`Created in May 2024`

`File: CGAN_under_deployment.ipynb`


`@author: Pavan Mohan Neelamraju`

`Affiliation: Indian Institute of Technology - Madras,`

**Email**: npavanmohan3@gmail.com


**Description**: This program shows the proper set up of a trained generator of C-GAN model for prediction of Response Spectra.

**Website**: pavanmohann.github.io

---

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from google.colab import drive

drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly
remount, call drive.mount("/content/drive", force_remount=True).
```

```
# # 'eqm','ftype','hyp','dist','log_dist','log_vs30','dir']
# eqm = float(input('Earthquake Magnitude $M_w$ '))
# f = float(input('Fault Type $F$ '))
# hd = float(input('Hypocentre Depth $H_d$ '))
# dist = float(input('RJB Distance $R_{JB}$ '))
# logdist = np.log(dist)
# vs30 = float(input('Vs30 $V_{s30}$ '))
# logvs30 = np.log(vs30)
# dir = float(input('Direction $dir$ '))
# l1 = [eqm, f, hd, dist, logdist, vs30, logvs30, dir]

df = pd.read_csv('/content/drive/MyDrive/SDA
Codes/Complete_GAN/final_data.csv')

# Assuming df is your DataFrame and columns is a list of your specific
columns
columns = ['pga',
'T0.010S',
'T0.020S',
'T0.030S',
'T0.040S',
'T0.050S',
```

```
'T0.060S',
'T0.070S',
'T0.080S',
'T0.090S',
'T0.150S',
'T0.200S',
'T0.300S',
'T0.500S',
'T0.600S',
'T0.700S',
'T0.800S',
'T0.900S',
'T1.000S',
'T1.200S',
'T1.500S',
'T2.000S',
'T3.000S',
'T4.000S',]  # replace with your actual column names
df = df[df[columns].max(axis=1) >= 0.001]
df.shape
```

```
(18943, 33)
```

```
num = 1# int(input('Enter the value of a record to analyse the
response spectra [0 to '+str(df.shape)+']: '))



# Assuming you have your features in a variable named X and your
labels in a variable named Y

X = df[['eqm','ftype','hyp','dist','log_dist','log_vs30','dir']]
Y = df[['pga',
'T0.010S',
'T0.020S',
'T0.030S',
'T0.040S',
'T0.050S',
'T0.060S',
'T0.070S',
'T0.080S',
'T0.090S',
'T0.150S',
'T0.200S',
'T0.300S',
'T0.500S',
'T0.600S',
'T0.700S',
'T0.800S',
'T0.900S',
```

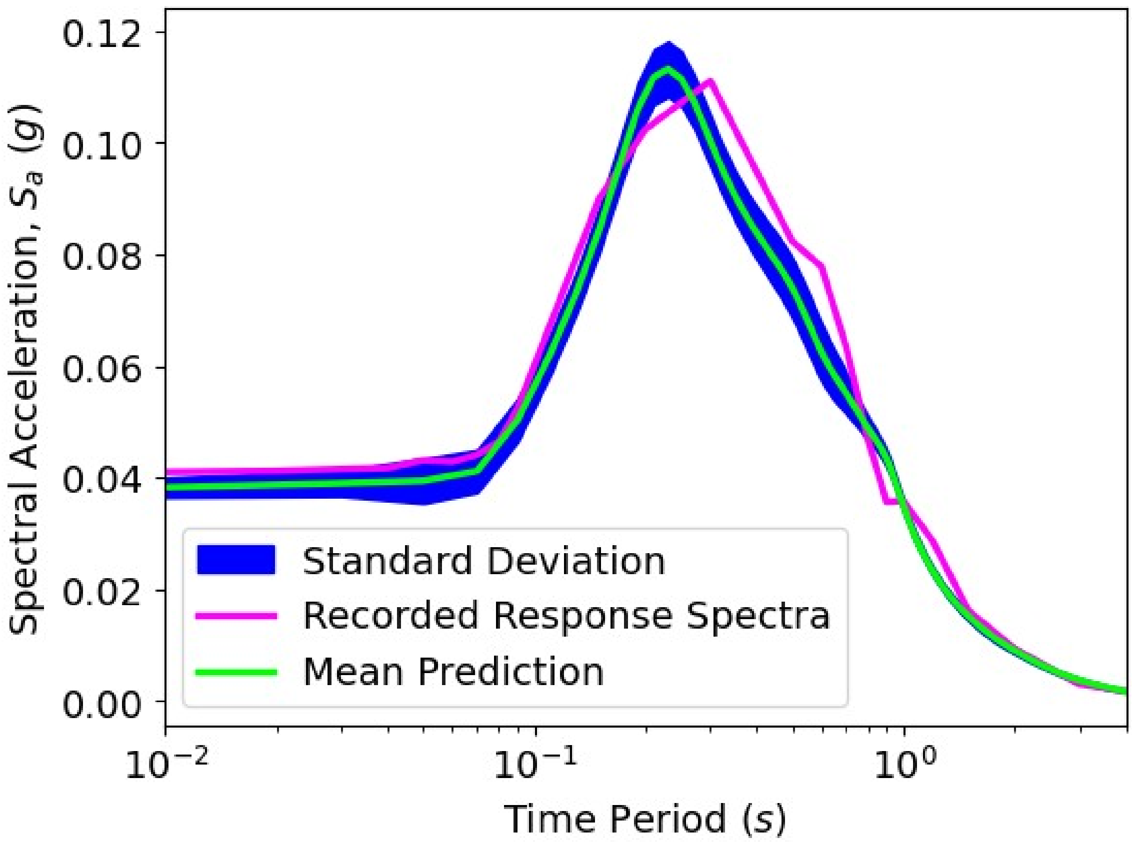
```
'T1.000S',
'T1.200S',
'T1.500S',
'T2.000S',
'T3.000S',
'T4.000S',
]]

