_dataY[i][2],training_dataY[i][4],training_dataY[i][5],training_dataY[i
                    ][6],training_dataY[i][7],training_dataY[i][8]],[training_dataY[j][0],
                    training_dataY[j][1],training_dataY[j][2],training_dataY[j][4],training_dataY[j
                    ][5],training_dataY[j][6],training_dataY[j][7],training_dataY[j][8]]])

                reduced_training_dataY.append([training_dataY[i][2]/3000,t_phase_arrival/3000])
                stations.append([training_dataY[i][9],training_dataY[i][6],training_dataY[i][7]])


            if ([training_dataY[j][9],training_dataY[j][6],training_dataY[j][7]]not in stations):
                component_training_dataX.append(training_dataX[j])
                t_phase_arrival=int(3000*(training_dataY[i][1].timestamp-training_dataY[j][4].timestamp
                    )/(training_dataY[j][5].timestamp-training_dataY[j][4].timestamp))

                component_training_dataY.append([[training_dataY[i][0],training_dataY[i][1],
                    t_phase_arrival,training_dataY[i][4],training_dataY[i][5],training_dataY[i][6],
                    training_dataY[i][7],training_dataY[i][8]],[training_dataY[j][0],training_dataY[j
                    ][1],training_dataY[j][2],training_dataY[j][4],training_dataY[j][5],training_dataY
                    [j][6],training_dataY[j][7],training_dataY[j][8]]])

                reduced_training_dataY.append([t_phase_arrival/3000,training_dataY[j][2]/3000])
                stations.append([training_dataY[j][9],training_dataY[j][6],training_dataY[j][7]])
            # if training_dataY[i][9]!=training_dataY[j][9]:
            #     break
```

```
#rutina para asegura que la fase P aparece primero que la fase S
for i in range(len(component_training_dataY)):
    if component_training_dataY[i][0][0]!='P':
        aux=component_training_dataY[i][0][0]
        aux2=reduced_training_dataY[i][0]
        component_training_dataY[i][0][0]=component_training_dataY[i][1][0]
        reduced_training_dataY[i][0]=reduced_training_dataY[i][1]
        component_training_dataY[i][1][0]=aux
        reduced_training_dataY[i][1]=aux2


component_training_dataY_file=open(component_training_dataY_file_path,'wb')
pickle.dump(component_training_dataY,component_training_dataY_file)
component_training_dataY_file.close()

np_training_dataX=np.array(component_training_dataX)
np_training_dataY=np.array(reduced_training_dataY)

print(np_training_dataX.shape)
print(np_training_dataY.shape)
np.save(outputX_file,np_training_dataX)
np.save(outputY_file,np_training_dataY)
```

4.2. Pre-Procesamiento

En esta etapa de pre-procesamiento, se formatean los datos de tal manera que cada muestra tenga dimensiones uniformes y que los datos sean transformados para que el entrenamiento del modelo sea mas veloz. Esta etapa es fundamental tanto para determinar el diseño de la arquitectura a implementar como para familiarizarse con la estructura de los datos. En esta etapa también se puede crear rutinas para la visualización de los datos con el fin de la extracción de características que puedan mejorar el desempeño del modelo. También se pueden invertir recursos en analizar la calidad relativa de los datos, con el fin de depurar, aumentar o limpiar el conjunto de datos original para crear un conjunto de datos de entrenamiento y de prueba.

4.2.1. Limpieza de datos

El primer factor que se utilizó para discriminar muestras del conjunto original de datos es que solo se toma en cuenta formas de ondas para las cuales existe una picada de fase registrada en el S-file del evento, ya que se pretende tener un modelo de aprendizaje supervisado, ademas de esto se verificó que por cada picada de fase considerada existiera también una picada de fase adicional. Por ejemplo si se encontró una picada de fase P para la estación 'STA01' se verifico que también hubiese una picada de fase S para esa estación del mismo evento sísmico. Esta discriminación se llevo a cabo ya que cada muestra con que se entrena al modelo debe de tener dos etiquetas, una que identifica a la fase P y la otra para la fase S. Por ultimo solo se tomaron formas de onda para las que existieran mas de $N = 3000$ puntos en la traza, esto por razones de remuestro de los datos.

4.2.2. Aumentación de datos

Para aumentar los datos se considero que las picadas de fase se hacen por lo regular en un canal especifico para cada estación, por ejemplo las picadas de fase P se hacen por lo regular en el canal vertical que comúnmente se le llama Z, esto se hace por la naturaleza de las ondas primarias P que son ondas compresionales. La aumentación de datos consistió en asignar para cada canal de cada estación que tuviera identificadas fases P y S, las etiquetas de cada fase. Por ejemplo si en el canal N(Norte-Sur) de la estación

'STA01' no hay picadas pero para los 2 otros canales(E, Z) de la misma estación si existen esta información entonces se le asignan las picadas de los otros canales.

4.2.3. Transformación de datos

Como se ha mencionado anteriormente es necesario realizar algunas transformaciones al estado original de los datos tratando que al hacerlo no se pierda información que perjudique la correcta caracterización que el modelo debe realizar en los datos. Esta transformaciones se hacen con el fin que modelo tenga un mejor desempeño, es una practica estándar en la literatura del aprendizaje automático realizar una normalización de los datos. En este caso de eventos sísmicos es crucial transformar a las amplitudes de los sismogramas ya que estas difieren mucho entre distintas magnitudes de eventos sísmicos, lo que se busca es que modelo logre aprender la representación de cada fase independientemente de su magnitud.

Normalización y regularización de datos

Como se ha mencionado varias veces la normalización y regularización de los datos son necesarias para obtener un buen desempeño en el modelo seleccionado. La regularización de los datos consiste en re-muestrear los datos de tal manera que todas las muestras tengan una dimensionalidad constante, para esta implementación se ha escogido que la dimension sea de $N = 3000$, esto se hizo inspirándose en el articulo de Zhu & Berroza, ver [6]. En el proceso de normalización se utilizaron practicas comunes del aprendizaje automático, específicamente se utilizo la normalización de media

$$x'_i = \frac{x_i - \mu(\mathbf{x})}{\max(\mathbf{x}) - \min(\mathbf{x})} \tag{4.1}$$

donde x'_i es el componente transformado, x_i es el componente a transformar, $\mu(\mathbf{x})$ es el promedio de los componentes x_i del vector $\mathbf{x}$, $\max(\mathbf{x})$ es el valor máximo de los componentes de $\mathbf{x}$ y $\min(\mathbf{x})$ es el mínimo. Para visualizar la naturaleza de los datos y el efecto que tiene la normalización y regularización se presentan los siguientes gráficos.

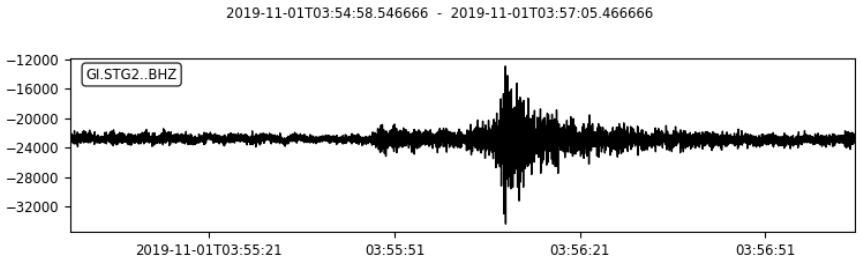


Figura 4.5: Forma de onda en su version original.

