PONTIFICIA UNIVERSIDAD CATÓLICA DEL ECUADOR

FACULTAD DE INGENIERÍA

CARRERA DE: SISTEMAS DE INFORMACIÓN

TRABAJO DE TITULACIÓN

# TEMA: DETERMINAR LA ACELERACIÓN DEL TERRENO UTILIZANDO REDES NEURONALES ARTIFICIALES CON DATA SÍSMICA DE CALIFORNIA.

AUTOR:

JOSÉ MIGUEL ALVIAR MORALES

QUITO DM, JULIO DE 2024

# ÍNDICE

# 1. CAPÍTULO I: INTRODUCCIÓN

## 1.1 Tema

Determinar la aceleración del terreno utilizando Redes Neuronales Artificiales con data sísmica de California.

## 1.2 Justificación

La región de California cuenta con redes sísmicas las cuales permiten tener información en tiempo real de terremotos que ocurran, lo cual permite almacenar la información, procesarla, analizarla e implementar medidas de respuesta oportuna, así como la determinación de parámetros asociados al evento sísmico.

El presente trabajo se realiza con la finalidad de proveer a los ingenieros civiles de un valor de salida que es la aceleración del terreno con la distancia epicentral cuando ocurren eventos sísmicos utilizando la técnica de redes neuronales.

Una vez conocida la aceleración del terreno, los ingenieros civiles calculan las fuerzas dinámicas sobre las estructuras que resulta del producto de la masa por la aceleración.

La utilidad de conocer o determinar las fuerzas sísmicas sobre las estructuras es que los ingenieros pueden realizar diseños confiables a fin de evitar el colapso de las estructuras y salvaguardar la vida de las personas.

## 1.3 Planteamiento del problema

Para el diseño de las estructuras ante cargas sísmicas se debe conocer un parámetro muy importante que es la aceleración del terreno. Se plantea en este estudio un método alternativo al que utilizan los ingenieros civiles (regresión multivariable) utilizando las redes neuronales artificiales como herramienta de cálculo y bajo la concepción de ingeniería en sistemas de manera de proveer otra forma de cuantificar la aceleración del terreno con la distancia epicentral. Una vez conocida la aceleración del terreno se determina la fuerza sísmica a la que estarían sometidas las estructuras permitiendo realizar su diseño evitando su colapso y salvaguardando la vida de las personas.

### 1.4 Antecedentes

En tres trabajos consultados **[17, 18 y 19]** que tienen relación con el presente estudio se destacan los siguientes aspectos:

En la referencia **[17]**, se realizó el análisis de la predicción de la aceleración del terreno, a partir de la data sísmica del sur de California de SCEC (South California Earthquake Center), utilizando dos modelos de machine learning que son Redes neuronales Artificiales y Random Forest, se usó un dataset con 150,000 datos y las variables fueron la distancia hipocentral, distancia epicentral, latitud, longitud y la variable predictora es la aceleración máxima del terreno. El valor del $R^2$ es de 0.86 y se obtuvo un error del 15%.

En la referencia **[18]**, se comparan métodos tradicionales empíricos vs el modelo de machine learning redes neuronales artificiales, el análisis es realizado en el sur de California con 52,297 registros proporcionados por la PEER, las variables consideradas fueron la magnitud, distancia hipocentral, velocidad de onda de corte y la variable predictora aceleración máxima del terreno. Para la red neuronal, solo se utilizó la librería TensorFlow trabajada en Python y se determinó una sola métrica de evaluación que es el $R^2$ (el valor que se obtuvo en este trabajo del $R^2$ es de 0.8251), mientras que para el cálculo empírico se utilizarán fórmulas a partir de valores conocidos para determinar la aceleración del terreno.

En la referencia **[19]**, se realizó el análisis de la predicción de la aceleración de la aceleración del terreno con 1184 registros proporcionados por la PEER utilizando las variables velocidad del terreno, desplazamiento del terreno, aceleración pseudoespectral y la variable predictora que es la aceleración del terreno. Como tipología de la red neuronal, se utilizó 3 capas que son la capa de entrada, la capa oculta y la capa de salida con 3 neuronas cada uno y se utilizaron las funciones de activación sigmoide logarítmica, tangente sigmoide y función lineal.

A diferencia de los trabajos previamente citados, se utiliza solo un modelo de machine Learning que es Red Neuronal artificial, utilizando tres librerías que son TensorFlow, PyTorch y Scikit-Learn para determinar cuál de las tres librerías es mejor en términos de $R^2$ y se realiza el análisis para toda el área de California a partir de los datos suministrados por la PEER (Pacific Earthquake Engineering Research Center).

Se utiliza el modelo de red neuronal artificial implementada en Python Jupyter Notebook en Google collab utilizando las tres librerías mencionadas anteriormente con tres funciones de activiación que son sigmoide, tangente hiperbólica y ReLU. Los registros son de 21,198 suministradas por la PEER y se utilizaran cinco métricas que sirven para evaluar el desempeño de la predicción de la red neuronal artificial que son: $R^2$, MSE, RMSE, RMSLE y MAE.

Adicionalmente, se utilizan algunas variables que no están contempladas en los estudios previamente indicados como mecanismo de falla (Mechanism Base on Rake Angle), tipología de suelo las cuales presentan un peso importante en la correlación, además de la distancia epicentral (EpiD (km)), magnitud del sismo (Earthquake Magnitude), velocidad de onda de corte (Vs30 m/s) y la aceleración máxima del terreno (PGA (g)).

La tipología de los modelos es una capa de entrada de 5 neuronas (debido a que tenemos 5 variables independientes), de 3 capas ocultas de 64, 32, 16 neuronas y 1 capa de salida conformada de una neurona donde se verá reflejado el valor de la predicción del modelo de la red neuronal correspondiente a la aceleración del terreno presente en las tres librerías mencionadas anteriormente.

Se realizan 100 iteraciones y se maneja el tema del overfitting y underfitting con la función de pérdida y el optimizador Adams en los modelos, con la finalidad de que el algoritmo sea capaz de generar y aprender nuevos datos y así obtener el mejor valor de la predicción de la aceleración del terreno.

## 1.5 Objetivos

### 1.5.1 General

Aplicar Redes Neuronales Artificiales para determinar la aceleración del terreno en eventos sísmicos ocurridos en California usando la base de datos de la PEER (Pacific Earthquake Engineering Research Center).

### 1.5.2 Específicos

- Buscar una base de datos de sismos ocurridos en California.
- Seleccionar las variables de entrada que serán ingresadas al aplicar la técnica de Redes Neuronales.
- Aplicar la técnica de Redes Neuronales implementada en Python utilizando las librerías TensorFlow, PyTorch y Scikit-Learn y las variables de entrada previamente seleccionadas para determinar la aceleración del terreno.
- Seleccionar la data de validación para someter a prueba la Red Neuronal Artificial implementada y cuantificar el error en la predicción de la aceleración del terreno en cada una de las librerías utilizadas y realizar comparación entre ellas.

## 1.6 Alcance

La contextualización de este estudio se centra en la región de California donde existe una amplia data sísmica recopilada por la Pacific Earthquake Engineering Research Center (PEER) mediante la instrumentación de redes sismológicas que miden la aceleración del terreno ante eventos sísmicos.

El alcance de este estudio es **determinar la aceleración del terreno con la distancia al epicentro del terremoto utilizando redes neuronales** usando el lenguaje de programación Python en Jupyter Notebook a partir de unos datos/variables sismológicas de entrada con la finalidad de hallar como resultado o salida la aceleración esperada del terreno.

### 1.7 Hipótesis

Utilizando la base de datos de eventos sísmicos ocurridos en California y aplicando la técnica de Redes Neuronales Artificiales se logra determinar la aceleración máxima del terreno con un coeficiente de correlación $R^2$ mayor o igual a 75% en función de la distancia epicentral, distancia hipocentral la magnitud del sismo, mecanismo de falla y tipología de suelo.

### 1.8 Procedimiento marco metodológico

A continuación, se presenta la metodología a seguir en el presente trabajo de titulación.

Se recolectará los datos en la base de datos de California disponible en un archivo tipo CSV. Se seleccionará las variables consideradas más importantes para predecir la aceleración máxima del terreno, las cuales son: distancia epicentral, la magnitud del sismo, mecanismo de falla, tipología de suelo. Se aplicará la técnica de Redes Neuronales Artificiales utilizando las librerías TensorFlow, PyTorch y Scikit-Learn implementadas en Python y utilizando las variables de entrada previamente seleccionadas para determinar la aceleración máxima del terreno. Se someterá a prueba las librerías implementadas para cuantificar el error en la predicción de la aceleración máxima del terreno y se realizará comparación entre ellas.

## 2. CAPÍTULO II: FUNDAMENTACIÓN TEÓRICA

### 2.1 Métodos de Redes Neuronales Artificiales

Las Redes Neuronales Artificiales (RNA) son un modelo computacional inspirado en el funcionamiento del cerebro humano. Consisten en una colección de unidades básicas llamadas neuronas, organizadas en capas interconectadas. Cada neurona realiza operaciones matemáticas simples en los datos de entrada y pasa el resultado a través de una función de activación.

**Cálculo de la Red Neuronal:** Para realizar los cálculos en la red neuronal se realiza de la siguiente forma:

**Entrada de datos**: La red neuronal recibe datos de entrada, que pueden ser cualquier tipo de información, como imágenes, texto, señales de audio, etc. Estos datos se representan como vectores numéricos.

**Propagación hacia adelante (forward propagation):** Los datos de entrada se propagan a través de la red neuronal, desde la capa de entrada hasta la capa de salida. Cada neurona en una capa está conectada a todas las neuronas de la capa siguiente mediante conexiones ponderadas.

**Operaciones en las neuronas:** Cada neurona realiza dos operaciones principales:

**Suma ponderada:** La neurona calcula la suma ponderada de todas las entradas, multiplicando cada entrada por el peso de la conexión correspondiente y sumándolas.

**Función de activación:** Después de que se realiza la suma ponderada de las entradas multiplicadas por los pesos, la función de activación se aplica al resultado para introducir no linealidades en la salida de la neurona. Esto permite que la red neuronal aprenda y capture relaciones complejas y no lineales entre las entradas y las salidas.

A continuación, se presentan 3 funciones de activación que muestran las librerías TensorFlow y PyTorch:

**Función Sigmoide (Logística):**

$$\boldsymbol{\sigma(x) = \frac{1}{1 + e^{-x}}}$$

**donde x es:** la suma ponderada en cada neurona.

e = 2.71828 (Euler)

Esta función mapea los valores de entrada en un rango de 0 a 1. Tiene la forma de una curva "S" y es útil en problemas de clasificación binaria.

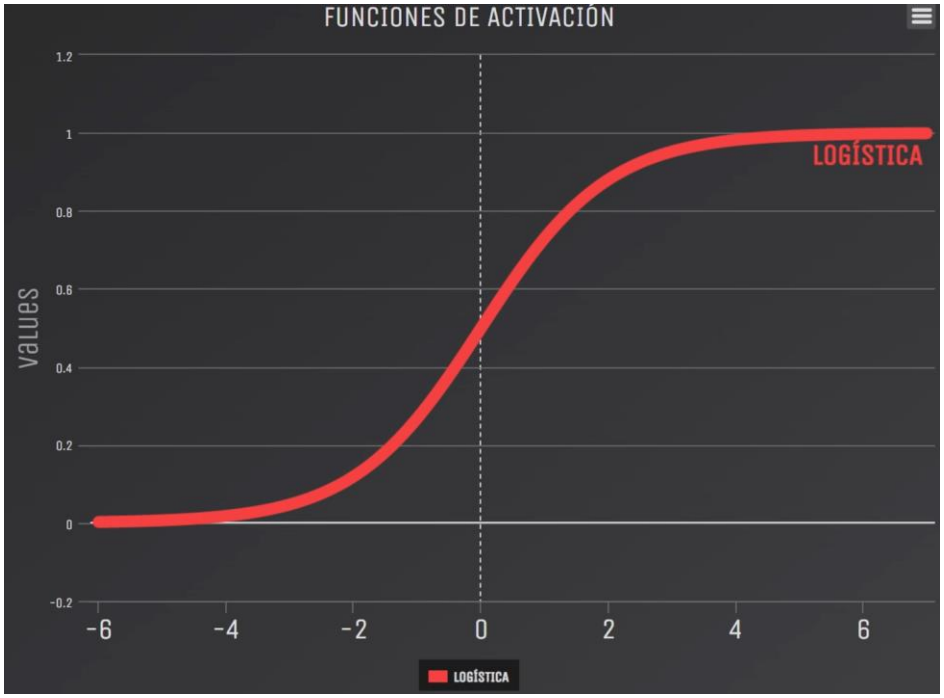



***Figura 1: Función Sigmoide (Logística).***

**Función Tangente Hiperbólica (Tanh):**

$$\boldsymbol{tanh}(\boldsymbol{x}) = \frac{\boldsymbol{e^{x}} - \boldsymbol{e^{-x}}}{\boldsymbol{e^{x}} + \boldsymbol{e^{-x}}}$$

Similar a la función sigmoide, pero mapea los valores de entrada en un rango de -1 a 1. Tiene la forma de una curva "S" centrada en el origen y es útil en problemas de clasificación y regresión.

Donde x la suma ponderada en cada neurona y e es una constante con un calor de 2.71828 (Euler).

