**Presentation on Earthquake Prediction Model Using Python**

**Project Overview:**

This presentation centers on the development of an earthquake prediction model through data-driven techniques and machine learning algorithms. The primary aim is to harness seismic data for the purpose of forecasting earthquakes, ultimately enhancing early warning systems and risk assessment strategies.

**Data Acquisition:**

Accessing Seismic Data

- Collaboration with Earthquake Monitoring Institutions

- Defining Data Parameters: Location, Magnitude, Depth, Geological Features

- Data Retrieval Methods: Python Scripts and Data Retrieval Tools

**Data Preprocessing**:

Preparing Seismic Data for Analysis

- Cleaning the Dataset: Handling Anomalies and Missing Values

- Feature Engineering: Extracting Relevant Features

- Data Transformation: Normalization and Standardization

**Model Development:**

Selecting and Training the Prediction Model

- Choosing Machine Learning or Deep Learning Models

- Model Training and Hyperparameter Optimization

- Model Evaluation and Validation Metrics

**Real-time Monitoring and Early Warning:**

Creating a Continuous Monitoring System

- Real-time Analysis of Seismic Data

- Providing Early Warnings and Risk Assessment

**Feedback and Collaboration:**

Enhancing Preparedness and Response

- Establishing a Feedback Mechanism

- Collaboration with Authorities and Scientific Community

**Continuous Improvement:**

Adapting to New Data and Scientific Advances

- Ongoing Model Refinement

- Incorporating the Latest Findings in Earthquake Science

**Stakeholder Engagement:**

Sharing Insights and Promoting Preparedness

- Collaboration with Seismologists and Emergency Agencies

- Dissemination of Research and Public Awareness

**Implementation of Safety Measures:**

Putting Knowledge into Action

- Translating Forecasts into Safety Measures

- Protecting Lives and Infrastructure

**Reporting and Research:**

Sharing Progress and Insights

- Research Reports and Publications

- Public Awareness Materials for Earthquake Prediction

It's important to remember that while data-driven approaches can improve preparedness and understanding, earthquake prediction remains a formidable scientific challenge, and precise prediction remains an ongoing endeavor.

**Code:**

**import** numpy **as** np

**import** tensorflow **as** tf

**import** pandas **as** pd

**import** time

**from** datetime **import** datetime

In [2]:

*#Extract data from CSV*

df1**=**pd**.**read\_csv("database.csv")

In [3]:

epoch **=** datetime(1970, 1, 1)

**def** mapdateTotime(x):

**try**:

dt **=** datetime**.**strptime(x, "%m/%d/%Y")

**except** ValueError:

dt **=** datetime**.**strptime(x, "%Y-%m-%dT%H:%M:%S.%fZ")

diff **=** dt **-** epoch

**return** diff**.**total\_seconds()

df1**.**Date **=** df1**.**Date**.**apply(mapdateTotime)

In [4]:

col1 **=** df1[['Date','Latitude','Longitude','Depth']]

col2 **=** df1['Magnitude']

*#Convert to Numpy array*

InputX1 **=** col1**.**as\_matrix()

InputY1 **=** col2**.**as\_matrix()

print(InputX1)

[[ -1.57680000e+08 1.92460000e+01 1.45616000e+02 1.31600000e+02]

[ -1.57507200e+08 1.86300000e+00 1.27352000e+02 8.00000000e+01]

[ -1.57420800e+08 -2.05790000e+01 -1.73972000e+02 2.00000000e+01]

...,

[ 1.48288320e+09 3.69179000e+01 1.40426200e+02 1.00000000e+01]

[ 1.48296960e+09 -9.02830000e+00 1.18663900e+02 7.90000000e+01]

[ 1.48305600e+09 3.73973000e+01 1.41410300e+02 1.19400000e+01]]

In [5]:

*#Min-max Normalization*

X1\_min **=** np**.**amin(InputX1,0)

X1\_max **=** np**.**amax(InputX1,0)

print("Mininum values:",X1\_min)

print("Maximum values:",X1\_max)

Y1\_min **=** np**.**amin(InputY1)

Y1\_max **=** np**.**amax(InputY1)

InputX1\_norm **=** (InputX1**-**X1\_min)**/**(X1\_max**-**X1\_min)

InputY1\_norm **=** InputY1 *#No normalization in output*

*#Reshape*

Xfeatures **=** 3 *#Number of input features*

Yfeatures **=** 1 *#Number of input features*

samples **=** 23000 *# Number of samples*

InputX1\_reshape **=** np**.**resize(InputX1\_norm,(samples,Xfeatures))

InputY1\_reshape **=** np**.**resize(InputY1\_norm,(samples,Yfeatures))

Mininum values: [ -1.57680000e+08 -7.70800000e+01 -1.79997000e+02 -1.10000000e+00]

Maximum values: [ 1.48305600e+09 8.60050000e+01 1.79998000e+02 7.00000000e+02]

In [6]:

*#Training data*

batch\_size **=** 2000

InputX1train **=** InputX1\_reshape[0:batch\_size,:]

InputY1train **=** InputY1\_reshape[0:batch\_size,:]

*#Validation data*

v\_size **=** 2500

InputX1v **=** InputX1\_reshape[batch\_size:batch\_size**+**v\_size,:]

InputY1v **=** InputY1\_reshape[batch\_size:batch\_size**+**v\_size,:]

In [7]:

learning\_rate **=** 0.001

training\_iterations **=** 1000

display\_iterations **=** 200

In [8]:

*#Input*

X **=** tf**.**placeholder(tf**.**float32,shape**=**(**None**,Xfeatures))

*#Output*

Y **=** tf**.**placeholder(tf**.**float32)

In [9]:

*#Neurons*

L1 **=** 3

L2 **=** 3

L3 **=** 3

*#Layer1 weights*

W\_fc1 **=** tf**.**Variable(tf**.**random\_uniform([Xfeatures,L1]))

b\_fc1 **=** tf**.**Variable(tf**.**constant(0.1,shape**=**[L1]))

*#Layer2 weights*

W\_fc2 **=** tf**.**Variable(tf**.**random\_uniform([L1,L2]))

b\_fc2 **=** tf**.**Variable(tf**.**constant(0.1,shape**=**[L2]))

*#Layer3 weights*

W\_fc3 **=** tf**.**Variable(tf**.**random\_uniform([L2,L3]))

b\_fc3 **=** tf**.**Variable(tf**.**constant(0.1,shape**=**[L3]))

*#Output layer weights*

W\_fO**=** tf**.**Variable(tf**.**random\_uniform([L3,Yfeatures]))

b\_fO **=** tf**.**Variable(tf**.**constant(0.1,shape**=**[Yfeatures]))

In [10]:

*#Layer 1*

matmul\_fc1**=**tf**.**matmul(X, W\_fc1) **+** b\_fc1

h\_fc1 **=** tf**.**nn**.**relu(matmul\_fc1) *#ReLU activation*

*#Layer 2*

matmul\_fc2**=**tf**.**matmul(h\_fc1, W\_fc2) **+** b\_fc2

h\_fc2 **=** tf**.**nn**.**relu(matmul\_fc2) *#ReLU activation*

*#Layer 3*

matmul\_fc3**=**tf**.**matmul(h\_fc2, W\_fc3) **+** b\_fc3

h\_fc3 **=** tf**.**nn**.**relu(matmul\_fc3) *#ReLU activation*

*#Output layer*

matmul\_fc4**=**tf**.**matmul(h\_fc3, W\_fO) **+** b\_fO

output\_layer **=** matmul\_fc4 *#linear activation*

In [11]:

*#Loss function*

mean\_square **=** tf**.**reduce\_mean(tf**.**square(Y**-**output\_layer))

train\_step **=** tf**.**train**.**AdamOptimizer(learning\_rate)**.**minimize(mean\_square)

*#Operation to save variables*

saver **=** tf**.**train**.**Saver()

In [12]:

*#Initialization and session*

init **=** tf**.**global\_variables\_initializer()

**with** tf**.**Session() **as** sess:

sess**.**run(init)

print("Training loss:",sess**.**run([mean\_square],feed\_dict**=**{X:InputX1train,Y:InputY1train}))

**for** i **in** range(training\_iterations):

sess**.**run([train\_step],feed\_dict**=**{X:InputX1train,Y:InputY1train})

**if** i**%display\_iterations** ==0:

print("Training loss is:",sess**.**run([mean\_square],feed\_dict**=**{X:InputX1train,Y:InputY1train}),"at itertion:",i)

print("Validation loss is:",sess**.**run([mean\_square],feed\_dict**=**{X:InputX1v,Y:InputY1v}),"at itertion:",i)

*# Save the variables to disk.*

save\_path **=** saver**.**save(sess, "/tmp/earthquake\_model.ckpt")

print("Model saved in file: %s" **%** save\_path)

print("Final training loss:",sess**.**run([mean\_square],feed\_dict**=**{X:InputX1train,Y:InputY1train}))

print("Final validation loss:",sess**.**run([mean\_square],feed\_dict**=**{X:InputX1v,Y:InputY1v}))

Training loss: [7.8162642]

Training loss is: [7.6603436] at itertion: 0

Validation loss is: [6.4701533] at itertion: 0

Training loss is: [1.7573606] at itertion: 200

Validation loss is: [1.680984] at itertion: 200

Training loss is: [1.09281] at itertion: 400

Validation loss is: [1.0900164] at itertion: 400

Training loss is: [0.69149578] at itertion: 600

Validation loss is: [0.70860344] at itertion: 600

Training loss is: [0.45098665] at itertion: 800

Validation loss is: [0.46836597] at itertion: 800

Model saved in file: /tmp/earthquake\_model.ckpt

Final training loss: [0.31291121]

Final validation loss: [0.32535991]