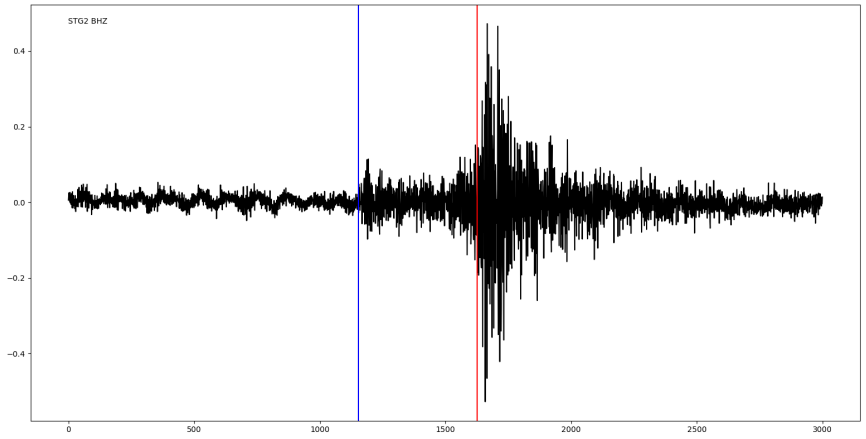


Figura 4.6: Forma de onda con datos normalizados y regularizados. En la figura se pueden apreciar una linea vertical de color azul que corresponde a la picada de la fase P y la linea vertical roja corresponde a la llegada de fase S. Las llegadas de fase son las etiquetas colectadas para cada muestra.

4.3. Entrenamiento del modelo

Para entrenar el modelo se han hecho varios conjuntos de datos que se han agrupado por mes y se han hecho conjuntos de varios meses para incrementar el numero de muestras para tener un mejor comportamiento del modelo. De la naturaleza de los datos y de implementaciones previas de aprendizaje automático en sismología(ver [3], [6]) se han considerado varios modelos, como una primera implementación se han considerado aplicar arquitecturas de regresión.

4.3.1. Selección del modelo

Para la selección del modelo se considero que las entradas se den por medio de un vector multidimensional de $N = 3000$ elementos que corresponde a las amplitudes registradas para un canal especifico de cada estación que vio el evento sísmico. Las salidas del modelo son las picadas de fase, los tiempos de llegada de fase son transformados utilizando el sello de tiempo correspondiente a un numero entre $1 - 3000$ que después se mapea a un numero entre $0 - 1$, como se puede ver en la figura 4.6. Como una implementación que pueda dar perspectiva de la construcción de modelos mas complejos y sofisticados se han considerado utilizar modelos de redes neuronales de propagación directa es decir el modelo básico de una red neuronal(ver 3.2.2). Como una implementación adicional se ha construido un modelo que incluye capas de convolución, (ver 3.2.6), se ha notado que estas capas de convolución le dan una mejora de desempeño en comparación con el modelo básico.

Red Neuronal Profunda(Deep Neural Network)

Una red neuronal profunda que también es conocida como perceptron de capas múltiples, se le llama profunda ya que se tienen varias capas de unidades ocultas para el procesamiento de las entradas de la red, la arquitectura de la red puede apreciarse en la figura 3.5 con la diferencia que se tienen mas capas. En especifico se han construido modelos que tienen hasta 9 capas de procesamiento. Se ha escogido como función de activación de las unidades ocultas a la función unidad lineal rectificada(*rectified linear unit*), ReLU.

$$f(x) = x^{+} = \max(0, x) \tag{4.2}$$

esta función devuelve 0 si $x < 0$ de lo contrario devuelve x, para referencia ver la siguiente figura. Se ha escogido esta función de activación debido a que la convergencia a la hora de optimizar los parámetros de la red, es mucho mas veloz que en comparación con funciones como tanh o la función logística sigmoidal, ver [7]. La red neuronal profunda(DNN) tiene

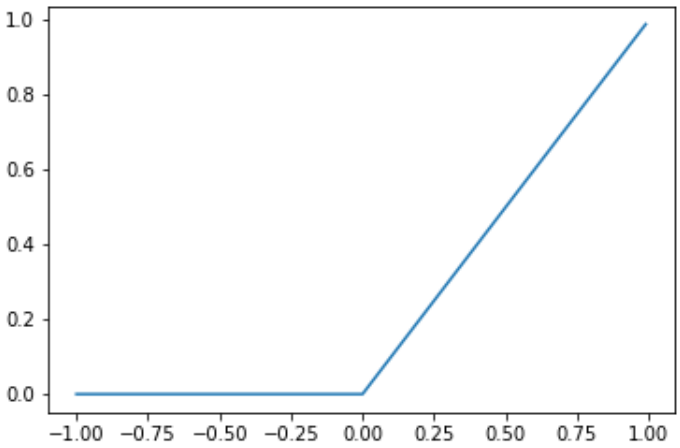


Figura 4.7: Grafica de la función ReLU

como función de error la función suma de cuadrados dada por 3.17. Para la optimización de los parámetros internos de la red se ha usado el algoritmo de estimación del momento adaptable(*Adam, Adaptive Moment Estimation*) que es una variante a una serie de algoritmos conocidos como descenso del gradiente estocástico.

Red Neuronal Convolucional(Covolutional Neural Network)

Una red neuronal convolucional tiene capas de procesamiento extras que después de procesarse son alimentadas a una red neuronal profunda. Las capas adicionales de procesamiento realizan transformaciones matriciales a los datos en su forma original, estas transformaciones están diseñadas de tal manera que se extraigan características propias del conjunto de datos de entrenamiento. Para ver un ejemplo de como funciona el proceso de convolución se pueden ver las siguientes imágenes.

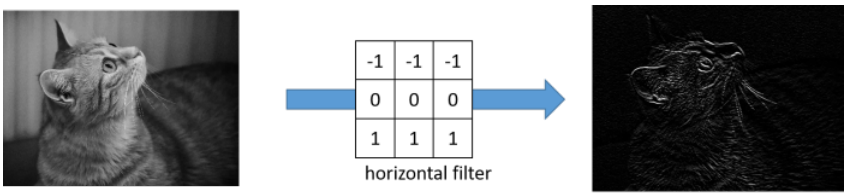


Figura 4.8: Efecto que tiene una capa de convolución que esta diseñada para extraer características de los bordes horizontales de una imagen.

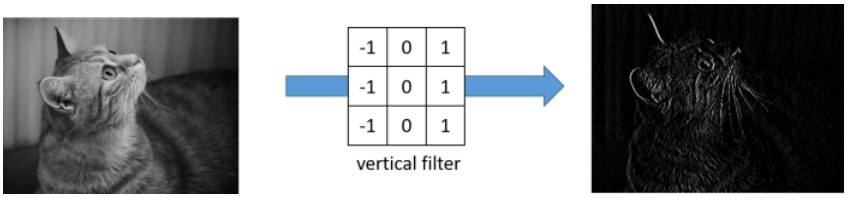


Figura 4.9: Efecto que tiene una capa de convolución que esta diseñada para extraer características de los bordes verticales de una imagen.

La función de activación de las unidades ocultas utilizada en el modelo de red convolucional, es la función ReLU que se ve en la figura 4.7. Se han creado dos redes convolucionales profundas de 7 capas de procesamiento, las salidas son dos valores continuos con los que se trata de predecir el tiempo de llegada de las fases sísmicas.