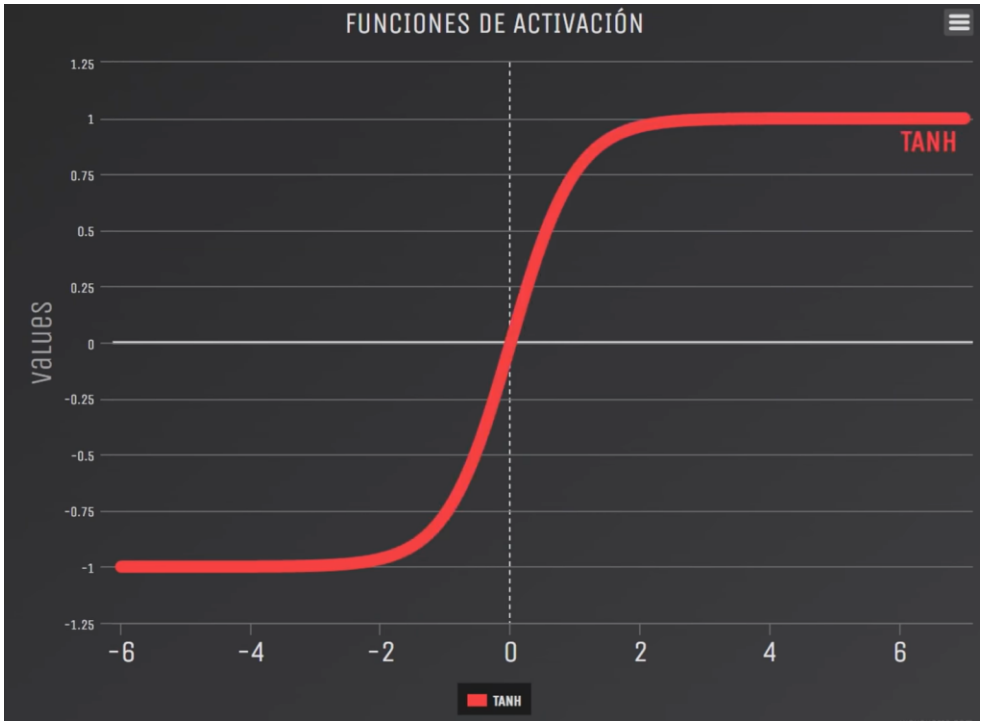



***Figura 2: Función de activación, Tangente Hiperbólica.***

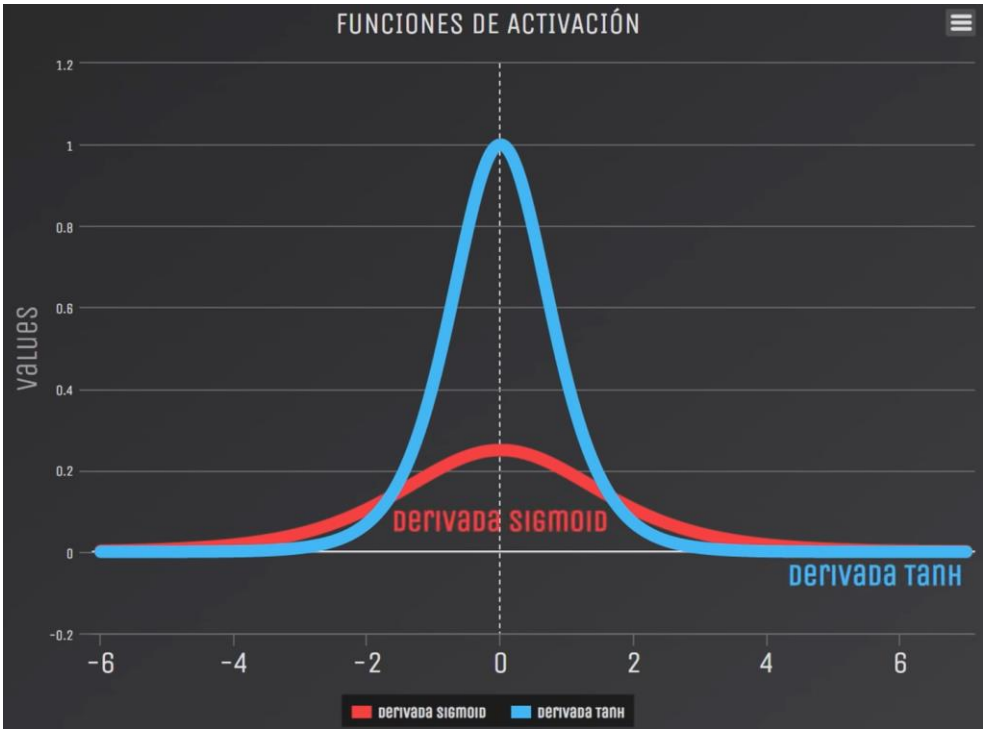



***Figura 3: Función de activación, Tangente Hiperbólica y Función Sigmoide (Logística).***

En la Figura 3, podemos observar que la derivada de la función tangente hiperbólica exhibe variaciones más amplias en sus valores en comparación con la función sigmoide. Esto conduce a una mayor eficiencia en el aprendizaje y en los costos computacionales al utilizar la función de activación tangente hiperbólica.

**Rectified Linear Unit (ReLU):**

**ReLU(x) = max(0,x)**

Esta función devuelve 0 si la entrada es negativa y la misma entrada si es positiva. Es una función simple y eficaz que se utiliza ampliamente en redes neuronales profundas debido a su capacidad para mitigar el problema del desvanecimiento del gradiente.

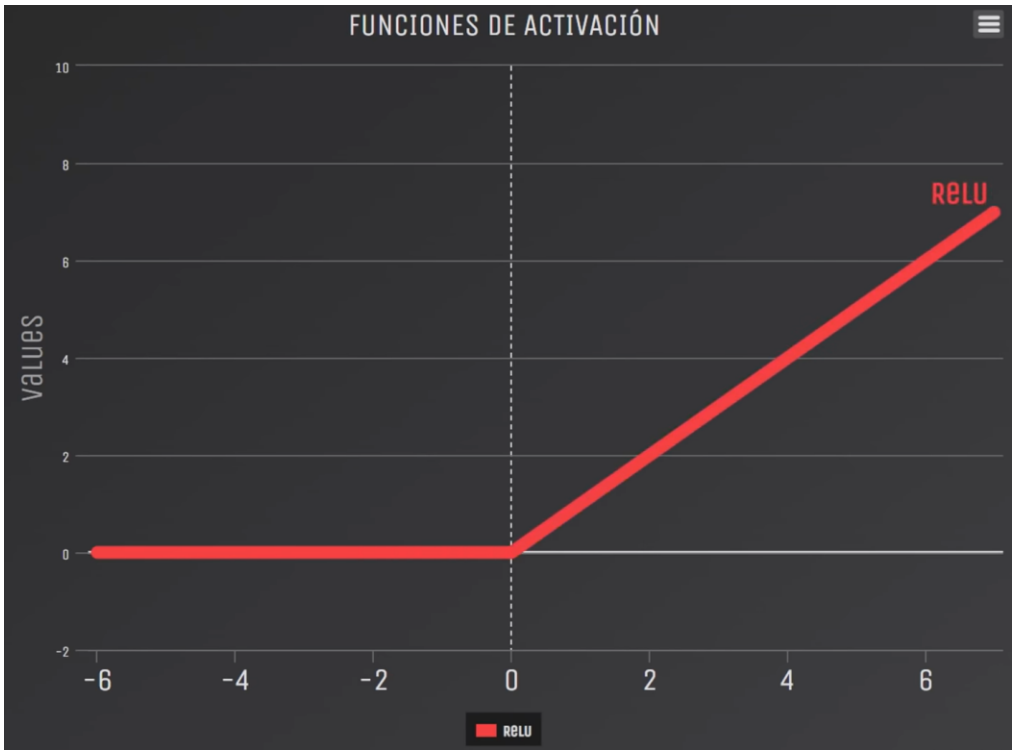



***Figura 4: Función de activación, Rectified Linear Unit (ReLU).***

Scikit-Learn no proporciona una implementación de redes neuronales como PyTorch y TensorFlow, pero ofrece herramientas para preprocesamiento de datos, selección de modelos, evaluación de modelos, etc.

Una vez realizada la función de activación, se procede con el siguiente paso para el cálculo de la red neuronal.

**Propagación hacia atrás (backpropagation):** Después de que los datos han atravesado la red y se ha generado una salida, se compara esta salida con la salida deseada (en el caso del aprendizaje supervisado). La diferencia entre la salida real y la deseada se utiliza para ajustar los pesos de las conexiones en la red, utilizando un algoritmo de optimización como el descenso de gradiente. Este proceso se realiza iterativamente para minimizar el error de la red.

El descenso de gradiente es un algoritmo de optimización utilizado para minimizar una función de costo (o pérdida) ajustando los parámetros de un modelo. Su objetivo principal es encontrar los valores de los parámetros que minimizan la función de costo, lo que lleva a un mejor rendimiento del modelo en la tarea que se está abordando.

Esta función de costo o perdida generalmente se calcula a partir del error cuadrático medio con la siguiente la fórmula:

$$ECM = \frac{1}{n}\sum_{i=1}^{n}(y - \hat{y})^2$$

Donde y es el valor de salida real conocido en el proceso de aprendizaje y $\bar{y}$ son los valores predichos con el modelo de la red neuronal. El objetico es reducir el error cuadrático medio a su valor mínimo como se muestra a continuación:

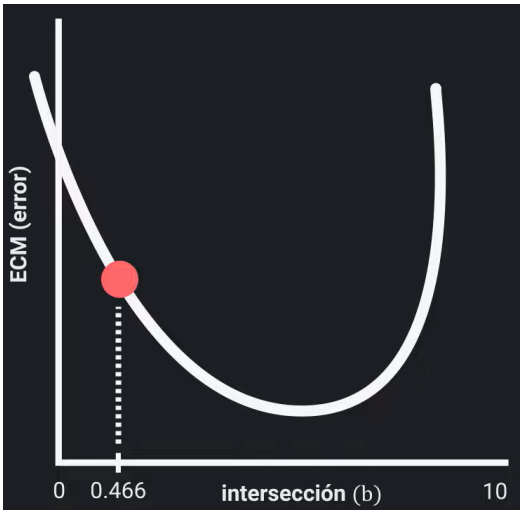



***Figura 5: Reducción del error cuadrático medio con el método del gradiente.***

En el contexto del entrenamiento de redes neuronales, el descenso de gradiente juega un papel crucial en la actualización de los pesos y sesgos de la red para mejorar su capacidad predictiva.

El cálculo del descenso de gradiente se presenta a continuación:

**Cálculo del gradiente**: El descenso de gradiente calcula el gradiente de la función de costo con respecto a los parámetros del modelo. El gradiente indica la dirección y la magnitud del cambio necesario en los parámetros para minimizar la función de costo. Se calcula utilizando técnicas de diferenciación automática, como la retro propagación (backpropagation), que permite calcular eficientemente el gradiente en redes neuronales profundas.

**Actualización de los parámetros**: Una vez calculado el gradiente, el descenso de gradiente actualiza los parámetros del modelo en la dirección opuesta al gradiente. Esto se hace multiplicando el gradiente por una tasa de aprendizaje, que controla el tamaño de los pasos que se dan en cada iteración del algoritmo. La fórmula general para la actualización de los parámetros $\theta$ en el paso $\tau$ del descenso de gradiente es:

$$\theta_{\tau+1} = \theta_{\tau} - \alpha \nabla J(\theta_{\tau})$$

Donde $\theta$ representa la tasa del modelo, $\alpha$ es la tasa de aprendizaje y $\nabla J(\theta_{\tau})$ es el gradiente de la Función de costo J evaluado en los parámetros actuales $\theta_{\tau}$.

**Iteración:** Este proceso de cálculo del gradiente y actualización de los parámetros se repite iterativamente durante varios pasos hasta que se alcance un valor mínimo en el error cuadrático medio.

El descenso de gradiente es fundamental en el entrenamiento de redes neuronales porque permite que el modelo ajuste sus parámetros para minimizar la diferencia entre las predicciones del modelo y los valores reales de los datos de entrenamiento.

Al optimizar los parámetros de la red neuronal de esta manera, se mejora su capacidad para realizar predicciones precisas en datos no vistos.

**Entrenamiento de la red:** El proceso de propagación hacia adelante y hacia atrás se repite muchas veces durante el entrenamiento de la red, utilizando un conjunto de datos de entrenamiento. El objetivo del entrenamiento es ajustar los pesos de las conexiones de la red para que pueda producir salidas precisas para entradas dadas.

**Uso de la red entrenada:** Una vez que la red ha sido entrenada con éxito, puede utilizarse para hacer predicciones o tomar decisiones sobre nuevos datos de entrada que no formaban parte del conjunto de entrenamiento.

## 2.2 Librería TensorFlow

TensorFlow es una biblioteca de código abierto desarrollada por Google y lanzada en 2015 para realizar cálculos numéricos utilizando gráficos de flujo de datos y sirve como herramienta para investigación y producción en aprendizaje automático. Está diseñado especialmente para el aprendizaje automático y la inteligencia artificial. La principal característica de TensorFlow es su capacidad para realizar cálculos en tensores, que son arreglos multidimensionales, lo que la hace ideal para trabajar con redes neuronales artificiales (RNA).

En un entorno de Jupyter Notebook, Tensor Flow se utiliza importando la biblioteca y luego definiendo y entrenando modelos de redes neuronales utilizando su API. Esto implica la definición de tensores para los datos de entrada y salida, la construcción de capas de la red neuronal utilizando las clases proporcionadas por TensorFlow, la especificación de la función de pérdida, el optimizador y finalmente, el entrenamiento del modelo iterativamente ajustando los pesos de la red para minimizar la pérdida.

TensorFlow es una librería importante debido a su capacidad para implementar una amplia variedad de algoritmos de aprendizaje automático y redes neuronales de manera eficiente. Su gran base de usuarios y su activa comunidad de desarrollo garantizan un soporte continuo y la disponibilidad de recursos educativos y ejemplos prácticos.

TensorFlow utiliza un paradigma de programación conocido como "grafo de flujo de datos", que es un enfoque para definir y ejecutar operaciones matemáticas de manera eficiente. A continuación, se presenta una explicación detallada sobre cómo funciona esto:

**Definición del grafo**: En TensorFlow, primero se define un grafo computacional que representa las operaciones matemáticas que se realizarán durante la ejecución del modelo. Este grafo consta de nodos que representan las operaciones y de bordes que representan los datos (tensores) que fluyen entre las operaciones.

