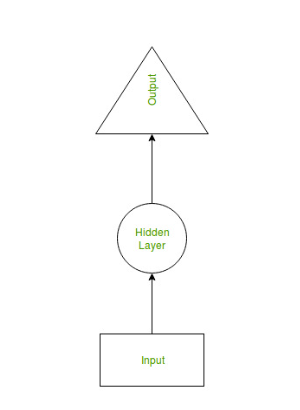
**ISL WEEK -10**

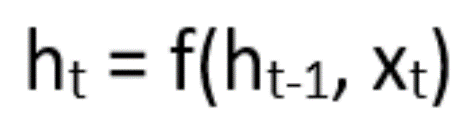
**Aim: Write a program to build basic Recurrent neural network model (eg. Covid-19 Forecasting using an RNN** [**https://www.kaggle.com/frlemarchand/covid-19-forecasting-with-an-rnn/#data**](https://www.kaggle.com/frlemarchand/covid-19-forecasting-with-an-rnn/#data)**)**

**Description:**

**Recurrent Neural Network (RNN) are a type of Neural Network where the output from previous step is fed as input to the current step. In traditional neural networks, all the inputs and outputs are independent of each other, but in cases like when it is required to predict the next word of a sentence, the previous words are required and hence there is a need to remember the previous words. Thus, RNN came into existence, which solved this issue with the help of a Hidden Layer. The main and most important feature of RNN is Hidden state, which remembers some information about a sequence.**

****

**RNN have a “memory” which remembers all information about what has been calculated. It uses the same parameters for each input as it performs the same task on all the inputs or hidden layers to produce the output. This reduces the complexity of parameters, unlike other neural networks.**

****

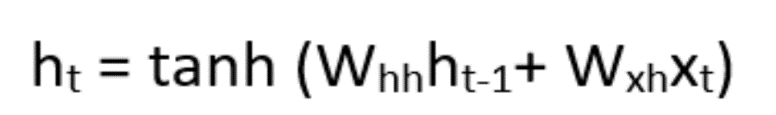
**where:**

**ht -> current state**

**ht-1 -> previous state**

**xt -> input state**

**Activation Function:**

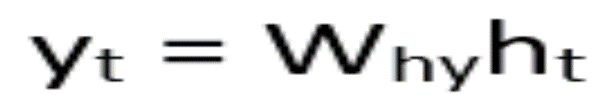
****

**where:**

**whh -> weight at recurrent neuron**

**wxh -> weight at input neuron**

**Output:**

****

**Yt -> output**

**Why -> weight at output layer**

**Code:**

import numpy as np

import pandas as pd

import geopandas as gpd

from shapely.geometry import Point

import os

import tensorflow as tf

from tqdm import tqdm

from sklearn.utils import shuffle

from sklearn.metrics import mean\_squared\_log\_error

from datetime import datetime

from datetime import timedelta

from tensorflow.keras import layers

from tensorflow.keras import Input

from tensorflow.keras.models import Model

from tensorflow.keras.callbacks import ModelCheckpoint, ReduceLROnPlateau, EarlyStopping

import matplotlib.pyplot as plt

import seaborn as sns

sns.set()

train\_df = gpd.read\_file("/kaggle/input/covid19-global-forecasting-week-4/train.csv")

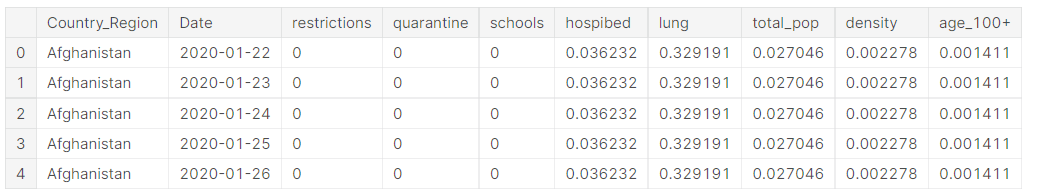
train\_df["ConfirmedCases"] = train\_df["ConfirmedCases"].astype("float")

train\_df["Fatalities"] = train\_df["Fatalities"].astype("float")