Capítulo 5

Resultados

Se han evaluado dos modelos y se han usado distintos conjuntos de datos de entrenamiento. Las arquitecturas de redes neuronales usadas para crear los modelos son la red neuronal profunda(*DNN, Deep Neural Network*) y la red neuronal convolucional(*CNN, Convolutional Neural Network*). Se han creado varios conjuntos de datos de resultados. Se han utilizado distintos meses y se han evaluado en los modelos entrenados. Se han considerado como coincidencias a los casos en que la diferencia entre las predicciones realizadas por el modelo y las etiquetas de fases identificadas manualmente tienen una diferencia de menos de un segundo. Estos resultados han sido agrupados en casos sin coincidencias, casos con coincidencia para una fase sísmica y casos en los que se tiene coincidencias para las dos fases sísmicas. Para poder visualizar los resultados se han graficado los datos de entrada normalizados y en la misma gráficas se han identificado a las llegadas de fase con lineas verticales, las llegadas de fase P realizadas por el modelo y las que formas parte del conjunto de datos de prueba se aprecian en color azul y las llegadas de fase S se aprecian en color rojo. En las gráficas también se pueden ver las diferencias en segundos entre las predicciones del modelo y los datos de prueba. A continuación se ve en las figuras los resultados para la red neuronal profunda.

5.1. DNN

Para el modelo de red neuronal profunda entrenado con datos del mes de noviembre 2019 se tienen los siguientes resultados que corresponden a datos de prueba del mes de julio 2019, se han evaluado 5098 muestras. A continuación se muestran tablas con los resultados, se han separado casos en los que se tienen menos de 1 segundo de diferencia con respecto al catalogo de prueba, menos de 2 segundos de diferencia y menos de 3 segundos, también se ha indicado que fase fue identificada(P o S) y se ha hecho un conteo de para cuantos casos se han identificado a las dos fases o solo una.

Resultados menores a 1 segundo

Fases P identificadas	148
Fases S identificadas	142
1 fase identificada	265
2 fases identificadas	25

Resultados menores a 2 segundos

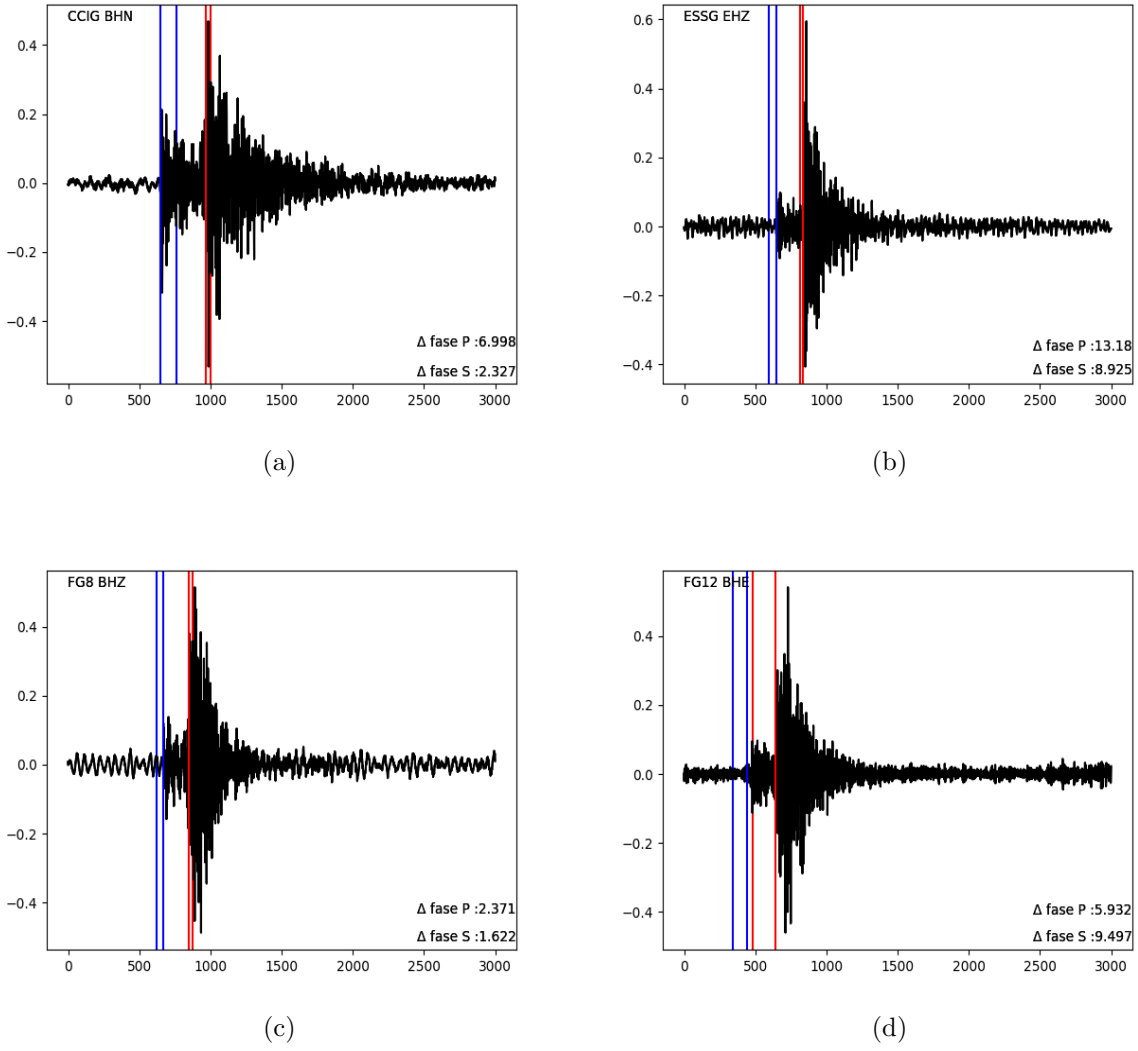
Fases P identificadas	311
Fases S identificadas	282
1 fase identificada	510
2 fases identificadas	83

Resultados menores a 3 segundos

Fases P identificadas	453
Fases S identificadas	439
1 fase identificada	719
2 fases identificadas	173

A continuación se pueden apreciar 4 casos para cada criterio de coincidencias.