**Construcción del grafo:** En el código de TensorFlow, se definen las operaciones matemáticas que componen el modelo, así como las variables que contienen los parámetros del modelo. Estas operaciones y variables se combinan para formar el grafo computacional. No se realiza ninguna computación real en este momento; solo se define la estructura del grafo y las relaciones entre las operaciones.

**Ejecución del grafo:** Una vez que el grafo ha sido definido, se puede ejecutar en una sesión de TensorFlow. Durante esta fase, se asignan los recursos computacionales necesarios y se realizan los cálculos. Los datos de entrada se proporcionan al grafo, y TensorFlow propaga estos datos a través de las operaciones del grafo siguiendo las dependencias definidas. Esto se conoce como "flujo de datos" a través del grafo.

**Finalización de la sesión:** Una vez que se han completado todos los cálculos necesarios, la sesión de TensorFlow se cierra y los recursos se liberan.

### 2.3 Librería Pytorch

PyTorch es una biblioteca de aprendizaje automático de código abierto desarrollada por Facebook y lanzado en 2017. Se destaca por su flexibilidad y facilidad de uso, especialmente en la investigación de aprendizaje profundo y el desarrollo de modelos de inteligencia artificial. Al igual que TensorFlow, PyTorch permite la creación y entrenamiento de modelos de redes neuronales. La principal característica de la librería PyTorch es su capacidad para realizar cómputo en tensores de forma dinámica y su enfoque flexible. PyTorch se basa en el lenguaje de programación Python, lo que facilita su integración con otros paquetes y librerías de Python.

En un entorno de Jupyter Notebook, Pytorch se utiliza importando la biblioteca y luego definiendo y entrenando modelos de redes neuronales utilizando su API. Esto implica la definición de tensores para los datos de entrada y salida, la construcción de capas de la red neuronal utilizando las clases proporcionadas por PyTorch, la especificación de la función de pérdida y el optimizador y finalmente el entrenamiento del modelo iterativamente ajustando los pesos de la red para minimizar la pérdida.

La importancia de PyTorch radica en su enfoque amigable para el usuario y su capacidad para realizar cómputo en tensores de manera dinámica. Esto lo hace ideal para la investigación y el

desarrollo de prototipos en el campo del aprendizaje profundo. Además, PyTorch ha ganado popularidad en la comunidad de investigación debido a su API intuitiva y su integración con Python, lo que facilita la experimentación y la iteración rápida en proyectos de aprendizaje automático.

TensorFlow y PyTorch son dos de las bibliotecas más populares para aprendizaje automático y redes neuronales. La principal diferencia radica en su enfoque para definir y ejecutar modelos. TensorFlow utiliza un grafo de flujo de datos estático, lo que significa que primero se define la estructura del modelo y luego se ejecuta. PyTorch, por otro lado, utiliza un enfoque dinámico, lo que permite una mayor flexibilidad durante la definición y ejecución del modelo.

PyTorch utiliza un grafo de ejecución dinámico. Esto significa que los grafos computacionales se construyen y ejecutan en tiempo real a medida que se ejecuta el código, en lugar de ser predefinidos antes de la ejecución.

En PyTorch, el código se ejecuta de manera imperativa, lo que significa que las operaciones se ejecutan en el orden en que se escriben en el código. Esto proporciona flexibilidad y facilidad para la depuración y el desarrollo interactivo, ya que los usuarios pueden inspeccionar y modificar los tensores y las operaciones en cualquier momento durante la ejecución.

Con el enfoque dinámico de PyTorch, los usuarios pueden definir modelos de manera más flexible y expresiva. No es necesario definir explícitamente el grafo computacional antes de la ejecución, lo que simplifica la construcción de modelos y reduce la sobrecarga de código.

La naturaleza dinámica de PyTorch facilita la construcción de modelos de manera iterativa, lo que permite a los usuarios experimentar con diferentes arquitecturas y técnicas de manera más fluida. Esto es especialmente útil durante la fase de desarrollo del modelo, ya que permite una iteración rápida y una depuración eficiente.

### 2.4 Librería Scikit-Learn

Scikit-Learn es una biblioteca de aprendizaje automático de código abierto desarrollada en Python y lanzado en 2007. Se centra en proporcionar herramientas simples y eficientes para el análisis de datos y la construcción de modelos de aprendizaje automático, incluyendo técnicas de clasificación, regresión, agrupamiento, reducción de dimensionalidad y selección de modelos. Scikit-Learn se basa en las bibliotecas NumPy, SciPy y matplolib, lo que le permite aprovechar las funcionalidades y la eficiencia de estas bibliotecas para el procesamiento de datos y la visualización.

Aunque Scikit-Learn proporciona una implementación básica de perceptrones multicapa (MLP), que es una forma básica de red neuronal en un entorno de Jupyter Notebook, los cálculos de redes neuronales en Scikit-Learn implican importar la biblioteca y luego definir y entrenar un modelo MLP utilizando la clase MLPClassifier o MLPRegressor, dependiendo del tipo de problema. Sin embargo, esta implementación es limitada en comparación con las capacidades de TensorFlow y PyTorch en términos de flexibilidad y rendimiento en el entrenamiento de redes neuronales profundas.

La importancia de Scikit-Learn radica en su capacidad para proporcionar implementaciones eficientes de una amplia variedad de algoritmos de aprendizaje automático en un entorno coherente y fácil de usar. Su amplia gama de herramientas y su documentación detallada hacen que sea una opción popular tanto para principiantes como para expertos en aprendizaje automático.

La principal diferencia entre Scikit-Learn y TensorFlow/Pytorch radica en su enfoque y en la gama de algoritmos que soportan. Scikit-Learn se centra en algoritmos de aprendizaje supervisado y no supervisado más tradicionales, como regresión lineal, clasificación y agrupamiento. Aunque puede implementar algunos tipos de redes neuronales, no está tan especializado en este ámbito como TensorFlow y PyTorch. Además, scikit-Learn está diseñado para trabajar con datos tabulares y no está tan especializado en el manejo de datos en forma de tensores como TensorFlow y PyTorch.

# 3. CAPÍTULO III: ANÁLISIS DE LA ACELERACIÓN DEL TERRENO

## 3.1 Metodología

Cuando ocurre un terremoto, las estaciones o redes sismológicas detectan o miden la ubicación del epicentro, la distancia epicentral, la aceleración del terreno, la magnitud del terremoto, además se conoce la tipología de suelo donde están ubicadas las estaciones sismológicas y el mecanismo de falla donde se originan lo sismos los cuales están indicados en la base de datos atendiendo a la geotectónica donde se originó el sismo.

La predicción de aceleración del terreno depende de varias variables entre cualitativas y cuantitativas, dentro de las cualitativas se ubica el tipo de falla geológica, tipología de suelo, dirección del movimiento, entre otros y dentro de las cuantitativas se encuentra la magnitud del sismo, profundidad del foco, distancia epicentral, etc.

Se ha seleccionado el modelado de redes neuronales porque es una técnica que puede ser implementado para variables tanto cualitativas como cuantitativas y es un tipo de modelado predictivo, automática de tipo supervisado, la cual va mejorando su respuesta si se cuenta con una gran cantidad de volúmenes de datos como la que se dispone que son de 21,305 registros.

A partir de la base de datos obtenida de la PEER, se pretende analizar la data con las tres librerías mencionadas anteriormente con la finalidad de encontrar los mejores resultados con respecto a la predicción de la aceleración del terreno, utilizando la técnica de redes neuronales artificiales.

### 3.1.1 Selección de datos de entrada.

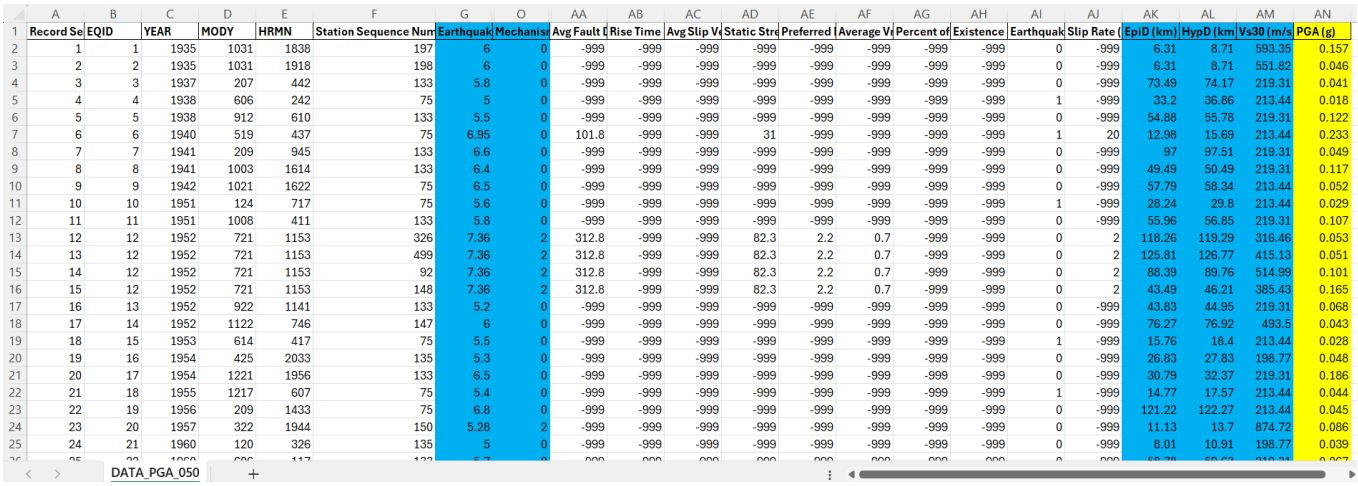
En el presente trabajo se cuenta con un archivo .CSV suministrada por la PEER en el cual cuenta con 21,198 registros y 40 variables, estos valores son los oficiales puesto que ya se ha limpiado/depurado la data, eliminando valores vacíos, blancos, nulos, caracteres inválidos y variables innecesarias. Las variables independientes más importantes consideradas son la **Distancia epicentral (EpiD (km)), Magnitud del sismo (Earthquake Magnitude), Mecanismo de falla (Mechanism Base on Rake Angle) y Tipología de suelo utilizando la velocidad de onda de corte (Vs30 m/s)** y la variable dependiente (predictora) es la **aceleración máxima del terreno (PGA (g)).**

La selección de las variables más importantes atiende a las variables que comúnmente utilizan los sismólogos para predecir la aceleración del terreno en sus modelos analíticos **[1, 2, 3, 4, 5, 6, 8, 9 y 12].**

A continuación, se presenta la definición a cada una de estas variables y una imagen del CSV a utilizar:

- **Distancia epicentral (EpiD (km)):** es la distancia entre un lugar específico en la superficie de la tierra y el epicentro de un terremoto. Es una variable cuantitativa discreta y se mide en kilómetros (Km).
- **Magnitud del sismo (Earthquake Magnitude):** es una medida cuantitativa discreta que indica la cantidad de energía liberada durante un terremoto. Es una variable importante para la predicción de la aceleración máxima del terreno.
- **Mecanismo de falla (Mechanism Based on Rake Angle):** es la forma en que las rocas o las estructuras geológicas se deforman y se rompen en respuesta a las fuerzas tectónicas o las cargas aplicadas, es una variable de tipo cualitativa nominal.
- **Aceleración máxima del terreno (PGA (g)):** es la variable que vamos a predecir, es una medida de la máxima aceleración experimentada por el terreno durante un terremoto. Indica la rapidez con la que se mueve el suelo en respuesta a las ondas sísmicas generadas por el terremoto. Es una variable cuantitativa discreta y se mide en términos de la gravedad (g).
- **Tipología de suelo (Vs30 m/s):** es la clasificación y categorización de los diferentes tipos de suelo en función de sus características físicas, composición, propiedades geotécnicas y comportamiento geológico. Esta variable esta correlacionada con la velocidad de propagación de las ondas de corte, por cuanto se sabe que, a mayor densidad del suelo, mayor es la velocidad de las ondas de corte. La tipificación de los suelos se basará en los datos medidos de la velocidad de las ondas de corte. El 30 hace referencia a que esta velocidad se mide en los primero 30 metros del perfil litológico. Es una variable cuantitativa discreta y se mide en m/s.