X = X.dropna()
Y = Y.dropna()

testing_sample_x = X.iloc[num,:]
testing_sample_y = Y.iloc[num,:]

from sklearn.preprocessing import MinMaxScaler

X_scaler = MinMaxScaler()

X_scaled = X_scaler.fit_transform(X)

Y_scaler = MinMaxScaler()

Y_scaled = Y_scaler.fit_transform(Y)

from keras.layers import Layer
import keras.backend as K
from tensorflow import keras
from tensorflow.keras import Sequential
from tensorflow.keras import models, layers
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Activation, Dense,
BatchNormalization, Dropout
from tensorflow.keras import optimizers
import warnings

# Suppress all warnings
warnings.filterwarnings("ignore")


class StochasticInputLayer(Layer):
    def __init__(self, output_dim, **kwargs):
        self.output_dim = output_dim
        super(StochasticInputLayer, self).__init__(**kwargs)

    def build(self, input_shape):
        super(StochasticInputLayer, self).build(input_shape)

    def call(self, x):

        noise = K.random_normal(shape=(self.output_dim,), mean=0.,
stddev=0.01)
```

```
        return x + x * noise

    def compute_output_shape(self, input_shape):
        return (input_shape[0], self.output_dim)

from keras.models import load_model

custom_objects = {'StochasticInputLayer': StochasticInputLayer}

# Load the model
model = load_model('/content/drive/MyDrive/SDA
Codes/Complete_GAN/cgan_gen_1.hdf5', custom_objects=custom_objects)

time_periods = [0.0099, 0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07,
0.08,0.09, 0.15, 0.2, 0.3, 0.5, 0.6, 0.7, 0.8, 0.9, 1, 1.2, 1.5, 2, 3,
4]

# Assuming x_vals is your unscaled input
x_vals = [testing_sample_x]
## 'eqm','ftype','hyp','dist','log_dist','log_vs30','dir'
# Use the fitted scaler to transform the inputs
x_vals_scaled = X_scaler.transform(x_vals)

# Use the model to predict the output
y_pred_scaled = model.predict(x_vals_scaled)

# Use the fitted scaler to inverse transform the predicted output
y_pred = Y_scaler.inverse_transform(y_pred_scaled)

print("Predicted Immediate output:", y_pred)
# plt.plot(y_pred[0])
# plt.ylim([0,0.5])

asd = testing_sample_x.to_list()
asdf = []
for i in asd:
  if asd.index(i) != 5:
    asdf.append(i)
  else:
    asdf.append(np.exp(i))
# 'eqm','ftype','hyp','dist','log_dist','log_vs30','dir'
print(asdf)
```