Sin coincidencias

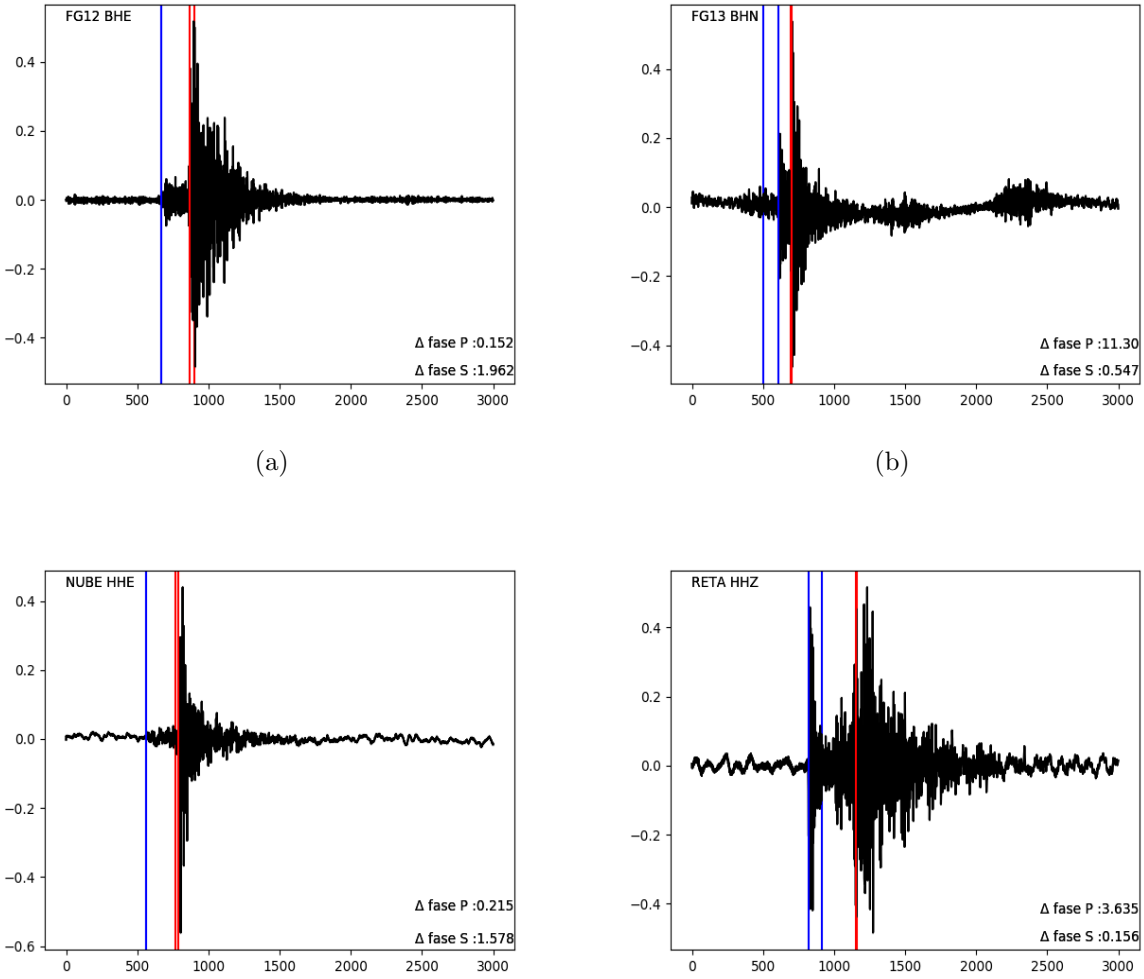


(a) (b)

(c) (d)

Figura 5.1: En la figura se pueden ver 4 formas de onda para las cuales el modelo de red neuronal profunda dio predicciones que difieren por mas de 1 segundo de las dos picadas de fase realizadas por los analistas del INSIVUMEH. Las picadas de fase P se pueden apreciar en azul y las picadas de fase S en rojo. Las diferencias en segundos de la predicción hecha por el modelo y los datos del catalogo de prueba se pueden en la parte inferior derecha de las trazas.

Coincidencias de 1 fase

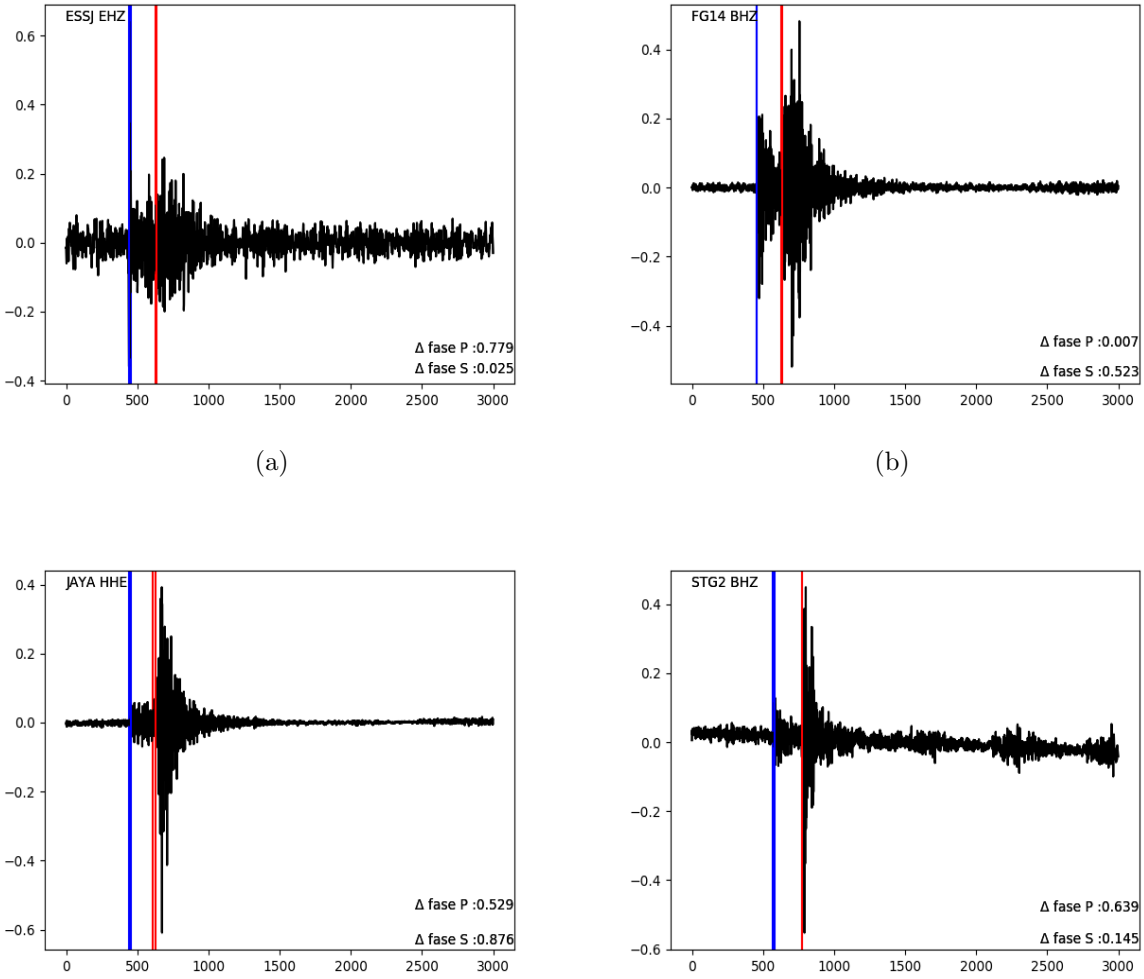


(a) (b)

(c) (d)

Figura 5.2: En la figura se pueden ver 4 formas de onda para las cuales el modelo de red neuronal profunda dio alguna predicción que difieren por mas de 1 segundo y una predicción que coincide(diferencia de menos de 1 segundo) con la picadas de fase realizadas por los analistas del INSIVUMEH. Las picadas de fase P se pueden apreciar en azul y las picadas de fase S en rojo. Las diferencias en segundos de la predicción hecha por el modelo y los datos del catalogo de prueba se pueden en la parte inferior derecha de las trazas.

Coincidencias de 2 fases



(a) (b)

(c) (d)

Figura 5.3: Para los cuatro casos que se ven en la figura se tienen coincidencias con las predicciones del modelo y los datos del conjunto de prueba. En estos casos es difícil ver gráficamente la diferencia entre la predicción y los datos del catalogo ya que las diferencias son de menos de 1 segundo. En la parte inferior derecha se puede la diferencias en segundos de las predicciones y los datos de prueba. Las picadas de fase P se ven en color azul y las picadas de fase S se ven en color rojo.

5.2. CNN

Se han entrenado dos modelos de redes neuronales convolucionales, en el primer modelo se han utilizado datos del mes de noviembre 2019 y para el segundo modelo se han utilizado datos de los meses de julio 2019 a noviembre 2019.

5.2.1. Primer modelo

El primer modelo se entreno con datos de noviembre 2019. Los siguientes resultados corresponden a datos de prueba del mes de julio 2019, se han evaluado 5098 muestras. A continuación se muestran tablas con los resultados, se han separado casos en los que se tienen menos de 1 segundo de diferencia con respecto al catalogo de prueba, menos de 2 segundos de diferencia y menos de 3 segundos, también se ha indicado que fase fue identificada(P o S) y se ha hecho un conteo de para cuantos casos se han identificado a las dos fases o solo una.