- **Distancia hipocentral (HypD (km)):** es la distancia que existe entre la estación sismológica y el foco del terremoto. Es una variable cuantitativa dsicreta y se mide en kilómetros (km).



| | A | B | C | D | E | F | G | O | AA | AB | AC | AD | AE | AF | AG | AH | AI | AJ | AK | AL | AM | AN |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Record Se | EQID | YEAR | MODY | HRMN | Station Sequence Num | Earthquak | Mechanist | Avg Fault D | Rise Time | Avg Slip V | Static Stre | Preferred | Average V | Percent of | Existence | Earthquak | Slip Rate ( | EpiD (km) | HypD (km | Vs30 (m/s | PGA (g) |
| 2 | 1 | 1 | 1935 | 1031 | 1838 | 197 | 6 | 0 | -999 | -999 | -999 | -999 | -999 | -999 | -999 | -999 | 0 | -999 | 6.31 | 8.71 | 593.35 | 0.157 |
| 3 | 2 | 2 | 1935 | 1031 | 1918 | 198 | 6 | 0 | -999 | -999 | -999 | -999 | -999 | -999 | -999 | -999 | 0 | -999 | 6.31 | 8.71 | 551.82 | 0.046 |
| 4 | 3 | 3 | 1937 | 207 | 442 | 133 | 5.8 | 0 | -999 | -999 | -999 | -999 | -999 | -999 | -999 | -999 | 0 | -999 | 73.49 | 74.17 | 219.31 | 0.041 |
| 5 | 4 | 4 | 1938 | 606 | 242 | 75 | 5 | 0 | -999 | -999 | -999 | -999 | -999 | -999 | -999 | -999 | 1 | -999 | 33.2 | 36.86 | 213.44 | 0.018 |
| 6 | 5 | 5 | 1938 | 912 | 610 | 133 | 5.5 | 0 | -999 | -999 | -999 | -999 | -999 | -999 | -999 | -999 | 0 | -999 | 54.88 | 55.78 | 219.31 | 0.122 |
| 7 | 6 | 6 | 1940 | 519 | 437 | 75 | 6.95 | 0 | 101.8 | -999 | -999 | 31 | -999 | -999 | -999 | -999 | 1 | 20 | 12.98 | 15.69 | 213.44 | 0.233 |
| 8 | 7 | 7 | 1941 | 209 | 945 | 133 | 6.6 | 0 | -999 | -999 | -999 | -999 | -999 | -999 | -999 | -999 | 0 | -999 | 97 | 97.51 | 219.31 | 0.049 |
| 9 | 8 | 8 | 1941 | 1003 | 1614 | 133 | 6.4 | 0 | -999 | -999 | -999 | -999 | -999 | -999 | -999 | -999 | 0 | -999 | 49.49 | 50.49 | 219.31 | 0.117 |
| 10 | 9 | 9 | 1942 | 1021 | 1622 | 75 | 6.5 | 0 | -999 | -999 | -999 | -999 | -999 | -999 | -999 | -999 | 0 | -999 | 57.79 | 58.34 | 213.44 | 0.052 |
| 11 | 10 | 10 | 1951 | 124 | 717 | 75 | 5.6 | 0 | -999 | -999 | -999 | -999 | -999 | -999 | -999 | -999 | 1 | -999 | 28.24 | 29.8 | 213.44 | 0.029 |
| 12 | 11 | 11 | 1951 | 1008 | 411 | 133 | 5.8 | 0 | -999 | -999 | -999 | -999 | -999 | -999 | -999 | -999 | 0 | -999 | 55.96 | 56.85 | 219.31 | 0.107 |
| 13 | 12 | 12 | 1952 | 721 | 1153 | 326 | 7.36 | 2 | 312.8 | -999 | -999 | 82.3 | 2.2 | 0.7 | -999 | -999 | 0 | 2 | 118.26 | 119.29 | 316.46 | 0.053 |
| 14 | 13 | 12 | 1952 | 721 | 1153 | 499 | 7.36 | 2 | 312.8 | -999 | -999 | 82.3 | 2.2 | 0.7 | -999 | -999 | 0 | 2 | 125.81 | 126.77 | 415.13 | 0.051 |
| 15 | 14 | 12 | 1952 | 721 | 1153 | 92 | 7.36 | 2 | 312.8 | -999 | -999 | 82.3 | 2.2 | 0.7 | -999 | -999 | 0 | 2 | 88.39 | 89.76 | 514.99 | 0.101 |
| 16 | 15 | 12 | 1952 | 721 | 1153 | 148 | 7.36 | 2 | 312.8 | -999 | -999 | 82.3 | 2.2 | 0.7 | -999 | -999 | 0 | 2 | 43.49 | 46.21 | 385.43 | 0.165 |
| 17 | 16 | 13 | 1952 | 922 | 1141 | 133 | 5.2 | 0 | -999 | -999 | -999 | -999 | -999 | -999 | -999 | -999 | 0 | -999 | 43.83 | 44.95 | 219.31 | 0.068 |
| 18 | 17 | 14 | 1952 | 1122 | 746 | 147 | 6 | 0 | -999 | -999 | -999 | -999 | -999 | -999 | -999 | -999 | 0 | -999 | 76.27 | 76.92 | 493.5 | 0.043 |
| 19 | 18 | 15 | 1953 | 614 | 417 | 75 | 5.5 | 0 | -999 | -999 | -999 | -999 | -999 | -999 | -999 | -999 | 1 | -999 | 15.76 | 18.4 | 213.44 | 0.028 |
| 20 | 19 | 16 | 1954 | 425 | 2033 | 135 | 5.3 | 0 | -999 | -999 | -999 | -999 | -999 | -999 | -999 | -999 | 0 | -999 | 26.83 | 27.83 | 198.77 | 0.048 |
| 21 | 20 | 17 | 1954 | 1221 | 1956 | 133 | 6.5 | 0 | -999 | -999 | -999 | -999 | -999 | -999 | -999 | -999 | 0 | -999 | 30.79 | 32.37 | 219.31 | 0.186 |
| 22 | 21 | 18 | 1955 | 1217 | 607 | 75 | 5.4 | 0 | -999 | -999 | -999 | -999 | -999 | -999 | -999 | -999 | 1 | -999 | 14.77 | 17.57 | 213.44 | 0.044 |
| 23 | 22 | 19 | 1956 | 209 | 1433 | 75 | 6.8 | 0 | -999 | -999 | -999 | -999 | -999 | -999 | -999 | -999 | 0 | -999 | 121.22 | 122.27 | 213.44 | 0.045 |
| 24 | 23 | 20 | 1957 | 322 | 1944 | 150 | 5.28 | 2 | -999 | -999 | -999 | -999 | -999 | -999 | -999 | -999 | 0 | -999 | 11.13 | 13.7 | 874.72 | 0.086 |
| 25 | 24 | 21 | 1960 | 120 | 326 | 135 | 5 | 0 | -999 | -999 | -999 | -999 | -999 | -999 | -999 | -999 | 0 | -999 | 8.01 | 10.91 | 198.77 | 0.039 |

DATA_PGA_050

***Figura 6: Hoja de cálculo con datos de entrada de la PEER.***

### 3.1.2 Descripción del modelo.

Lo que se busca es utilizar los tres modelos que son TensorFlow, PyTorch y Scikit-Learn para encontrar la mejor solución que se adapte a esta problemática.

La tipología de los modelos es una capa de entrada de 5 neuronas (debido a que tenemos 5 variables independientes), 3 capas ocultas de 64, 32, 16 neuronas y 1 capa de salida conformada de una neurona donde se verá reflejado el valor de la predicción del modelo de la red neuronal correspondiente a la aceleración del terreno.

### 3.1.3 Selección de datos de entrenamiento.

Se realizarán 100 épocas o iteraciones y se distribuirá la data de entrenamiento y prueba como se indica a continuación:

Se distribuirá la data total en un 80% (0.8) de data de entrenamiento y en un 20% (0.2) de data de prueba y luego se llegará a un resultado final y lo cuantificaremos a partir de unas métricas

estadísticas. Con relación a la ponderación anterior se hará un análisis de sensibilidad variando los porcentajes de distribución de la data de entrenamiento entre 70% y 90% y la data de prueba entre 30% y 10% respectivamente.

### 3.1.4 Entrenamiento de los modelos

Con el objetivo de mantener en igualdad de condiciones todas las librerías para poder hallar el mejor resultado, se utilizará la tipología indicada en el ítem 3.1.2, se realizarán 100 épocas o iteraciones y se distribuirá la data de entrenamiento y prueba como se indicó en el ítem 3.1.3 con sus respectivas funciones de activación.

Para las variables seleccionadas se muestra la matriz de correlación la cual permite entender el peso o la importancia que tienen estas variables con respecto a la variable que se va a predecir.

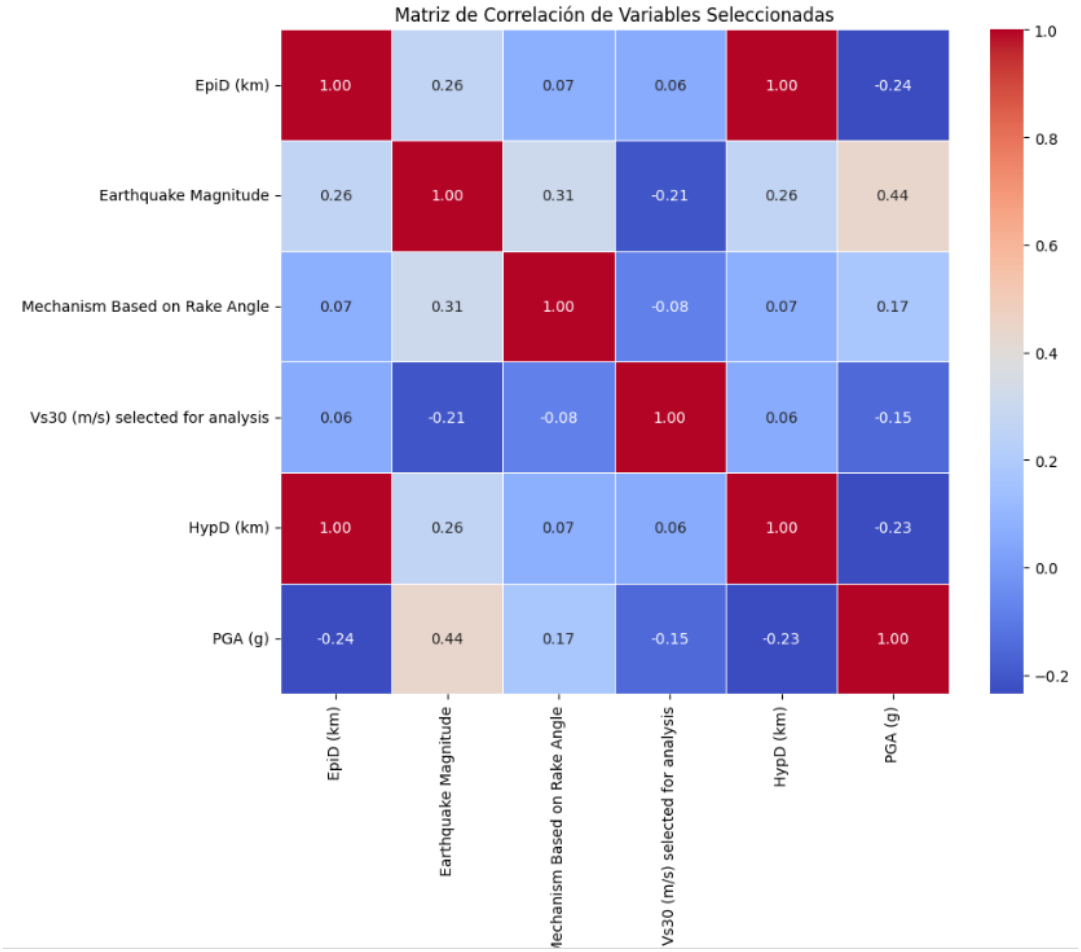



***Figura 7: Matriz de Correlación.***

**Librería TensorFlow**

En el anexo 1, 2, 3, 4 y 5 se muestran los detalles de entrenamiento de los modelos de Red Neuronal Artificial con la función de activación ReLU, tangente hiperbólica y sigmoide (logística), con data de validación/prueba 80% 20%, 70% 30% y 90% 10% y las características planteadas anteriormente, del cual se destaca los siguientes aspectos:

1. Se importa la librería TensorFlow.
2. Se carga el archivo CSV.
3. Se seleccionan las variables relevantes.
4. Se divide los datos en características (x) y etiquetas (y).
5. Se dividen los datos en conjuntos de entrenamiento y prueba.
6. Se escalan los datos.
7. Se definen los modelos de red neuronal.
8. Se compila el modelo.
9. Se entrena el modelo.
10. Se evalúa el modelo en el conjunto de prueba.
11. Se calculan las predicciones.
12. Se guarda el modelo.

**Librería PyTorch**

En los anexos 6, 7, 8, 9 y 10 se presenta el entrenamiento de los modelos de Red Neuronal Artificial con la función de activación ReLU, tangente hiperbólica y sigmoide (logística), con data de validación/prueba 80% 20%, 70% 30% y 90% 10% y las características planteadas anteriormente, del cual se destaca los siguientes aspectos:

1. Se importa la librería torch.nn.
2. Se carga el archivo CSV.
3. Se seleccionan las variables relevantes.
4. Se divide los datos en características (x) y etiquetas (y).
5. Se dividen los datos en conjuntos de entrenamiento y prueba.
6. Se escalan los datos.

7. Se convierten los datos a tensores de PyTorch.
8. Se define el modelo de red neuronal con PyTorch.
9. Se inicializa el modelo.
10. Se define la función de pérdida y el optimizador.
11. Se entrena el modelo.
12. Se evalúa el modelo en el conjunto de prueba.
13. Se convierten las predicciones y etiquetas a arrays de numpy.
14. Se guarda el modelo.

**Librería Scikit-Learn**

En los anexos 11, 12, 13, 14 y 15 se muestran los detalles de entrenamiento de los modelos de Red Neuronal Artificial con la función de activación ReLU, tangente hiperbólica y sigmoide (logística), con data de validación/prueba 80% 20%, 70% 30% y 90% 10% y las características planteadas anteriormente, del cual se destaca los siguientes aspectos:

1. Se importa la librería numpy y pandas.
2. Se carga el archivo CSV.
3. Se seleccionan las variables relevantes.
4. Se divide los datos en características (x) y etiquetas (y).
5. Se dividen los datos en conjuntos de entrenamiento y prueba.
6. Se escalan los datos.
7. Se definen los modelos de red neuronal con Scikit-Learn.
8. Se entrena el modelo.
9. Se realizan las predicciones en el conjunto de prueba.
10. Se guarda el modelo.

### 3.1.5 Validación con la data de prueba.

Para determinar cuál de los modelos previamente descritos predice de mejor manera la aceleración del terreno, se tomó de la data original un 10%, 20% y 30% de los datos no usados en la fase de entrenamiento, con los valores de aceleración conocidos y los valores de aceleración predichos por los modelos de RNA.

De estos resultados se extraerán valores como el coeficiente de determinación $R^2$, el error cuadrático medio (MSE), la raíz del error cuadrático medio (RMSE), el error logarítmico cuadrático medio (RMSLE) y el error absoluto medio (MAE) que serán presentados y analizados en el capítulo 4.

# 4. CAPÍTULO IV: ANÁLISIS DE RESULTADOS

## 4.1 Coeficiente de Determinación ($R^2$).

El coeficiente de determinación $R^2$ es una medida estadística que indica que tan bueno es el ajuste de la predicción a los valores reales. El $R^2$ proporciona una medida de cuánto mejor se ajusta el modelo a los datos en comparación con un modelo base que simplemente predice la media de la variable dependiente.