*#The country\_region*

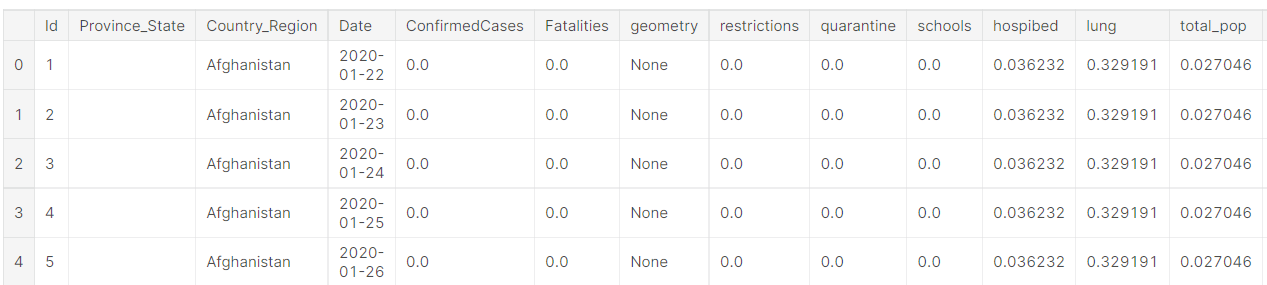
train\_df["Country\_Region"] = [ row.Country\_Region.replace("'","").strip(" ") if row.Province\_State=="" else str(row.Country\_Region+"\_"+row.Province\_State).replace("'","").strip(" ") for idx,row **in** train\_df.iterrows()]

extra\_data\_df.head()



train\_df = train\_df.merge(extra\_data\_df, how="left", on=['Country\_Region','Date']).drop\_duplicates()

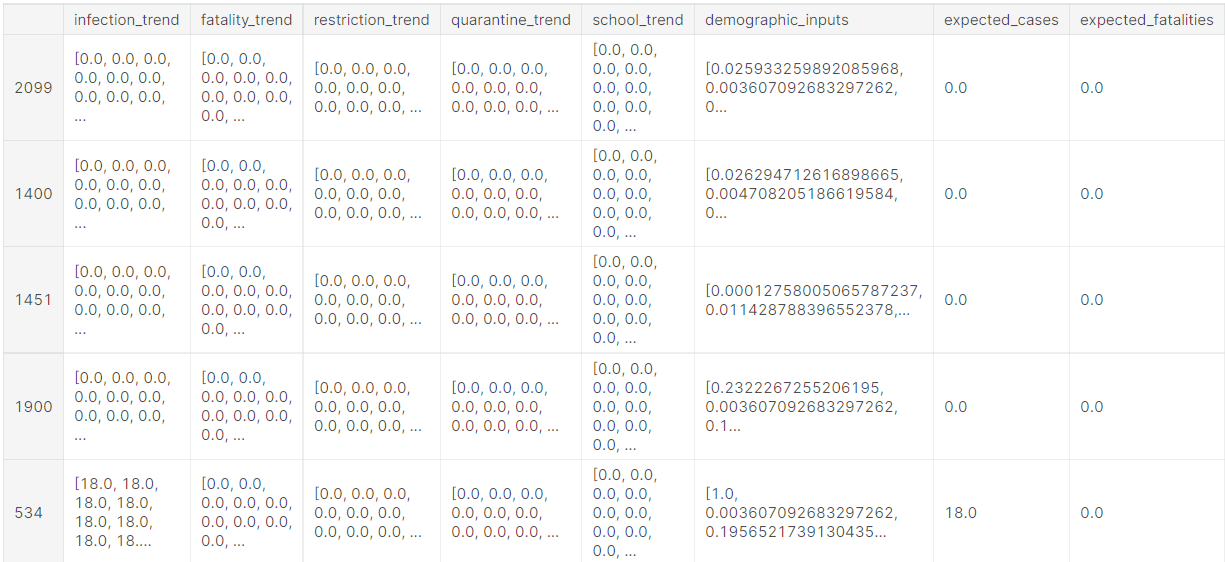
train\_df.head()



trend\_df["temporal\_inputs"] = [np.asarray([trends["infection\_trend"],trends["fatality\_trend"],trends["restriction\_trend"],trends["quarantine\_trend"],trends["school\_trend"]]) for idx,trends **in** trend\_df.iterrows()]

trend\_df = shuffle(trend\_df)

trend\_df.head()



*#temporal input branch*

temporal\_input\_layer = Input(shape=(sequence\_length,5))

main\_rnn\_layer = layers.LSTM(64, return\_sequences=True, recurrent\_dropout=0.2)(temporal\_input\_layer)

*#demographic input branch*

demographic\_input\_layer = Input(shape=(5))

demographic\_dense = layers.Dense(16)(demographic\_input\_layer)

demographic\_dropout = layers.Dropout(0.2)(demographic\_dense)

*#cases output branch*

rnn\_c = layers.LSTM(32)(main\_rnn\_layer)

merge\_c = layers.Concatenate(axis=-1)([rnn\_c,demographic\_dropout])

dense\_c = layers.Dense(128)(merge\_c)

dropout\_c = layers.Dropout(0.3)(dense\_c)

cases = layers.Dense(1, activation=layers.LeakyReLU(alpha=0.1),name="cases")(dropout\_c)

*#fatality output branch*

rnn\_f = layers.LSTM(32)(main\_rnn\_layer)

merge\_f = layers.Concatenate(axis=-1)([rnn\_f,demographic\_dropout])