Resultados menores a 1 segundo

Fases P identificadas	528
Fases S identificadas	1108
1 fase identificada	1428
2 fases identificadas	208

Resultados menores a 2 segundos

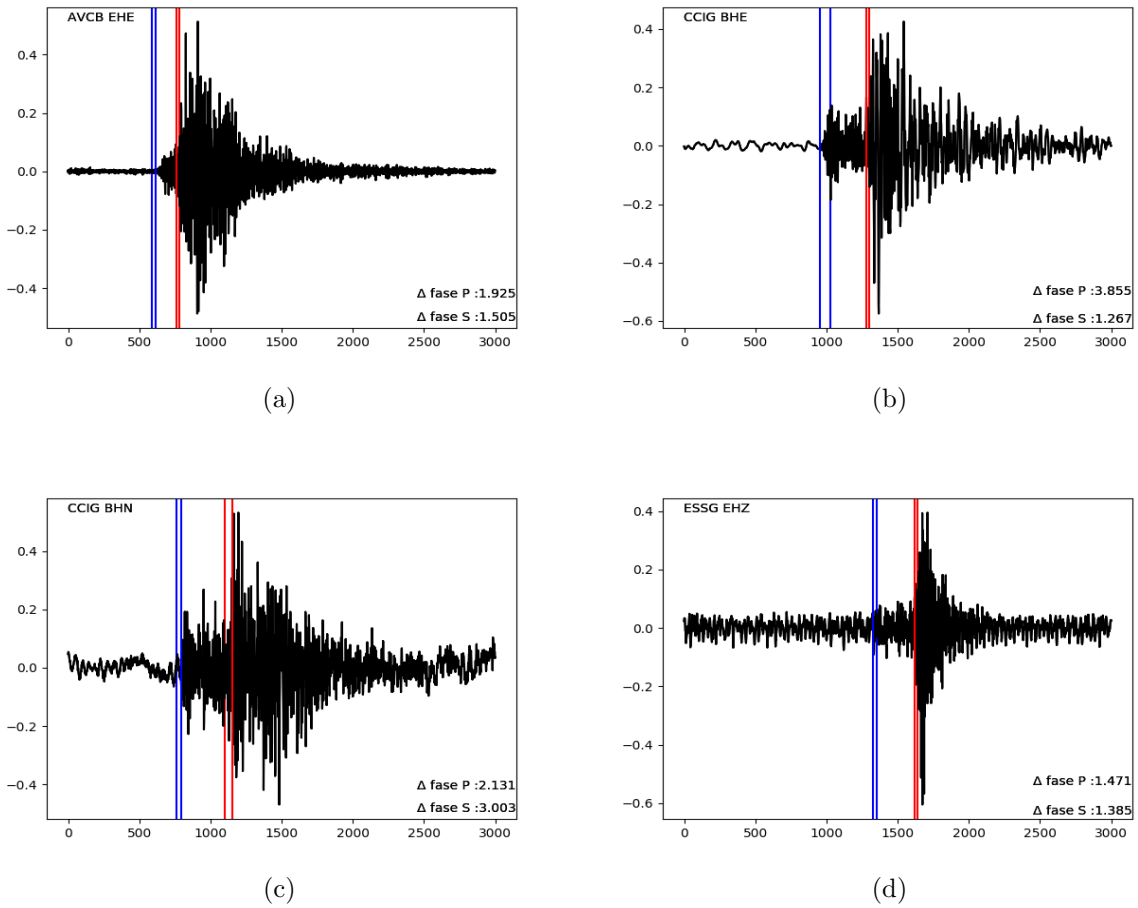
Fases P identificadas	964
Fases S identificadas	1980
1 fase identificada	2356
2 fases identificadas	588

Resultados menores a 3 segundos

Fases P identificadas	1406
Fases S identificadas	2581
1 fase identificada	2935
2 fases identificadas	1052

A continuación se pueden apreciar 4 casos para cada criterio de coincidencias.

Sin coincidencias

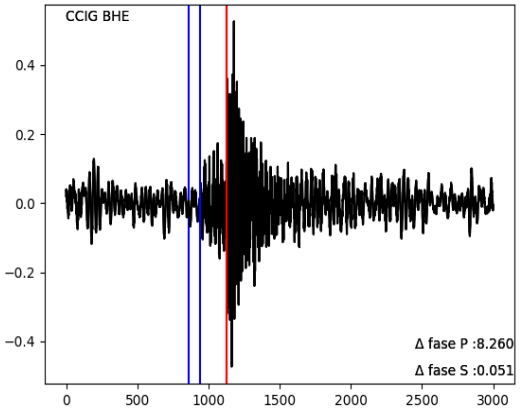


(a) (b)

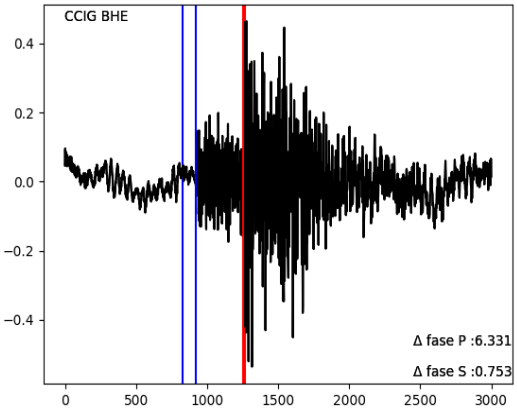
(c) (d)

Figura 5.4: En la figura se pueden ver 4 formas de onda para las cuales el modelo de red convolucional dio predicciones que difieren por mas de 1 segundo de las dos picadas de fase realizadas por los analistas del INSIVUMEH. Las picadas de fase P se pueden apreciar en azul y las picadas de fase S en rojo. Las diferencias en segundos de la predicción hecha por el modelo y los datos del catalogo de prueba se pueden en la parte inferior derecha de las trazas.