El valor de $R^2$ puede variar entre 0 y 1, donde:

- Un valor de 1 indica que el modelo explica perfectamente toda la variabilidad de los datos.
- Un valor de 0 indica que el modelo no explica ninguna variabilidad y es esencialmente equivalente a usar solo la media de la variable dependiente para hacer predicciones.
- Valores intermedios indican la proporción de variabilidad explicada por el modelo en relación con la variabilidad total de los datos.

Cuanto más cercano esté el valor de $R^2$ a 1, mejor se ajusta el modelo a los datos y más explicativo es el modelo de la variabilidad observada en la variable dependiente.

La fórmula del coeficiente de determinación ($R^2$) se calcula de la siguiente manera:

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}}$$

Donde:

$SS_{res}$ es la suma de los cuadrados de los residuos (la diferencia entre los valores observados y los valores predichos).

$SS_{tot}$ es la suma total de los cuadrados, que mide la variabilidad total de los datos.

El coeficiente de determinación se calcula como 1 menos la proporción de la variabilidad que no es explicada por el modelo respecto a la variabilidad total de los datos. Un $R^2$ de = 1 indica que los valores predichos se ajustan perfectamente a los valores reales y en la medida en que el $R^2$ va descendiendo se empieza a producir mayor dispersión entre los valores predichos y los valores reales.

De una de las fuentes consultadas **[16]**, a continuación, se presenta una tabla explicativa de los valores del $R^2$ aceptables de Pearson:

***Tabla 1: Valores aceptables del $R^2$.***

| Valores del $R^2$ | Descripción |
|---|---|
| 0 – 0,25 | Escasa o nula. |
| 0,26 – 0,50 | Débil. |
| 0,51 – 0,75 | Entre moderada y fuerte. |
| 0,76 – 1,00 | Entre fuerte y perfecta. |

### 4.2 Error Cuadrático Medio (MSE).

El error cuadrático medio (MSE) es una medida comúnmente utilizada para evaluar la calidad de un modelo de regresión. Se calcula tomando la diferencia entre los valores predichos por el modelo y los valores reales en el conjunto de datos, elevando al cuadrado estas diferencias y luego tomando el promedio de estos valores cuadrados.

Matemáticamente, el MSE se define como:

$$ECM = \frac{1}{n}\sum_{i=1}^{n}(y - \hat{y})^2$$

Donde y es el valor de salida real conocido en el proceso de aprendizaje y $\bar{y}$ son los valores predichos con el modelo de la red neuronal.

El MSE mide el promedio de los cuadrados de los errores de predicción del modelo. Cuanto menor sea el valor del MSE, mejor será el rendimiento del modelo, ya que indica que las predicciones del modelo están más cerca de los valores reales. Por lo tanto, el MSE se utiliza para comparar diferentes modelos de regresión y determinar cuál tiene un rendimiento predictivo superior.

### 4.3 Raíz del Error Cuadrático Medio (RMSE).

La raíz del error cuadrático medio (RMSE) es una medida de la dispersión de los errores en un modelo de regresión, similar al Error Cuadrático Medio (MSE), pero expresado en la misma unidad que la variable de respuesta original, lo que facilita su interpretación.

El RMSE se calcula tomando la raíz cuadrada del MSE. Matemáticamente, se define como:

$$RMSE = \sqrt{MSE}$$

Por lo tanto, el RMSE es simplemente la raíz cuadrada del promedio de los cuadrados de los errores de predicción. Al igual que el MSE, el RMSE proporciona una medida de la diferencia entre los valores predichos por el modelo y los valores reales, pero en una escala que es interpretable en las mismas unidades que la variable de respuesta.

Un RMSE más bajo indica que el modelo tiene un mejor ajuste a los datos y que las predicciones del modelo están más cerca de los valores reales en promedio. Por lo tanto, al comparar diferentes modelos de regresión, se prefiere aquel que tenga el RMSE más bajo, ya que indica un mejor rendimiento predictivo en términos de la dispersión de los errores.

### 4.4 Error Logarítmico Cuadrático Medio (RMSLE).

El error logarítmico cuadrático medio (RMSLE) es una métrica de evaluación comúnmente utilizada en problemas de regresión cuando las variables objetivo tienen una escala muy amplia o cuando se desea penalizar de manera menos severa las diferencias en predicciones que están en las partes extremas de la distribución.

El RMSLE se calcula tomando el logaritmo natural (o cualquier otro logaritmo) de los valores predichos y reales antes de calcular el error cuadrático medio. Luego, se calcula el error cuadrático medio entre los logaritmos de los valores predichos y reales, y finalmente se toma la raíz cuadrada de este valor.

Matemáticamente, el RMSLE se define como:

$$RMSLE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\log(y_i + 1) - \log(\hat{y}_i + 1))^2}$$

Donde $y_i$ es el valor de salida real conocido en el proceso de aprendizaje y $\bar{y}_i$ son los valores predichos con el modelo de la red neuronal.

El RMSLE penaliza menos los errores en las predicciones que están en las partes extremas de la distribución, ya que los valores logarítmicos reducen la escala de los valores originales. Esto puede ser útil en situaciones donde las diferencias entre predicciones en las partes extremas de la distribución son menos importantes que las diferencias en las partes centrales.

Un RMSLE más bajo indica un mejor rendimiento del modelo, donde las predicciones logarítmicas están más cerca de los valores reales. Por lo tanto, al comparar diferentes modelos de regresión, se prefiere aquel que tenga el RMSLE más bajo.

### 4.5 Error Absoluto Medio (MAE).

El error absoluto medio (MAE) es una medida de la precisión de un modelo de regresión que calcula la magnitud promedio de los errores en las predicciones, sin tener en cuenta su dirección. El MAE se calcula tomando el valor absoluto de la diferencia entre cada valor predicho por el modelo y su correspondiente valor real, y luego calculando el promedio de estas diferencias absolutas.

Matemáticamente, el MAE se define como:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

Donde $y_i$ es el valor de salida real conocido en el proceso de aprendizaje y $\bar{y}_i$ son los valores predichos con el modelo de la red neuronal.

El MAE proporciona una medida de cuánto, en promedio, las predicciones del modelo difieren de los valores reales. Un MAE más bajo indica un mejor rendimiento del modelo, ya que las predicciones del modelo están más cerca de los valores reales en promedio.

El MAE es útil porque es fácil de interpretar representa la magnitud promedio de los errores en las unidades originales de la variable dependiente. Por lo tanto, al comparar diferentes modelos de regresión, se prefiere aquel que tenga el MAE más bajo, ya que indica una menor discrepancia entre las predicciones del modelo y los valores reales.

### 4.6 Evaluación de la Red Neuronal.

Con el propósito de comparar las métricas de evaluación utilizadas en este estudio, se presentarán sus valores asociados a las tres librerías utilizadas, junto con las funciones de activación correspondientes y las distribuciones de entrenamiento y prueba.

A continuación, se muestra una tabla comparativa de los valores de las métricas de las 3 librerías utilizadas en este estudio, tomando en cuenta la función de activación y la distribución de entrenamiento y prueba.

***Tabla 2: Valores de las métricas estadísticas de ajuste del RNA.***

| Librería | Función | Distribución | R2 | MSE | RMSE | RMSLE | MAE |
|---|---|---|---|---|---|---|---|
| TensorFlow | ReLU | 70% - 30% | 0.71006 | 0.00089 | 0.02991 | 0.01374 | 0.01214 |
| | | 80% - 20% | 0.72740 | 0.00095 | 0.03094 | 0.02668 | 0.01344 |
| | | 90% - 10% | 0.70205 | 0.00091 | 0.03032 | 0.01391 | 0.01190 |
| | Tanh | 80% - 20% | 0.69240 | 0.00092 | 0.03044 | 0.01404 | 0.01299 |
| | Sigmoide | 80% - 20% | 0.68510 | 0.00094 | 0.03066 | 0.01413 | 0.01296 |
| PyTorch | ReLU | 70% - 30% | 0.71462 | 0.00088 | 0.02967 | 0.02967 | 0.01188 |
| | | 80% - 20% | 0.72740 | 0.00088 | 0.02979 | 0.01368 | 0.01128 |
| | | 90% - 10% | 0.71361 | 0.00088 | 0.02972 | 0.01362 | 0.01147 |
| | Tanh | 80% - 20% | 0.69240 | 0.00087 | 0.02960 | 0.01361 | 0.01203 |
| | Sigmoide | 80% - 20% | 0.68510 | 0.00090 | 0.03010 | 0.01386 | 0.01256 |
| Scikit-Learn | ReLU | 70% - 30% | 0.65097 | 0.00107 | 0.03282 | 0.01510 | 0.01407 |
| | | 80% - 20% | 0.67340 | 0.00097 | 0.03114 | 0.01437 | 0.01397 |
| | | 90% - 10% | 0.68471 | 0.00097 | 0.03119 | 0.01439 | 0.01495 |
| | Tanh | 80% - 20% | 0.54210 | 0.00123 | 0.03514 | 0.01632 | 0.01930 |
| | Sigmoide | 80% - 20% | 0.67340 | 0.00126 | 0.03553 | 0.01647 | 0.01940 |

Con relación a las métricas estadísticas MSE, RMSE, RMSLE y MAE se observan que los valores son muy bajos y similares entre sí en todos los casos para hacer una representación gráfica, a diferencia del coeficiente de correlación $R^2$ donde se muestra mayor diferencia en sus valores, lo

cual permite elaborar una gráfica que muestre su variación en función de la librería utilizada y la función de activación.

A continuación, se presenta un gráfico de barras de los valores del $R^2$ de Pearson de las 3 librerías utilizadas con las 3 funciones de activación que son ReLU, PyTorch y Scikit-Learn, en donde el color azul representa la librería TensorFlow, el color verde hace referencia a la librería PyTorch y el color rojo hace referencia a la librería Scikit-Learn.

En la columna número 1 de cada librería se representa la función de activación ReLU, la columna número 2 hace referencia a la función de activación tangente hiperbólica y la columna número 3 significa la función de activación Sigmoide.

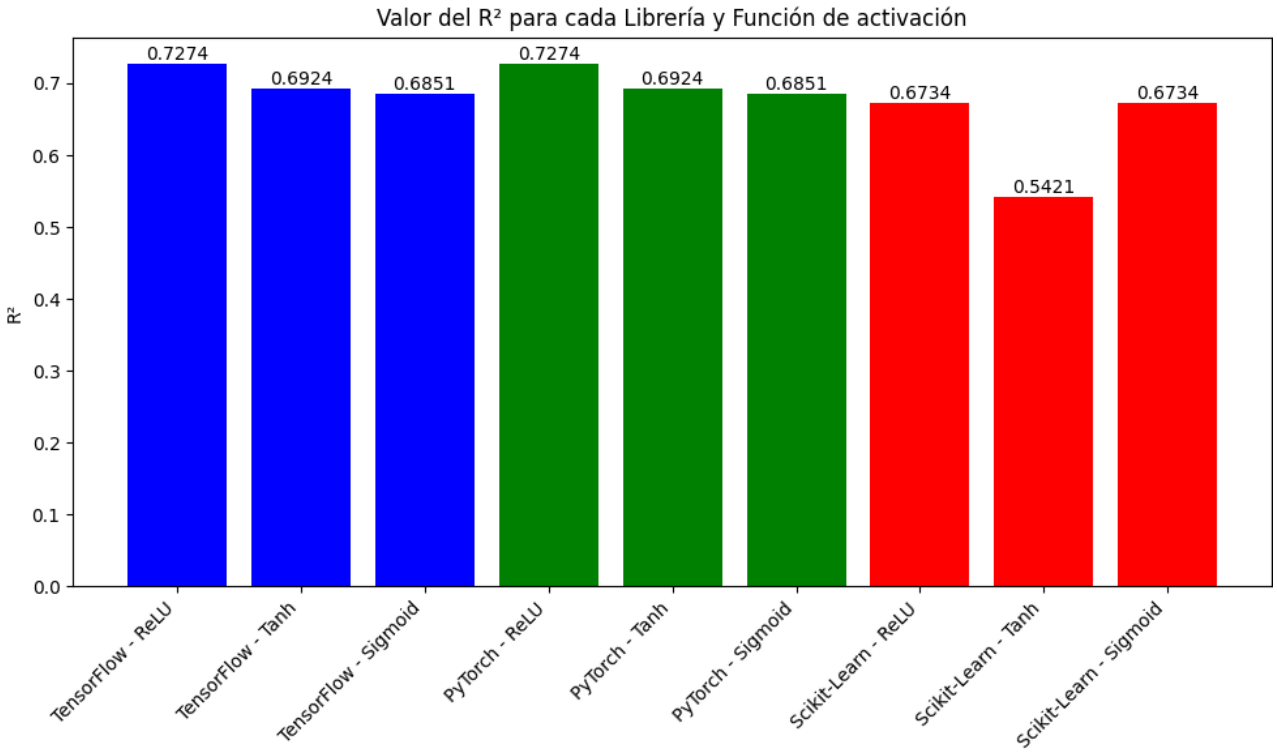



***Figura 8: Gráfico de barras de los valores del $R^2$ de cada librería y funciones de activación.***

La gráfica revela que el coeficiente de determinación $R^2$ varía según la función de activación utilizada en cada librería. En términos generales, se observa que la función de activación ReLU tiende a producir valores más altos de $R^2$ en todas las librerías evaluadas, lo que sugiere que ReLU podría ser una opción preferida en cuanto a ajuste se refiere en RNA asociados a este tipo de estudio.

Utilizando la función de activación ReLU, se destaca que las librerías TensorFlow y PyTorch exhiben un coeficiente de correlación idéntico de 0.73 para esta función de activación específica, a diferencia de la librería Scikit-Learn en la que se alcanzó un $R^2$ de 0.67.

Esto sugiere que tanto TensorFlow como PyTorch pueden ser consideradas opciones viables en estudios relacionados con datos sísmicos que empleen redes neuronales artificiales y utilicen la función de activación ReLU.

En la tabla 2 se colocaron de color verde los dos valores más bajos de las métricas MSE, RMSE, RMSLE Y MAE y en el $R^2$ se pintó de color amarillo los valores más altos del coeficiente de correlación a fin de detectar cual es la librería y la función de activación que presentan los mejores valores en el ajuste de la correlación.