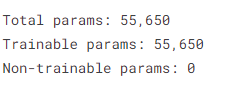
dense\_f = layers.Dense(128)(merge\_f)

dropout\_f = layers.Dropout(0.3)(dense\_f)

fatalities = layers.Dense(1, activation=layers.LeakyReLU(alpha=0.1), name="fatalities")(dropout\_f)

model = Model([temporal\_input\_layer,demographic\_input\_layer], [cases,fatalities])

model.summary()



callbacks = [ReduceLROnPlateau(monitor='val\_loss', patience=4, verbose=1, factor=0.6),

EarlyStopping(monitor='val\_loss', patience=20),

ModelCheckpoint(filepath='best\_model.h5', monitor='val\_loss', save\_best\_only=True)]

model.compile(loss=[tf.keras.losses.MeanSquaredLogarithmicError(),tf.keras.losses.MeanSquaredLogarithmicError()], optimizer="adam")

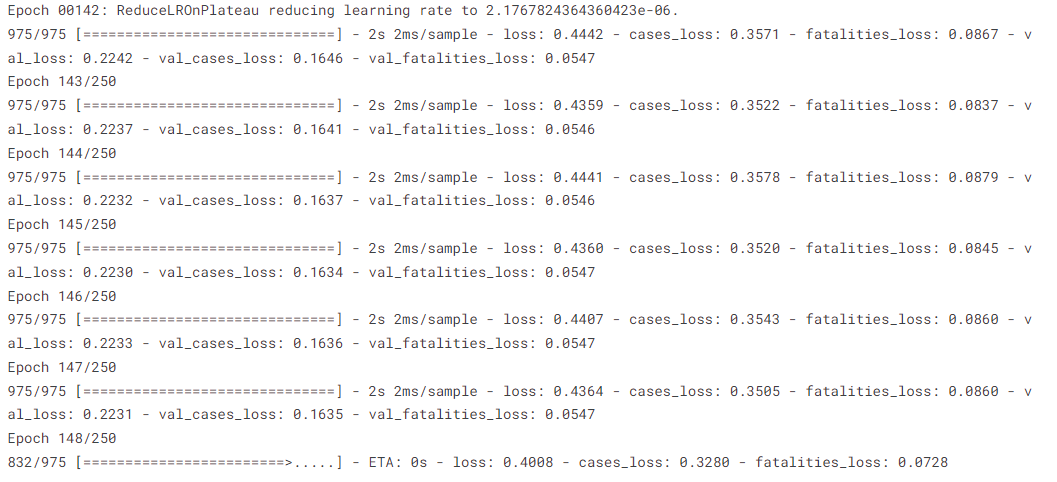
history = model.fit([X\_temporal\_train,X\_demographic\_train], [Y\_cases\_train, Y\_fatalities\_train],

epochs = 250,

batch\_size = 16,

validation\_data=([X\_temporal\_test,X\_demographic\_test], [Y\_cases\_test, Y\_fatalities\_test]),

callbacks=callbacks)



plt.plot(history.history['loss'])

plt.plot(history.history['val\_loss'])

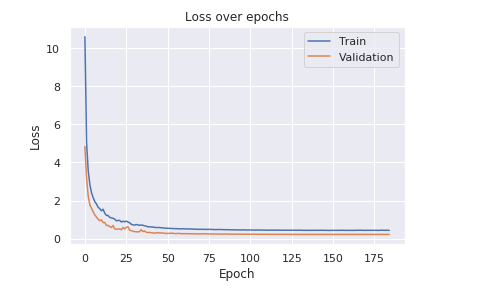
plt.title('Loss over epochs')

plt.ylabel('Loss')

plt.xlabel('Epoch')

plt.legend(['Train', 'Validation'], loc='best')

plt.show()



plt.plot(history.history['fatalities\_loss'])

plt.plot(history.history['val\_fatalities\_loss'])

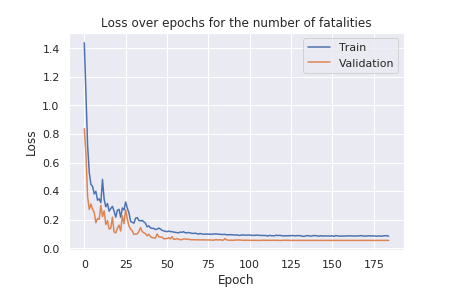
plt.title('Loss over epochs for the number of fatalities')

plt.ylabel('Loss')

plt.xlabel('Epoch')

plt.legend(['Train', 'Validation'], loc='best')

plt.show()



model.load\_weights("best\_model.h5")

predictions = model.predict([X\_temporal\_test,X\_demographic\_test])