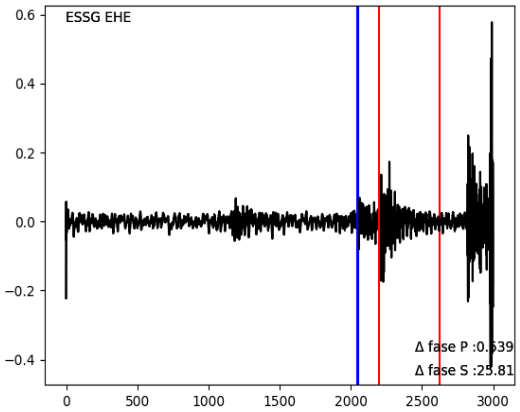
Coincidencias de 1 fase



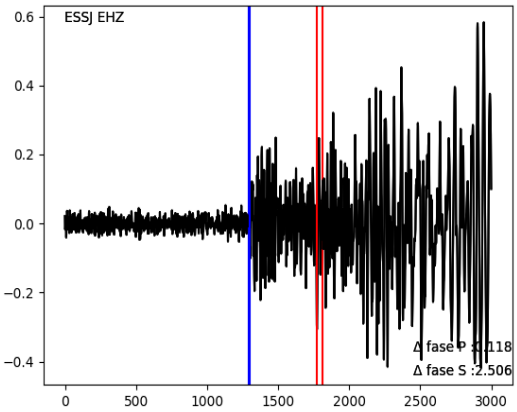
(a)



(b)



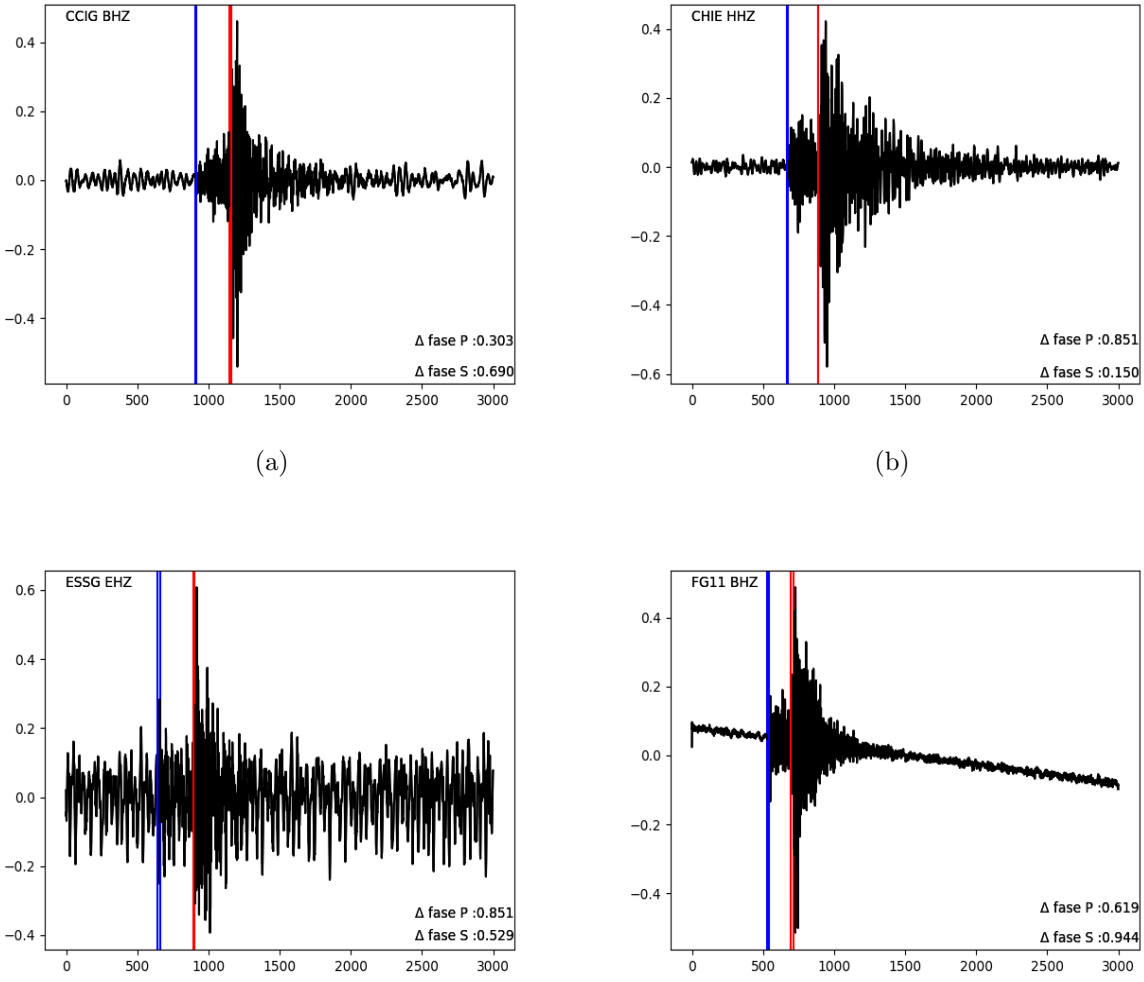
(c)



(d)

Figura 5.5: En la figura se pueden ver 4 formas de onda para las cuales el modelo de red convolucional dio alguna predicción que difieren por mas de 1 segundo y una predicción que coincide(diferencia de menos de 1 segundo) con la picadas de fase realizadas por los analistas del INSIVUMEH. Las picadas de fase P se pueden apreciar en azul y las picadas de fase S en rojo. Las diferencias en segundos de la predicción hecha por el modelo y los datos del catalogo de prueba se pueden en la parte inferior derecha de las trazas.

Coincidencias de 2 fases



(a) (b)

(c) (d)

Figura 5.6: Para los cuatro casos que se ven en la figura se tienen coincidencias con las predicciones del modelo y los datos del conjunto de prueba. En estos casos es difícil ver gráficamente la diferencia entre la predicción y los datos del catalogo ya que las diferencias son de menos de 1 segundo. En la parte inferior derecha se puede la diferencias en segundos de las predicciones y los datos de prueba. Las picadas de fase P se ven en color azul y las picadas de fase S se ven en color rojo.

5.2.2. Segundo Modelo

El segundo modelo se entreno con datos de agosto 2019 a noviembre 2019. Los siguientes resultados corresponden a datos de prueba del mes de julio 2019, se han evaluado 5098 casos. A continuación se muestran tablas con los resultados, se han separado casos en los que se tienen menos de 1 segundo de diferencia con respecto al catalogo de prueba, menos de 2 segundos de diferencia y menos de 3 segundos, también se ha indicado que fase fue identificada(P o S) y se ha hecho un conteo de para cuantos casos se han identificado a las dos fases o solamente una.

Resultados menores a 1 segundo

Fases P identificadas	607
Fases S identificadas	1279
1 fase identificada	1621
2 fases identificadas	265

Resultados menores a 2 segundos

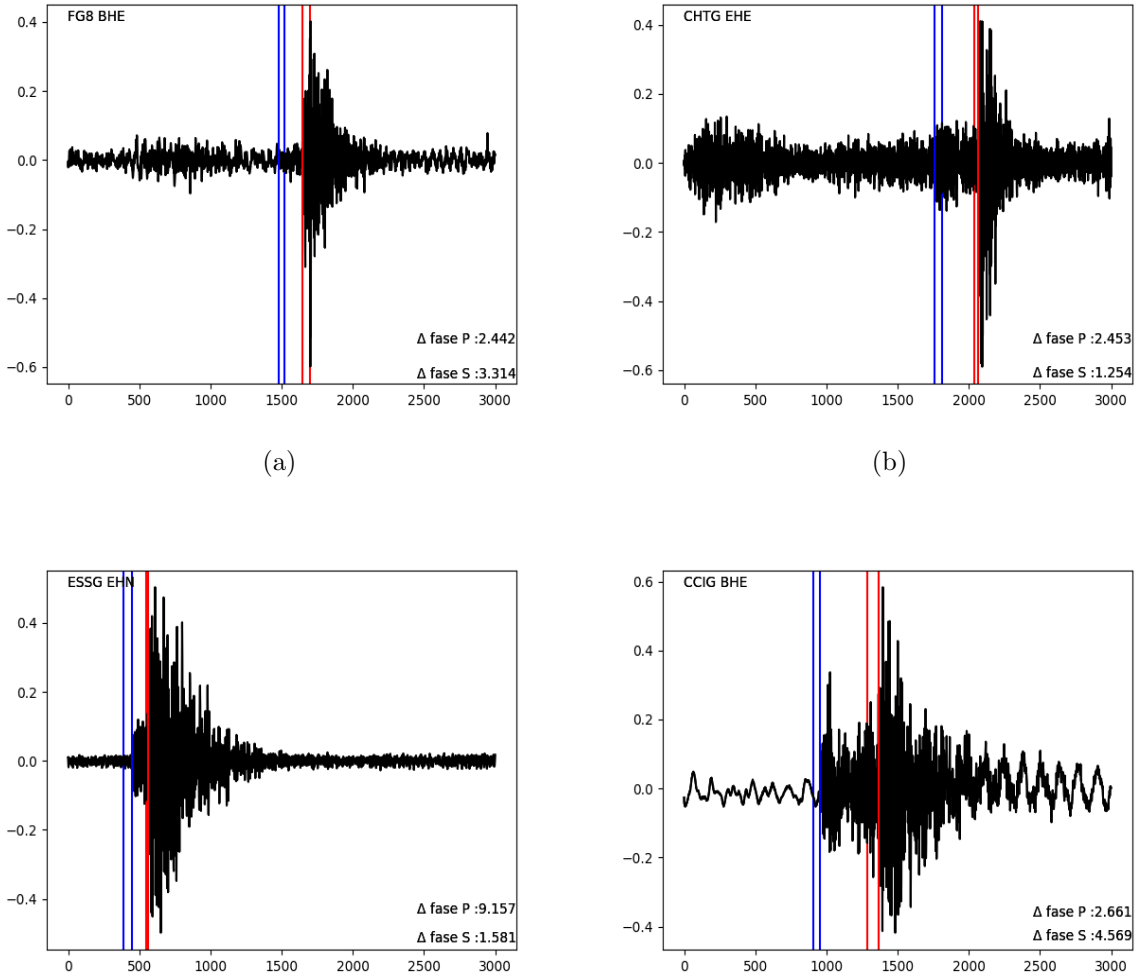
Fases P identificadas	1168
Fases S identificadas	2239
1 fase identificada	2629
2 fases identificadas	778

Resultados menores a 3 segundos

Fases P identificadas	1640
Fases S identificadas	2847
1 fase identificada	3242
2 fases identificadas	1245

A continuación se pueden apreciar 4 casos para cada criterio de coincidencias.