Como se puede observar, la librería PyTorch con la función de activación ReLU es en donde se presenta el menor error y el mayor valor de $R^2$.

Considerando que la función de activación ReLU fue la que arrojó mejores resultados de $R^2$, solo con ella se hará el estudio de variabilidad de las distribuciones de entrenamiento y prueba del 70% - 30%, 80% - 20% y 90% - 10%.

Observando la tabla 2, los mayores coeficientes de correlaciones de $R^2$ se obtuvieron para el caso de entrenamiento y prueba en proporción a 80% - 20% estando por encima de la proporción 70% – 30% y por debajo de la proporción 90% - 10%.

En este tipo de problemas con data sísmica podría utilizarse como proporción de entrenamiento y prueba de 80% - 20%.

# 5. CAPÍTULO V: CONCLUSIONES Y RECOMENDACIONES

## 5.1 Conclusiones.

- Se ha ubicado una base de datos confiable con 21,198 registros de la PEER, relacionado con el movimiento del terreno.
- Se realizó el estudio de correlación de variables para determinar el peso de cada una de ellas y seleccionar las variables más importantes para determinar la aceleración máxima del terreno (PGA (g)), las cuales fueron: la distancia epicentral (EpiD (km)), la magnitud del sismo (Earthquake Magnitud), el mecanismo de falla (Mechanism Base on Rake Angle) y la Tipología de suelo utilizando la velocidad de onda de corte (Vs30 m/s).
- Se utilizó la técnica de redes neuronales artificiales con las librerías TensorFlow, PyTorch y Scikit-Learn y las funciones de activación Sigmoide (logística), tangente hiperbólica (tanh) y ReLU (Rectified Linear Unit) para determinar la aceleración del terreno.
- Al utilizar las redes neuronales artificiales con las librerías TensorFlow, PyTorch y scikit-Learn y las funciones de activación Sigmoide (logística), tangente hiperbólica (tanh) y ReLU (Rectified Linear Unit), se obtuvieron valores de $R^2$ de 0.72740, 0.72740 y 0.67340 respectivamente.
- De las librerías y funciones de activación evaluadas, se recomienda usar la librería PyTorch con la función de activación ReLU en los problemas que tienen relación con data sísmica, debido a que produjeron los mayores valores de $R^2$ y los valores menores en las métricas MSE, RMSE, RMSLE y MAE.

## 5.2 Recomendaciones.

- Con miras a mejorar el $R^2$, se recomienda experimentar con diferentes tipologías, números de capas y neuronas por cada capa, así como utilizar diferentes funciones de activación y librerías.
- Se puede realizar el estudio de correlación discretizando la data por tipología de suelo o por rango de distancias epicentrales o por niveles de aceleración del terreno, por cuanto en este estudio se hizo una correlación de forma global.

- Se recomienda experimentar con otros modelos alternativos de machine learning como regresiones lineales o no lineales, XGBoost y LightGBM, los cuales se han visto producen buenos resultados en problemas de la industria petrolera.

## BIBLIOGRAFÍA


1. Bozorgnia Y., Stewart J.P. (eds.) (2020). Data resources for NGA-subduction project, PEER Report No. 2020/02, Pacific Earthquake Engineering Research Center, University of California, Berkeley, CA.
2. Campbell K.W., Bozorgnia Y. (2007). Campbell-Bozorgnia NGA ground motion relations for the geometric mean horizontal component of peak and spectral ground motion parameters, PEER Report No. 2007/02, Pacific Earthquake Engineering Research Center, University of California, Berkeley, CA.
3. Campbell K.W., Hashash Y.M.A., Kim B., Kottke A.R., Rathje E.M., Silva W.J., Stewart J.P. (2014). Reference-rock site conditions for Central and Eastern North America: Part II – Attenuation (kappa) definitionPEER Report No. 2014/12, Pacific Earthquake Engineering Research Center, University of California, Berkeley, CA.
4. Chou C, U. C. (2000). An evaluation of seismic energy demand: an attenuation approach. California: Berkeley.Financial Engineering (Proceedings of the Fourth International Conference on Neural Networks in the Capital MArkets, NNCM-96), pp. 339-412. 1997.
5. Goulet C.G., Bozorgnia Y., Abrahamson N.A., Kuehn H., Al Atik L., Youngs R.R., Graves R.W., Atkinson G. (2018). Central and Eastern North America ground-motion characterization: NGA-East final report, PEER Report No. 2018/08, Pacific Earthquake Engineering Research Center, University of California, Berkeley, CA.
6. Martín J.D. “Aplicación de redes neuro-difusas en problemas de modelización y clasificación”. Proyecto final de carrera, Ingeniería Eelctrónica, Universitat de València, 1999.
7. Parker G.A., Stewart J.P., Boore D.M., Atkinson G.M., Hassani B. (2020). NGA-Subduction global ground-motion models with regional adjustment factors, PEER Report No. 2020/03, Pacific Earthquake Engineering Research Center, University of California, Berkeley, CA.
8. Roy, S., Shynk, J.J. “Analysis of the Momentum LMS Algorithm”. IEEE Transactions on Acoustics, Speech and Signal Processing, vol 38, n˚ 12, pp 2088-2098, Diciembre 1990.
9. Silipo, R., Bortolan, G., Marchesi, C. “Supervised and Unsupervised Learning for Diagnostic ECG Classification”. IEEE Engineering in Medicine and Biology, paper 1054, 1996.
10. Soria, E. et alt. “Application of an Artificial Neural Network with a Piecewise-Linear Activation Function to Determine the End of the T- Wave in an ECG”. World Congress on Medical Physics & Biomedical Engineering, Niza, Septiembre 1997.
11. Tothong P., Cornell C.A. (2006). Probabilistic seismic demand analysis using advanced ground motion intensity measures, attenuation relationships, and near-fault effects, PEER Report No. 2006/11, Pacific Earthquake Engineering Research Center, University of California, Berkeley, CA

12. Vonk, E., Jain, L.C., Johnson, R.P. ”Automatic Generation of Neural Network Architecture Using Evolutionary Computation”. Advances in Fuzzy Systems, World Scientific, 1997.
13. Wan, E. “Finite Impulse Response Neural Networks with Applications in Time Series Prediction”. PhD dissertation, Universidad de Stanford, 1993.
14. Weigend, A.S., “Data Mining in Finance: Rreport from the Post- NNCM-96 Workshop on Teaching Computer Intensive Methods for Financial Modeling and Data Analysis.” Decision Technologies for Financial Engineering (Proceedings of the Fourth International Conference on Neural Networks in the Capital MArkets, NNCM-96), pp. 339-412. 1997.
15. Widrow, B., Hoff, M.E. “Adaptive Switching Circuits”. Neurocomputing: Foundations of Research, pp 126-134, MIT Press, 1989.
16. Martínez R., Pérez A., Tuya L. y Cánovas A. El Coeficiente de Correlación de los Rangos de Spearman. Caracterización, Rev Haban cienc méd v.8 n.2, 2009 Disponible en: http://scielo.sld.cu/scielo.php?scri pt=sci_arttext&pid=S1729-519X2009000200017&fbclid=Iw AR3tyvaGEcLg54BeR7DmzUED LxfFOwlszyxzHoaJZ_hECNG1SCi5z9GT gg
17. Monterrubio-Velasco, M., Callaghan, S., Modesto, D. *et al.* A machine learning estimator trained on synthetic data for real-time earthquake ground-shaking predictions in Southern California. *Commun Earth Environ* **5**, 258 (2024). https://doi.org/10.1038/s43247-024-01436-1
18. Klimasewski Alexis. 2020: Comparing artificial neural networks with traditional ground-motion models for small magnitude earthquakes in southern California. https://scholarsbank.uoregon.edu/xmlui/bitstream/handle/1794/26187/Klimasewski_oregon_0171N_12920.pdf?sequence=1
19. Hanping Hong, Taojun Liu, Chien-Shen Lee. 2012: Observations on the application of artificial neural network to predicting ground motion measures. Earthquake Science, 25(2): 161-175. https://www.equsci.org.cn/article/doi/10.1007/s11589-012-0843-5?pageType=en

## ANEXOS

### Anexo 1: Librería TensorFlow - Función ReLU – 80% entrenamiento – 20% prueba.

```
# Importamos las librerías necesarias
import numpy as np
import pandas as pd
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
```

```
# Cargamos el archivo CSV
url = '/content/DATA_PGA_050.csv'  # Ruta del archivo CSV
df = pd.read_csv(url)

# Seleccionamos solo las variables relevantes
df_selected = df[['EpiD (km)', 'Earthquake Magnitude', 'Mechanism Based on
Rake Angle', 'Vs30 (m/s) selected for analysis','HypD (km)', 'PGA (g)']]

# Mostramos las primeras filas del dataframe para verificar que se haya
cargado correctamente
print(df_selected.head())

# Dividimos los datos en características (X) y etiquetas (y)
X = df_selected.drop('PGA (g)', axis=1).values
y = df_selected['PGA (g)'].values

# Dividimos los datos en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Escalamos los datos
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Definimos el modelo de red neuronal
model = tf.keras.Sequential([
    tf.keras.layers.Dense(5, activation='relu',
input_shape=(X_train.shape[1],)),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(32, activation='relu'),
    tf.keras.layers.Dense(16, activation='relu'),
    tf.keras.layers.Dense(1)
])

# Compilamos el modelo
model.compile(optimizer='adam', loss='mean_squared_error')

# Entrenamos el modelo
model.fit(X_train, y_train, epochs=100, batch_size=32, verbose=1)

# Evaluamos el modelo en el conjunto de prueba
mse = model.evaluate(X_test, y_test, verbose=0)
print('Mean Squared Error on Test Set:', mse)
```

```
# Calculamos las predicciones
y_pred = model.predict(X_test).flatten()

# Calculamos el coeficiente de determinación R cuadrado (R²)
r2 = r2_score(y_test, y_pred)
print('Coeficiente de determinación R cuadrado (R²):', r2)

# Guardamos el modelo
model.save('modelo.h5')
```

**Anexo 2: Librería TensorFlow - Función ReLU – 70% entrenamiento – 30% prueba.**

```
# Importamos las librerías necesarias
import numpy as np
import pandas as pd
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score

# Cargamos el archivo CSV
url = '/content/DATA_PGA_050.csv'  # Ruta del archivo CSV
df = pd.read_csv(url)

# Seleccionamos solo las variables relevantes
df_selected = df[['EpiD (km)', 'Earthquake Magnitude', 'Mechanism Based on
Rake Angle', 'Vs30 (m/s) selected for analysis','HypD (km)', 'PGA (g)']]

# Mostramos las primeras filas del dataframe para verificar que se haya
cargado correctamente
print(df_selected.head())

# Dividimos los datos en características (X) y etiquetas (y)
X = df_selected.drop('PGA (g)', axis=1).values
y = df_selected['PGA (g)'].values

# Dividimos los datos en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

# Escalamos los datos
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
```

```
X_test = scaler.transform(X_test)

# Definimos el modelo de red neuronal
model = tf.keras.Sequential([
    tf.keras.layers.Dense(5, activation='relu',
input_shape=(X_train.shape[1],)),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(32, activation='relu'),
    tf.keras.layers.Dense(16, activation='relu'),
    tf.keras.layers.Dense(1)
])

# Compilamos el modelo
model.compile(optimizer='adam', loss='mean_squared_error')

# Entrenamos el modelo
model.fit(X_train, y_train, epochs=100, batch_size=32, verbose=1)

# Evaluamos el modelo en el conjunto de prueba
mse = model.evaluate(X_test, y_test, verbose=0)
print('Mean Squared Error on Test Set:', mse)

# Calculamos las predicciones
y_pred = model.predict(X_test).flatten()

# Calculamos el coeficiente de determinación R cuadrado (R²)
r2 = r2_score(y_test, y_pred)
print('Coeficiente de determinación R cuadrado (R²):', r2)

# Guardamos el modelo
model.save('modelo.h5')
```

**Anexo 3: Librería TensorFlow - Función ReLU – 90% entrenamiento – 10% prueba.**

```
# Importamos las librerías necesarias
import numpy as np
import pandas as pd
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score

# Cargamos el archivo CSV
url = '/content/DATA_PGA_050.csv'  # Ruta del archivo CSV
```

```
df = pd.read_csv(url)

# Seleccionamos solo las variables relevantes
df_selected = df[['EpiD (km)', 'Earthquake Magnitude', 'Mechanism Based on
Rake Angle', 'Vs30 (m/s) selected for analysis','HypD (km)', 'PGA (g)']]

# Mostramos las primeras filas del dataframe para verificar que se haya
cargado correctamente
print(df_selected.head())

# Dividimos los datos en características (X) y etiquetas (y)
X = df_selected.drop('PGA (g)', axis=1).values
y = df_selected['PGA (g)'].values

# Dividimos los datos en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1,
random_state=42)

# Escalamos los datos
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Definimos el modelo de red neuronal
model = tf.keras.Sequential([
    tf.keras.layers.Dense(5, activation='relu',
input_shape=(X_train.shape[1],)),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(32, activation='relu'),
    tf.keras.layers.Dense(16, activation='relu'),
    tf.keras.layers.Dense(1)
])

# Compilamos el modelo
model.compile(optimizer='adam', loss='mean_squared_error')

# Entrenamos el modelo
model.fit(X_train, y_train, epochs=100, batch_size=32, verbose=1)

# Evaluamos el modelo en el conjunto de prueba
mse = model.evaluate(X_test, y_test, verbose=0)
print('Mean Squared Error on Test Set:', mse)

# Calculamos las predicciones
y_pred = model.predict(X_test).flatten()
```

```
# Calculamos el coeficiente de determinación R cuadrado (R²)
r2 = r2_score(y_test, y_pred)
print('Coeficiente de determinación R cuadrado (R²):', r2)

# Guardamos el modelo
model.save('modelo.h5')
```