```
1/1 [==============================] - 0s 77ms/step
Predicted Immediate output: [[0.04166881 0.04182466 0.04331673
0.04522937 0.05158574 0.05218262
  0.05216977 0.05273917 0.05601563 0.06080849 0.08948894 0.11389616
  0.10221852 0.0847429  0.07411798 0.06352392 0.05336797 0.04647194
  0.0369678  0.02466503 0.01657144 0.01012807 0.00494487 0.00225194]]
[5.8, 0.0, 10.0, 71.28, 4.266615783162554, 219.31, 0.0]
```

```
model.summary()

Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 stochastic_input_layer (St  (None, 7)                 0
 ochasticInputLayer)

 dense (Dense)               (None, 182)               1456

 dense_1 (Dense)             (None, 182)               33306

 dense_2 (Dense)             (None, 182)               33306

 dense_3 (Dense)             (None, 182)               33306

 dense_4 (Dense)             (None, 24)                4392

=================================================================
Total params: 105766 (413.15 KB)
Trainable params: 105766 (413.15 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________

import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import make_interp_spline
fontsize = 14

def make_predictions(model, X_scaler, Y_scaler, testing_sample_x,
num_predictions=75):
    # Scale the input values
    x_vals_scaled = X_scaler.transform([testing_sample_x])

    # Make predictions
    y_pred_scaled = [model.predict(x_vals_scaled, verbose=0) for _ in
range(num_predictions)]

    # Inverse transform the predicted output
    predictions = [Y_scaler.inverse_transform(y)[0] for y in
y_pred_scaled]

    return np.array(predictions)

%matplotlib inline

def plot_predictions_and_true_values(predictions, true_values,
time_periods):
    # Calculate the mean and standard deviation of the predictions
    mean = np.mean(predictions, axis=0)
```

```
    std_dev = np.std(predictions, axis=0)

    # Define the x values for the plot
    x = np.array(time_periods)

    # Create a spline for the mean
    mean_spline = make_interp_spline(x, mean)
    xnew = np.linspace(x.min(), x.max(), 200)
    mean_smooth = mean_spline(xnew)

    # Create a spline for the standard deviation
    std_dev_spline = make_interp_spline(x, std_dev)
    std_dev_smooth = std_dev_spline(xnew)

    # Plot the mean and standard deviation of the predictions
    # plt.figure(figsize=(10, 6), dpi=300)  # Adjust plot size and set
DPI
    plt.fill_between(xnew, mean_smooth - std_dev_smooth, mean_smooth +
std_dev_smooth, color='b', alpha=1,
                     label='Standard Deviation')

    # Plot true response spectra
    plt.plot(time_periods, true_values, color='magenta',
linewidth=2.5, label='Recorded Response Spectra')
    plt.plot(xnew, mean_smooth, label='Mean Prediction', color='lime',
linewidth=2.5)

    plt.xscale('log')
    plt.xlabel('Time Period ($s$)',fontsize=fontsize)
    plt.xticks(fontsize=fontsize)
    plt.yticks(fontsize=fontsize)
    plt.xlim([0.0099,4])
    plt.ylabel('Spectral Acceleration, $S_a$ ($g$)',fontsize=fontsize)
    plt.legend(fontsize=fontsize)
    plt.show()

# Make predictions
predictions = make_predictions(model, X_scaler, Y_scaler,
testing_sample_x)

# Plot predictions and true values
plot_predictions_and_true_values(predictions, testing_sample_y,
time_periods)
```




```
asdf
# 'eqm','ftype','hyp','dist','log_dist','log_vs30','dir'

[5.8, 0.0, 10.0, 71.28, 4.266615783162554, 219.31, 0.0]

# import numpy as np
# import matplotlib.pyplot as plt
# from scipy.interpolate import make_interp_spline

# fontsize = 14

# def make_predictions(model, X_scaler, Y_scaler, testing_sample_x,
num_predictions=75):
#     # Scale the input values
#     x_vals_scaled = X_scaler.transform([testing_sample_x])

#     # Make predictions
#     y_pred_scaled = [model.predict(x_vals_scaled, verbose=0) for _
in range(num_predictions)]

#     # Inverse transform the predicted output
#     predictions = [Y_scaler.inverse_transform(y)[0] for y in
y_pred_scaled]
```

```
#     return np.array(predictions)

# # Example usage
# l1 = [eqm, f, hd, dist, logdist, logvs30, dir]
# predictions = make_predictions(model, X_scaler, Y_scaler, l1)
```