Sin coincidencias

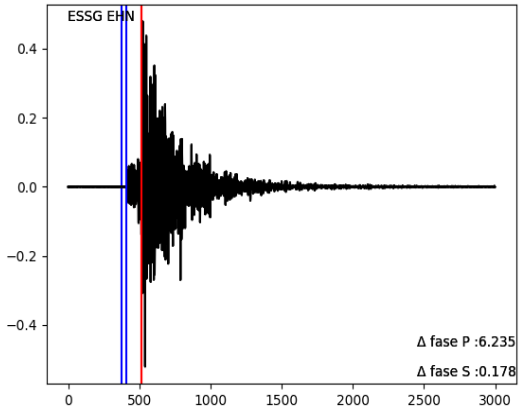


(a) (b)

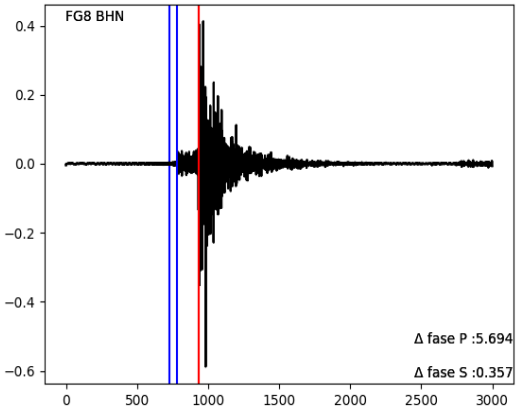
(c) (d)

Figura 5.7: En la figura se pueden ver 4 formas de onda para las cuales el modelo de red convolucional dio predicciones que difieren por mas de 1 segundo de las dos picadas de fase realizadas por los analistas del INSIVUMEH. Las picadas de fase P se pueden apreciar en azul y las picadas de fase S en rojo. Las diferencias en segundos de la predicción hecha por el modelo y los datos del catalogo de prueba se pueden en la parte inferior derecha de las trazas.

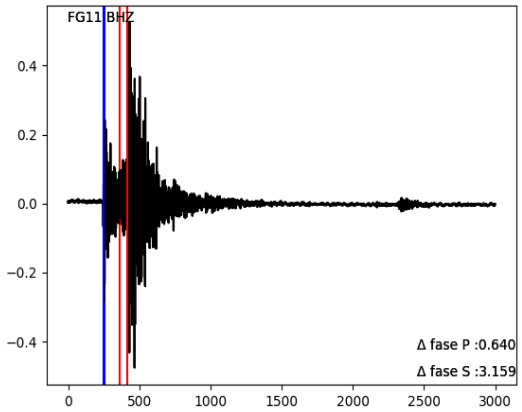
Coincidencias de 1 fase



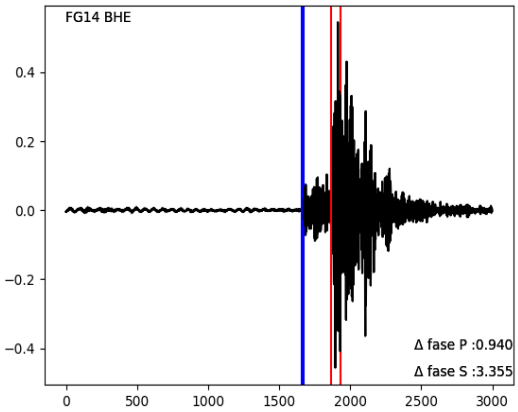
(a)



(b)



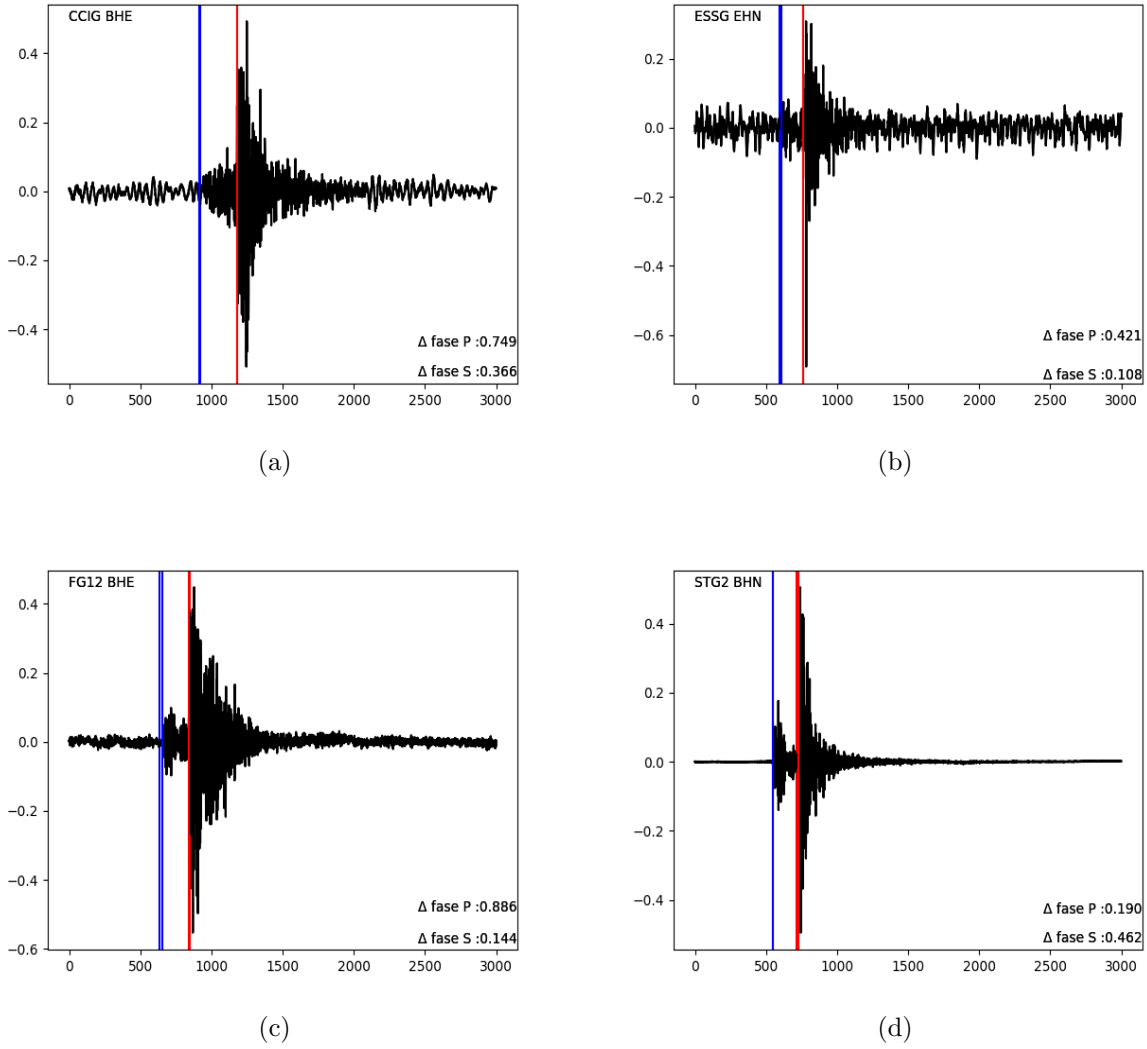
(c)



(d)

Figura 5.8: En la figura se pueden ver 4 formas de onda para las cuales el modelo de red convolucional dio alguna predicción que difieren por mas de 1 segundo y una predicción que coincide(diferencia de menos de 1 segundo) con la picadas de fase realizadas por los analistas del INSIVUMEH. En las figuras (a) y (b) se puede ver una coincidencia en la fase S y en la figuras (c) y (d) se puede apreciar coincidencia en la fase P.