**Anexo 4: Librería TensorFlow - Función Tangente Hiperbólica – 80% entrenamiento – 20% prueba.**

```
# Importamos las librerías necesarias
import numpy as np
import pandas as pd
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score

# Cargamos el archivo CSV
url = '/content/DATA_PGA_050.csv'  # Ruta del archivo CSV
df = pd.read_csv(url)

# Seleccionamos solo las variables relevantes
df_selected = df[['EpiD (km)', 'Earthquake Magnitude', 'Mechanism Based on
Rake Angle', 'Vs30 (m/s) selected for analysis','HypD (km)', 'PGA (g)']]

# Mostramos las primeras filas del dataframe para verificar que se haya
cargado correctamente
print(df_selected.head())

# Dividimos los datos en características (X) y etiquetas (y)
X = df_selected.drop('PGA (g)', axis=1).values
y = df_selected['PGA (g)'].values

# Dividimos los datos en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Escalamos los datos
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

```
# Definimos el modelo de red neuronal
model = tf.keras.Sequential([
    tf.keras.layers.Dense(5, activation='tanh',
input_shape=(X_train.shape[1],)),
    tf.keras.layers.Dense(64, activation='tanh'),
    tf.keras.layers.Dense(32, activation='tanh'),
    tf.keras.layers.Dense(16, activation='tanh'),
    tf.keras.layers.Dense(1)
])

# Compilamos el modelo
model.compile(optimizer='adam', loss='mean_squared_error')

# Entrenamos el modelo
model.fit(X_train, y_train, epochs=100, batch_size=32, verbose=1)

# Evaluamos el modelo en el conjunto de prueba
mse = model.evaluate(X_test, y_test, verbose=0)
print('Mean Squared Error on Test Set:', mse)

# Calculamos las predicciones
y_pred = model.predict(X_test).flatten()

# Calculamos el coeficiente de determinación R cuadrado (R²)
r2 = r2_score(y_test, y_pred)
print('Coeficiente de determinación R cuadrado (R²):', r2)

# Guardamos el modelo
model.save('modelo.h5')
```

**Anexo 5: Librería TensorFlow - Función Sigmoide– 80% entrenamiento – 20% prueba.**

```
# Importamos las librerías necesarias
import numpy as np
import pandas as pd
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score

# Cargamos el archivo CSV
url = '/content/DATA_PGA_050.csv'  # Ruta del archivo CSV
df = pd.read_csv(url)
```

```
# Seleccionamos solo las variables relevantes
df_selected = df[['EpiD (km)', 'Earthquake Magnitude', 'Mechanism Based on
Rake Angle', 'Vs30 (m/s) selected for analysis','HypD (km)', 'PGA (g)']]

# Mostramos las primeras filas del dataframe para verificar que se haya
cargado correctamente
print(df_selected.head())

# Dividimos los datos en características (X) y etiquetas (y)
X = df_selected.drop('PGA (g)', axis=1).values
y = df_selected['PGA (g)'].values

# Dividimos los datos en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Escalamos los datos
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Definimos el modelo de red neuronal
model = tf.keras.Sequential([
    tf.keras.layers.Dense(5, activation='sigmoid',
input_shape=(X_train.shape[1],)),
    tf.keras.layers.Dense(64, activation='sigmoid'),
    tf.keras.layers.Dense(32, activation='sigmoid'),
    tf.keras.layers.Dense(16, activation='sigmoid'),
    tf.keras.layers.Dense(1)
])

# Compilamos el modelo
model.compile(optimizer='adam', loss='mean_squared_error')

# Entrenamos el modelo
model.fit(X_train, y_train, epochs=100, batch_size=32, verbose=1)

# Evaluamos el modelo en el conjunto de prueba
mse = model.evaluate(X_test, y_test, verbose=0)
print('Mean Squared Error on Test Set:', mse)

# Calculamos las predicciones
y_pred = model.predict(X_test).flatten()
```

```
# Calculamos el coeficiente de determinación R cuadrado (R²)
r2 = r2_score(y_test, y_pred)
print('Coeficiente de determinación R cuadrado (R²):', r2)

# Guardamos el modelo
model.save('modelo.h5')
```

**Anexo 6: Librería PyTorch - Función ReLU – 80% entrenamiento – 20% prueba.**

```
import numpy as np
import pandas as pd
import torch
import torch.nn as nn
import torch.optim as optim
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, r2_score

# Cargamos el archivo CSV
url = '/content/DATA_PGA_050.csv'  # Ruta del archivo CSV
df = pd.read_csv(url)

# Seleccionamos solo las variables relevantes
df_selected = df[['EpiD (km)', 'Earthquake Magnitude', 'Mechanism Based on
Rake Angle', 'Vs30 (m/s) selected for analysis','HypD (km)', 'PGA (g)']]

# Dividimos los datos en características (X) y etiquetas (y)
X = df_selected.drop('PGA (g)', axis=1).values
y = df_selected['PGA (g)'].values

# Dividimos los datos en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Escalamos los datos
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Convertimos los datos a tensores de PyTorch
X_train_tensor = torch.tensor(X_train, dtype=torch.float32)
y_train_tensor = torch.tensor(y_train, dtype=torch.float32)
X_test_tensor = torch.tensor(X_test, dtype=torch.float32)
y_test_tensor = torch.tensor(y_test, dtype=torch.float32)
```

```
# Definimos el modelo de red neuronal con PyTorch
class NeuralNetwork(nn.Module):
    def __init__(self, input_size):
        super(NeuralNetwork, self).__init__()
        self.fc1 = nn.Linear(input_size, 64)
        self.fc2 = nn.Linear(64, 32)
        self.fc3 = nn.Linear(32, 16)
        self.fc4 = nn.Linear(16, 1)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.relu(self.fc1(x))
        x = self.relu(self.fc2(x))
        x = self.relu(self.fc3(x))
        x = self.fc4(x)
        return x

# Inicializamos el modelo
model = NeuralNetwork(X_train.shape[1])

# Definimos la función de pérdida y el optimizador
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Entrenamos el modelo
batch_size = 64
epochs = 100
for epoch in range(epochs):
    for i in range(0, len(X_train_tensor), batch_size):
        inputs = X_train_tensor[i:i+batch_size]
        labels = y_train_tensor[i:i+batch_size]

        # Forward pass
        outputs = model(inputs)
        loss = criterion(outputs, labels.view(-1, 1))

        # Backward pass y optimización
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    # Imprimir el progreso
    if (epoch+1) % 10 == 0:
        print(f'Epoch [{epoch+1}/{epochs}], Loss: {loss.item():.4f}')
```

```
# Evaluamos el modelo en el conjunto de prueba
with torch.no_grad():
    y_pred_tensor = model(X_test_tensor)
    mse = criterion(y_pred_tensor, y_test_tensor.view(-1, 1))
    print('Mean Squared Error on Test Set:', mse.item())

    # Convertimos las predicciones y etiquetas a arrays de numpy
    y_pred = y_pred_tensor.numpy().flatten()
    y_test = y_test_tensor.numpy()

    # Calculamos el coeficiente de determinación R² (R cuadrado)
    r2 = r2_score(y_test, y_pred)
    print('Coeficiente de determinación R²:', r2)

# Guardamos el modelo (esto es opcional en PyTorch)
torch.save(model.state_dict(), 'modelo.pth')
```

**Anexo 7: Librería PyTorch - Función ReLU – 70% entrenamiento – 30% prueba.**

```
import numpy as np
import pandas as pd
import torch
import torch.nn as nn
import torch.optim as optim
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, r2_score

# Cargamos el archivo CSV
url = '/content/DATA_PGA_050.csv'  # Ruta del archivo CSV
df = pd.read_csv(url)

# Seleccionamos solo las variables relevantes
df_selected = df[['EpiD (km)', 'Earthquake Magnitude', 'Mechanism Based on
Rake Angle', 'Vs30 (m/s) selected for analysis','HypD (km)', 'PGA (g)']]

# Dividimos los datos en características (X) y etiquetas (y)
X = df_selected.drop('PGA (g)', axis=1).values
y = df_selected['PGA (g)'].values

# Dividimos los datos en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)
```

```
# Escalamos los datos
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Convertimos los datos a tensores de PyTorch
X_train_tensor = torch.tensor(X_train, dtype=torch.float32)
y_train_tensor = torch.tensor(y_train, dtype=torch.float32)
X_test_tensor = torch.tensor(X_test, dtype=torch.float32)
y_test_tensor = torch.tensor(y_test, dtype=torch.float32)

# Definimos el modelo de red neuronal con PyTorch
class NeuralNetwork(nn.Module):
    def __init__(self, input_size):
        super(NeuralNetwork, self).__init__()
        self.fc1 = nn.Linear(input_size, 64)
        self.fc2 = nn.Linear(64, 32)
        self.fc3 = nn.Linear(32, 16)
        self.fc4 = nn.Linear(16, 1)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.relu(self.fc1(x))
        x = self.relu(self.fc2(x))
        x = self.relu(self.fc3(x))
        x = self.fc4(x)
        return x

# Inicializamos el modelo
model = NeuralNetwork(X_train.shape[1])

# Definimos la función de pérdida y el optimizador
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Entrenamos el modelo
batch_size = 64
epochs = 100
for epoch in range(epochs):
    for i in range(0, len(X_train_tensor), batch_size):
        inputs = X_train_tensor[i:i+batch_size]
        labels = y_train_tensor[i:i+batch_size]

        # Forward pass
        outputs = model(inputs)
```

```
        loss = criterion(outputs, labels.view(-1, 1))

        # Backward pass y optimización
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    # Imprimir el progreso
    if (epoch+1) % 10 == 0:
        print(f'Epoch [{epoch+1}/{epochs}], Loss: {loss.item():.4f}')

# Evaluamos el modelo en el conjunto de prueba
with torch.no_grad():
    y_pred_tensor = model(X_test_tensor)
    mse = criterion(y_pred_tensor, y_test_tensor.view(-1, 1))
    print('Mean Squared Error on Test Set:', mse.item())

    # Convertimos las predicciones y etiquetas a arrays de numpy
    y_pred = y_pred_tensor.numpy().flatten()
    y_test = y_test_tensor.numpy()

    # Calculamos el coeficiente de determinación R² (R cuadrado)
    r2 = r2_score(y_test, y_pred)
    print('Coeficiente de determinación R²:', r2)

# Guardamos el modelo (esto es opcional en PyTorch)
torch.save(model.state_dict(), 'modelo.pth')
```

**Anexo 8: Librería PyTorch - Función ReLU – 90% entrenamiento – 10% prueba.**

```
import numpy as np
import pandas as pd
import torch
import torch.nn as nn
import torch.optim as optim
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, r2_score

# Cargamos el archivo CSV
url = '/content/DATA_PGA_050.csv'  # Ruta del archivo CSV
df = pd.read_csv(url)

# Seleccionamos solo las variables relevantes
```

```
df_selected = df[['EpiD (km)', 'Earthquake Magnitude', 'Mechanism Based on
Rake Angle', 'Vs30 (m/s) selected for analysis','HypD (km)', 'PGA (g)']]

# Dividimos los datos en características (X) y etiquetas (y)
X = df_selected.drop('PGA (g)', axis=1).values
y = df_selected['PGA (g)'].values

# Dividimos los datos en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1,
random_state=42)

# Escalamos los datos
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Convertimos los datos a tensores de PyTorch
X_train_tensor = torch.tensor(X_train, dtype=torch.float32)
y_train_tensor = torch.tensor(y_train, dtype=torch.float32)
X_test_tensor = torch.tensor(X_test, dtype=torch.float32)
y_test_tensor = torch.tensor(y_test, dtype=torch.float32)

# Definimos el modelo de red neuronal con PyTorch
class NeuralNetwork(nn.Module):
    def __init__(self, input_size):
        super(NeuralNetwork, self).__init__()
        self.fc1 = nn.Linear(input_size, 64)
        self.fc2 = nn.Linear(64, 32)
        self.fc3 = nn.Linear(32, 16)
        self.fc4 = nn.Linear(16, 1)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.relu(self.fc1(x))
        x = self.relu(self.fc2(x))
        x = self.relu(self.fc3(x))
        x = self.fc4(x)
        return x

# Inicializamos el modelo
model = NeuralNetwork(X_train.shape[1])

# Definimos la función de pérdida y el optimizador
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

```
# Entrenamos el modelo
batch_size = 64
epochs = 100
for epoch in range(epochs):
    for i in range(0, len(X_train_tensor), batch_size):
        inputs = X_train_tensor[i:i+batch_size]
        labels = y_train_tensor[i:i+batch_size]

        # Forward pass
        outputs = model(inputs)
        loss = criterion(outputs, labels.view(-1, 1))

        # Backward pass y optimización
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    # Imprimir el progreso
    if (epoch+1) % 10 == 0:
        print(f'Epoch [{epoch+1}/{epochs}], Loss: {loss.item():.4f}')

# Evaluamos el modelo en el conjunto de prueba
with torch.no_grad():
    y_pred_tensor = model(X_test_tensor)
    mse = criterion(y_pred_tensor, y_test_tensor.view(-1, 1))
    print('Mean Squared Error on Test Set:', mse.item())

    # Convertimos las predicciones y etiquetas a arrays de numpy
    y_pred = y_pred_tensor.numpy().flatten()
    y_test = y_test_tensor.numpy()

    # Calculamos el coeficiente de determinación R² (R cuadrado)
    r2 = r2_score(y_test, y_pred)
    print('Coeficiente de determinación R²:', r2)

# Guardamos el modelo (esto es opcional en PyTorch)
torch.save(model.state_dict(), 'modelo.pth')
```