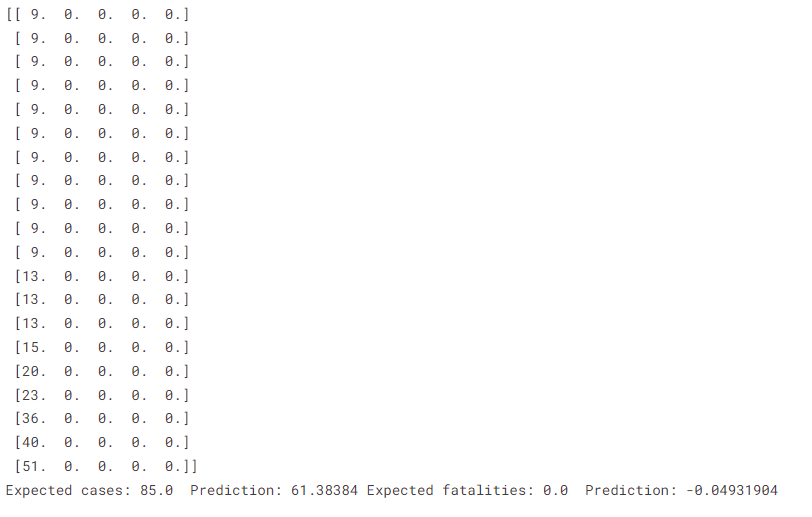
display\_limit = 30

for inputs, pred\_cases, exp\_cases, pred\_fatalities, exp\_fatalities **in** zip(X\_temporal\_test,predictions[0][:display\_limit], Y\_cases\_test[:display\_limit], predictions[1][:display\_limit], Y\_fatalities\_test[:display\_limit]):

print("================================================")

print(inputs)

print("Expected cases:", exp\_cases, " Prediction:", pred\_cases[0], "Expected fatalities:", exp\_fatalities, " Prediction:", pred\_fatalities[0] )



def get\_RMSLE\_for\_all\_regions(groundtruth\_df):

RMSLE\_cases\_list = []

RMSLE\_fatalities\_list = []

for region **in** groundtruth\_df.Country\_Region.unique():

RMSLE\_cases, RMSLE\_fatalities = get\_RMSLE\_per\_region(region, groundtruth\_df, False)

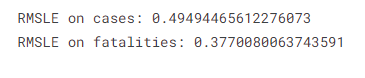
RMSLE\_cases\_list.append(RMSLE\_cases)

RMSLE\_fatalities\_list.append(RMSLE\_fatalities)

print("RMSLE on cases:",np.mean(RMSLE\_cases\_list))

print("RMSLE on fatalities:",np.mean(RMSLE\_fatalities\_list))

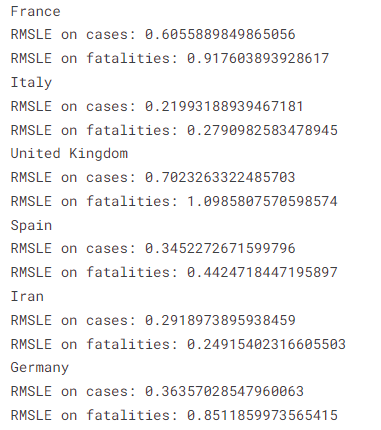
get\_RMSLE\_for\_all\_regions(groundtruth\_df)



badly\_affected\_countries = ["France","Italy","United Kingdom","Spain","Iran","Germany"]

for country **in** badly\_affected\_countries:

get\_RMSLE\_per\_region(country, groundtruth\_df, display\_only=True)



def display\_comparison(region,groundtruth\_df):

groundtruth = groundtruth\_df.query("Country\_Region=='"+region+"' and Date>='2020-04-01' and Date<='2020-04-15'")

prediction = copy\_df.query("Country\_Region=='"+region+"' and Date>='2020-04-01' and Date<='2020-04-15'")

plt.plot(groundtruth.ConfirmedCases.values)

plt.plot(prediction.ConfirmedCases.values)

plt.title("Comparison between the actual data and our predictions for the number of cases")

plt.ylabel('Number of cases')

plt.xlabel('Date')

plt.xticks(range(len(prediction.Date.values)),prediction.Date.values,rotation='vertical')

plt.legend(['Groundtruth', 'Prediction'], loc='best')

plt.show()

plt.plot(groundtruth.Fatalities.values)

plt.plot(prediction.Fatalities.values)

plt.title("Comparison between the actual data and our predictions for the number of fatalities")

plt.ylabel('Number of fatalities')

plt.xlabel('Date')

plt.xticks(range(len(prediction.Date.values)),prediction.Date.values,rotation='vertical')

plt.legend(['Groundtruth', 'Prediction'], loc='best')

plt.show()

display\_comparison("Canada\_Newfoundland and Labrador", groundtruth\_df)