Coincidencias de 2 fases



(a) (b)

(c) (d)

Figura 5.9: Para los cuatro casos que se ven en la figura se tienen coincidencias con las predicciones del modelo y los datos del conjunto de prueba. En estos casos es difícil ver gráficamente la diferencia entre la predicción y los datos del catalogo ya que las diferencias son de menos de 1 segundo. En la parte inferior derecha se puede ver la diferencias en segundos de las predicciones y los datos de prueba. Las picadas de fase P se ven en color azul y las picadas de fase S se ven en color rojo.

Capítulo 6

Conclusiones

- Las redes neuronales profundas tienen capacidad de reconocer patrones en las señales de los sismogramas, sin embargo se necesitan muchos mas datos para que estas tengan desempeños aceptables.
- Las redes neuronales convolucionales tienen un mejor desempeño que las redes neuronales profundas para identificar fases sísmicas en los modelos evaluados.
- Mejorar el conjunto de datos de entrenamiento, prueba y validación es crucial para mejorar los resultados de las redes, esto incluye depuración de datos ruidosos, otro factor es crear rutinas que diferencien las muestras en las que hay dos eventos sísmicos.
- Se pudo notar en los resultados que las redes tienen un desempeño prometedor en la identificación de fases sísmicas y que tienen el potencial de automatizar o auxiliar la picada de fases sísmicas en el departamento de Geofísica del INSIVUMEH.
- Se puede concluir que las redes neuronales pueden asistir a los analistas del departamento de Geofísica del INSIVUMEH.
- Se pudo notar que las redes tuvieron un mejor desempeño al identificar fases de tipo S, lo cual es contradictorio ya que por lo regular la fase P es mas clara debido a que la fase S viene escondida por la coda de la fase P, se necesitaría investigar en proyectos futuros este comportamiento para explotarlo o corregir aspectos de la arquitectura utilizada para tener mejores resultados en la identificación de la onda Primaria.

Capítulo 7

Recomendaciones

- Investigar la utilizacion de 3 canales por cada muestra, esto puede concluirse de varios articulos citados en las referencias [5], [6].
- Se puede cambiar las arquitecturas de redes neuronales elaboradas para que las salidas sean distribuciones de probabilidad como un vector multidimensional con la misma dimensionalidad que las entradas.
- Crear rutinas para determinación de hiperparametros, esto con la finalidad de optimizar el procesamiento de las redes.
- Investigar la utilización de redes neuronales recurrentes, esto debido a que estas arquitecturas tienen resultados excepcionales con series de tiempo tales como los sismogramas.
- Realizar pruebas de gray box y black box para encontrar aspectos físicos que las redes neuronales determinan en sus capas ocultas, es decir accesar directamente a los parámetros determinados en el proceso del descenso del gradiente con el fin de determinar como ve la red neuronal internamente a una picada de fase.
- Se ha notado que al utilizar modelos muy complejos, es decir modelos en los que se utilizan varias capas con una cantidad elevada de neuronas, se consume demasiada memoria RAM y mucha capacidad de procesamiento, para minimizar esto se pueden utilizar las capacidades de redes neuronales convolucionales o recurrentes.

Bibliografía

[1] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[2] C.M. Bishop, I.T. Nabney. *Pattern Recognition and Machine Learning: A Matlab companion.* Springer. 2008

[3] Jack Woollam, Andreas Rietbrock, Angel Bueno, Silvio De Angelis. *Convolutional Neural Network for Seismic Phase Classification, Performance Demonstration over a Local Seismic Network*. Seismological Research Letters. `https://doi.org/10.1785/0220180312`.

[4] Patrice Y. Simard, Dave Steinkraus, John C. Platt. *Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis*. IEEE Computer Society. Proceedings of the Seventh International Conference on Document Analysis and Recognition (ICDAR 2003). `http://cognitivemedium.com/assets/rmnist/Simard.pdf`

[5] Qingkai Kong, Daniel T. Trugman, Zachary E. Ross, Michael J. Bianco, Brendan J. Meade, Peter Gersoft. *Machine Learning in Seismology: Turning Data into Insights*. Seismological Research Letters. `https://doi.org/10.1785/0220180259`.

[6] Weiqiang Zhu, Gregory Beroza. *PhaseNet: A Deep-Neural-Network-Based Seismic Arrival Time Picking Method*. Geophysical Journal International. `https://arxiv.org/abs/1803.03211`

[7] Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton. *ImageNet Classification with Deep Convolutional Neural Networks*. `http://www.cs.toronto.edu/~fritz/absps/imagenet.pdf`

[8] Jens Havskov, Lars Ottemöller. *Routine Data Processing in Earthquake Seismology*. Springer, 2010.

[9] Seth Stein, Michael Wysession. *An Introduction to Seismology, Earthquakes and Earth Structure*. Blackwell Publishing, 2003.

[10] Chris H. Chapman. *FUNDAMENTAL OF SEISMIC WAVE PROPAGATION*. Cambridge University Press, 2004.

[11] Lars Ottemöller, Peter Voss, Jens Havskov. *SEISAN EARTHQUAKE ANALYSIS SOFTWARE*. [En linea] `seis.geus.net/software/seisan/seisan.pdf`

[12] Trevor Hastie, Robert Tibshirani, Jerome Friedman. *The Elements of Statistical Learning*. Springer. 2nd Edition.

[13] Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn and TensorFlow, Concepts, Tools and Techniques to Build Inteligent Systems*. O'REILLY.

[14] Lars Ottemöller, Peter Voss, Jens Havskov. *SEISAN EARTHQUAKE ANALYSIS SOFTWARE FOR WINDOWS, SOLARIS, LINUX and MACOSX*,2017. `http://seis.geus.net/software/seisan/seisan.pdf`