**Anexo 9: Librería PyTorch - Función Tangente Hiperbólica – 80% entrenamiento – 20% prueba.**

```
import numpy as np
import pandas as pd
```

```
import torch
import torch.nn as nn
import torch.optim as optim
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, r2_score

# Cargamos el archivo CSV
url = '/content/DATA_PGA_050.csv'  # Ruta del archivo CSV
df = pd.read_csv(url)

# Seleccionamos solo las variables relevantes
df_selected = df[['EpiD (km)', 'Earthquake Magnitude', 'Mechanism Based on
Rake Angle', 'Vs30 (m/s) selected for analysis','HypD (km)', 'PGA (g)']]

# Dividimos los datos en características (X) y etiquetas (y)
X = df_selected.drop('PGA (g)', axis=1).values
y = df_selected['PGA (g)'].values

# Dividimos los datos en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Escalamos los datos
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Convertimos los datos a tensores de PyTorch
X_train_tensor = torch.tensor(X_train, dtype=torch.float32)
y_train_tensor = torch.tensor(y_train, dtype=torch.float32)
X_test_tensor = torch.tensor(X_test, dtype=torch.float32)
y_test_tensor = torch.tensor(y_test, dtype=torch.float32)

# Definimos el modelo de red neuronal con PyTorch
class NeuralNetwork(nn.Module):
    def __init__(self, input_size):
        super(NeuralNetwork, self).__init__()
        self.fc1 = nn.Linear(input_size, 64)
        self.fc2 = nn.Linear(64, 32)
        self.fc3 = nn.Linear(32, 16)
        self.fc4 = nn.Linear(16, 1)
        self.tanh = nn.Tanh()

    def forward(self, x):
```

```
        x = self.tanh(self.fc1(x))
        x = self.tanh(self.fc2(x))
        x = self.tanh(self.fc3(x))
        x = self.fc4(x)
        return x

# Inicializamos el modelo
model = NeuralNetwork(X_train.shape[1])

# Definimos la función de pérdida y el optimizador
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Entrenamos el modelo
batch_size = 64
epochs = 100
for epoch in range(epochs):
    for i in range(0, len(X_train_tensor), batch_size):
        inputs = X_train_tensor[i:i+batch_size]
        labels = y_train_tensor[i:i+batch_size]

        # Forward pass
        outputs = model(inputs)
        loss = criterion(outputs, labels.view(-1, 1))

        # Backward pass y optimización
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    # Imprimir el progreso
    if (epoch+1) % 10 == 0:
        print(f'Epoch [{epoch+1}/{epochs}], Loss: {loss.item():.4f}')

# Evaluamos el modelo en el conjunto de prueba
with torch.no_grad():
    y_pred_tensor = model(X_test_tensor)
    mse = criterion(y_pred_tensor, y_test_tensor.view(-1, 1))
    print('Mean Squared Error on Test Set:', mse.item())

    # Convertimos las predicciones y etiquetas a arrays de numpy
    y_pred = y_pred_tensor.numpy().flatten()
    y_test = y_test_tensor.numpy()

    # Calculamos el coeficiente de determinación R² (R cuadrado)
```

```
    r2 = r2_score(y_test, y_pred)
    print('Coeficiente de determinación R²:', r2)

# Guardamos el modelo (esto es opcional en PyTorch)
torch.save(model.state_dict(), 'modelo.pth')
```

**Anexo 10: Librería PyTorch - Función Sigmoide – 80% entrenamiento – 20% prueba.**

```
import numpy as np
import pandas as pd
import torch
import torch.nn as nn
import torch.optim as optim
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, r2_score

# Cargamos el archivo CSV
url = '/content/DATA_PGA_050.csv'  # Ruta del archivo CSV
df = pd.read_csv(url)

# Seleccionamos solo las variables relevantes
df_selected = df[['EpiD (km)', 'Earthquake Magnitude', 'Mechanism Based on
Rake Angle', 'Vs30 (m/s) selected for analysis','HypD (km)', 'PGA (g)']]

# Dividimos los datos en características (X) y etiquetas (y)
X = df_selected.drop('PGA (g)', axis=1).values
y = df_selected['PGA (g)'].values

# Dividimos los datos en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Escalamos los datos
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Convertimos los datos a tensores de PyTorch
X_train_tensor = torch.tensor(X_train, dtype=torch.float32)
y_train_tensor = torch.tensor(y_train, dtype=torch.float32)
X_test_tensor = torch.tensor(X_test, dtype=torch.float32)
y_test_tensor = torch.tensor(y_test, dtype=torch.float32)

# Definimos el modelo de red neuronal con PyTorch
```

```
class NeuralNetwork(nn.Module):
    def __init__(self, input_size):
        super(NeuralNetwork, self).__init__()
        self.fc1 = nn.Linear(input_size, 64)
        self.fc2 = nn.Linear(64, 32)
        self.fc3 = nn.Linear(32, 16)
        self.fc4 = nn.Linear(16, 1)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        x = self.sigmoid(self.fc1(x))
        x = self.sigmoid(self.fc2(x))
        x = self.sigmoid(self.fc3(x))
        x = self.fc4(x)
        return x

# Inicializamos el modelo
model = NeuralNetwork(X_train.shape[1])

# Definimos la función de pérdida y el optimizador
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Entrenamos el modelo
batch_size = 64
epochs = 100
for epoch in range(epochs):
    for i in range(0, len(X_train_tensor), batch_size):
        inputs = X_train_tensor[i:i+batch_size]
        labels = y_train_tensor[i:i+batch_size]

        # Forward pass
        outputs = model(inputs)
        loss = criterion(outputs, labels.view(-1, 1))

        # Backward pass y optimización
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    # Imprimir el progreso
    if (epoch+1) % 10 == 0:
        print(f'Epoch [{epoch+1}/{epochs}], Loss: {loss.item():.4f}')

# Evaluamos el modelo en el conjunto de prueba
```

```
with torch.no_grad():
    y_pred_tensor = model(X_test_tensor)
    mse = criterion(y_pred_tensor, y_test_tensor.view(-1, 1))
    print('Mean Squared Error on Test Set:', mse.item())

    # Convertimos las predicciones y etiquetas a arrays de numpy
    y_pred = y_pred_tensor.numpy().flatten()
    y_test = y_test_tensor.numpy()

    # Calculamos el coeficiente de determinación R² (R cuadrado)
    r2 = r2_score(y_test, y_pred)
    print('Coeficiente de determinación R²:', r2)

# Guardamos el modelo (esto es opcional en PyTorch)
torch.save(model.state_dict(), 'modelo.pth')
```

**Anexo 11: Librería Scikit-Learn - Función ReLU – 80% entrenamiento – 20% prueba.**

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPRegressor
from sklearn.metrics import mean_squared_error, r2_score

# Cargamos el archivo CSV
url = '/content/DATA_PGA_050.csv'  # Ruta del archivo CSV
df = pd.read_csv(url)

# Seleccionamos solo las variables relevantes
df_selected = df[['EpiD (km)', 'Earthquake Magnitude', 'Mechanism Based on
Rake Angle', 'Vs30 (m/s) selected for analysis','HypD (km)', 'PGA (g)']]

# Dividimos los datos en características (X) y etiquetas (y)
X = df_selected.drop('PGA (g)', axis=1).values
y = df_selected['PGA (g)'].values

# Dividimos los datos en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Escalamos los datos
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
```

```
X_test = scaler.transform(X_test)

# Definimos el modelo de red neuronal con Scikit-Learn
model = MLPRegressor(hidden_layer_sizes=(5, 64, 32, 16),
activation='relu', solver='adam', alpha=0.0001, max_iter=100,
random_state=42)

# Entrenamos el modelo
model.fit(X_train, y_train)

# Realizamos predicciones en el conjunto de prueba
y_pred = model.predict(X_test)

# Calculamos las métricas de evaluación
mse = mean_squared_error(y_test, y_pred)
print('Mean Squared Error on Test Set:', mse)

r2 = r2_score(y_test, y_pred)
print('Coeficiente de determinación R²:', r2)

# Guardamos el modelo (esto es opcional en Scikit-Learn)
import joblib
joblib.dump(model, 'modelo.pkl')
```

**Anexo 12: Librería Scikit-Learn - Función ReLU – 70% entrenamiento – 30% prueba.**

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPRegressor
from sklearn.metrics import mean_squared_error, r2_score

# Cargamos el archivo CSV
url = '/content/DATA_PGA_050.csv'  # Ruta del archivo CSV
df = pd.read_csv(url)

# Seleccionamos solo las variables relevantes
df_selected = df[['EpiD (km)', 'Earthquake Magnitude', 'Mechanism Based on
Rake Angle', 'Vs30 (m/s) selected for analysis','HypD (km)', 'PGA (g)']]

# Dividimos los datos en características (X) y etiquetas (y)
X = df_selected.drop('PGA (g)', axis=1).values
y = df_selected['PGA (g)'].values
```

```
# Dividimos los datos en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

# Escalamos los datos
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Definimos el modelo de red neuronal con Scikit-Learn
model = MLPRegressor(hidden_layer_sizes=(5, 64, 32, 16),
activation='relu', solver='adam', alpha=0.0001, max_iter=100,
random_state=42)

# Entrenamos el modelo
model.fit(X_train, y_train)

# Realizamos predicciones en el conjunto de prueba
y_pred = model.predict(X_test)

# Calculamos las métricas de evaluación
mse = mean_squared_error(y_test, y_pred)
print('Mean Squared Error on Test Set:', mse)

r2 = r2_score(y_test, y_pred)
print('Coeficiente de determinación R²:', r2)

# Guardamos el modelo (esto es opcional en Scikit-Learn)
import joblib
joblib.dump(model, 'modelo.pkl')
```

**Anexo 13: Librería Scikit-Learn - Función ReLU – 90% entrenamiento – 10% prueba.**

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPRegressor
from sklearn.metrics import mean_squared_error, r2_score

# Cargamos el archivo CSV
url = '/content/DATA_PGA_050.csv'  # Ruta del archivo CSV
df = pd.read_csv(url)

# Seleccionamos solo las variables relevantes
```

```
df_selected = df[['EpiD (km)', 'Earthquake Magnitude', 'Mechanism Based on
Rake Angle', 'Vs30 (m/s) selected for analysis','HypD (km)', 'PGA (g)']]

# Dividimos los datos en características (X) y etiquetas (y)
X = df_selected.drop('PGA (g)', axis=1).values
y = df_selected['PGA (g)'].values

# Dividimos los datos en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1,
random_state=42)

# Escalamos los datos
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Definimos el modelo de red neuronal con Scikit-Learn
model = MLPRegressor(hidden_layer_sizes=(5, 64, 32, 16),
activation='relu', solver='adam', alpha=0.0001, max_iter=100,
random_state=42)

# Entrenamos el modelo
model.fit(X_train, y_train)

# Realizamos predicciones en el conjunto de prueba
y_pred = model.predict(X_test)

# Calculamos las métricas de evaluación
mse = mean_squared_error(y_test, y_pred)
print('Mean Squared Error on Test Set:', mse)

r2 = r2_score(y_test, y_pred)
print('Coeficiente de determinación R²:', r2)

# Guardamos el modelo (esto es opcional en Scikit-Learn)
import joblib
joblib.dump(model, 'modelo.pkl')
```

**Anexo 14: Librería Scikit-Learn - Función Tangente Hiperbólica – 80% entrenamiento – 10% prueba.**

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
```

```
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPRegressor
from sklearn.metrics import mean_squared_error, r2_score

# Cargamos el archivo CSV
url = '/content/DATA_PGA_050.csv'  # Ruta del archivo CSV
df = pd.read_csv(url)

# Seleccionamos solo las variables relevantes
df_selected = df[['EpiD (km)', 'Earthquake Magnitude', 'Mechanism Based on
Rake Angle', 'Vs30 (m/s) selected for analysis','HypD (km)', 'PGA (g)']]

# Dividimos los datos en características (X) y etiquetas (y)
X = df_selected.drop('PGA (g)', axis=1).values
y = df_selected['PGA (g)'].values

# Dividimos los datos en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Escalamos los datos
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Definimos el modelo de red neuronal con Scikit-Learn y la función de
activación Tangente Hiperbólica
model = MLPRegressor(hidden_layer_sizes=(5, 64, 32, 16),
activation='tanh', solver='adam', alpha=0.0001, max_iter=100,
random_state=42)

# Entrenamos el modelo
model.fit(X_train, y_train)

# Realizamos predicciones en el conjunto de prueba
y_pred = model.predict(X_test)

# Calculamos las métricas de evaluación
mse = mean_squared_error(y_test, y_pred)
print('Mean Squared Error on Test Set:', mse)

r2 = r2_score(y_test, y_pred)
print('Coeficiente de determinación R²:', r2)

# Guardamos el modelo (esto es opcional en Scikit-Learn)
```

```
import joblib
joblib.dump(model, 'modelo.pkl')
```

**Anexo 15: Librería Scikit-Learn - Función Sigmoide – 80% entrenamiento – 10% prueba.**

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPRegressor
from sklearn.metrics import mean_squared_error, r2_score

# Cargamos el archivo CSV
url = '/content/DATA_PGA_050.csv'  # Ruta del archivo CSV
df = pd.read_csv(url)

# Seleccionamos solo las variables relevantes
df_selected = df[['EpiD (km)', 'Earthquake Magnitude', 'Mechanism Based on
Rake Angle', 'Vs30 (m/s) selected for analysis','HypD (km)', 'PGA (g)']]

# Dividimos los datos en características (X) y etiquetas (y)
X = df_selected.drop('PGA (g)', axis=1).values
y = df_selected['PGA (g)'].values

# Dividimos los datos en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Escalamos los datos
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Definimos el modelo de red neuronal con Scikit-Learn y la función de
activación Sigmoide
model = MLPRegressor(hidden_layer_sizes=(5, 64, 32, 16),
activation='logistic', solver='adam', alpha=0.0001, max_iter=100,
random_state=42)

# Entrenamos el modelo
model.fit(X_train, y_train)

# Realizamos predicciones en el conjunto de prueba
```

```
y_pred = model.predict(X_test)

# Calculamos las métricas de evaluación
mse = mean_squared_error(y_test, y_pred)
print('Mean Squared Error on Test Set:', mse)

r2 = r2_score(y_test, y_pred)
print('Coeficiente de determinación R²:', r2)

# Guardamos el modelo (esto es opcional en Scikit-Learn)
import joblib
joblib.dump(model, 'modelo.pkl')
```