```
# # Grid Search com Early Stopping
# ## GRU


import numpy as np
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '0' # 0 (default), 1 (WARNING), 2 (ERROR), 3 (FATAL)
import tensorflow as tf
import pandas as pd

from sklearn import metrics
from sklearn.model_selection import GridSearchCV
from tensorflow import keras
from keras.models import Model
from keras.utils import plot_model
from keras.callbacks import EarlyStopping
# from keras.models import Modelconda
from keras.layers import Input, Dense, TimeDistributed, GRU
import csv


from sklearn.metrics import mean_squared_error

from helperFunctions import split_dataset_by_window, split_dataset, scaler


# numero de produtores
n_prod=4
# numero de injetores
n_inj=8
# define input to injetor wells--> bhp or rate
type_control_inj='rate'

dbase = np.load('EggDataset_bhp_rate_full_projeto.npy')

db=dbase[0:1463,:,:]
# print( db.shape)

# _________________________ Parameters _________________________
# define 0 to full injector rate wells;
# 1 to Liquid/Injetion rate wells;
# 2 to Oil/WaterP/WaterI
# 3 to one injector
```

```
well_type=2
n_timesteps = db.shape[1]
window_size =59
n_partitions = n_timesteps // window_size
# print(n_timesteps, window_size, n_partitions)

# Set split mode
# 0: normal partition (ignore last step)
# 1: partition with overlap
# 2: normal partition (ignore first step)
split_mode = 1

# _________________________ Inputs: Wells' BHP _________________________
prod_bhp = split_dataset_by_window(db, window_size, 0, 4*n_prod, 4, split_mode)

if type_control_inj=='bhp':
  inj_bhp = split_dataset_by_window(db, window_size, 4*n_prod, None, 2, split_mode)
elif type_control_inj=='rate':
  inj_bhp = split_dataset_by_window(db, window_size, 4*n_prod+1, None, 2, split_mode)

X = np.concatenate((prod_bhp, inj_bhp), axis = 2)

# Apply custom normalization
X, _ = scaler(X, 3)

# print(prod_bhp.shape, inj_bhp.shape, X.shape)

# _________________________ Outputs _________________________
# Producer Wells' oil Rate
y1 = split_dataset_by_window(db, window_size, 2, 4*n_prod, 4, split_mode)
prod_qwr_size = y1.shape[2]
# Producer Wells' water Rate
y2 = split_dataset_by_window(db, window_size, 3, 4*n_prod, 4, split_mode)
prod_qor_size = y2.shape[2]
# Injector Wells' Water Rate
if type_control_inj=='bhp':
  y3 = split_dataset_by_window(db, window_size, 4*n_prod+1, None, 2, split_mode)
elif type_control_inj=='rate':
  y3 = split_dataset_by_window(db, window_size, 4*n_prod, None, 2, split_mode)

inj_qwr_size = y3.shape[2]

y = np.concatenate((y1, y2, y3), axis = 2)

# Reshape to apply normalization
y_reshaped = y.reshape(y.shape[0] * y.shape[1], y.shape[2])
```

```
y_reshaped, sc = scaler(y_reshaped)

# Restore original dimension
y = y_reshaped.reshape(y.shape[0], y.shape[1], y.shape[2])

# print(y.shape)


# _________________________ Divisão dos dados _________________________
X_train, X_val, X_test = split_dataset(X, n_partitions)
y_train, y_val, y_test = split_dataset(y, n_partitions)

# print(X_train.shape, X_val.shape, X_test.shape)
# print(y_train.shape, y_val.shape, y_test.shape)

n_steps_in = X.shape[1]
n_features_in = X.shape[2]

n_steps_out = y.shape[1]
n_features_out = y.shape[2]

# print(n_steps_in, n_features_in)
# print(n_steps_out, n_features_out)

# Função para criar e compilar o modelo com os hiperparâmetros específicos
def create_model(neurons, batch_size, learning_rate, activation):
    inputs = Input(shape=(n_steps_in, n_features_in))
    layer2 = GRU(neurons, activation=activation, return_sequences=True)(inputs)
    output = tf.keras.layers.TimeDistributed(Dense(n_features_out))(layer2)
    model = Model(inputs=inputs, outputs=output)
    model.compile(
       optimizer=tf.keras.optimizers.Adam(learning_rate=learning_rate),
       loss='mse')
    return model

# _________________________ GRID SEARCH _________________________


# Parametros de teste
neurons_list = [350, 400]
batch_size_list = [4]
learning_rate_list = [0.001]
activation_list = ['linear', 'relu', 'tanh']

qntEpocas = 500
```

```
best_score = float('inf')
best_params = {}

monitor = EarlyStopping(monitor='val_loss',
                        min_delta=1e-4,
                        patience=10,
                        verbose=2,
                        mode='auto',
                        restore_best_weights=True)


with open('resultadosEarlyGRUExtra.csv', 'a', newline='') as f:
    writer = csv.writer(f)
    # Se o arquivo estiver vazio, escreva o cabeçalho
    if f.tell() == 0:
        writer.writerow(
           ['neurons',
            'batch_size',
            'learning_rate',
            'activation',
            'score'])

    # Realizar a pesquisa em grade manualmente
    for neurons in neurons_list:
        for batch_size in batch_size_list:
            for learning_rate in learning_rate_list:
                for activation in activation_list:
                    print(f"Testing model with neurons={neurons},
                          batch_size={batch_size},
                          learning_rate={learning_rate},
                          activation={activation}")

                    # Criar e compilar o modelo
                    model = create_model(neurons=neurons,
                                         batch_size=batch_size,
                                         learning_rate=learning_rate,
                                         activation=activation)

                    # Treinar o modelo
                    history = model.fit(X_train, y_train,
                                        callbacks=[monitor],
                                        batch_size=batch_size,
                                        epochs=qntEpocas,
                                        verbose=0,
                                        validation_split=0.2)
```

```
                    # Avaliar o modelo
                    score = model.evaluate(X_val, y_val)

                    # Atualizar os melhores hiperparâmetros se necessário
                    if score < best_score:
                        best_score = score
                        best_params = {'neurons': neurons,
                                       'batch_size': batch_size,
                                       'learning_rate': learning_rate,
                                       'activation': activation}

                    # Escrever os resultados no arquivo CSV
                    writer.writerow([neurons,
                                     batch_size,
                                     learning_rate,
                                     activation,
                                     score])

# Imprimir os melhores parâmetros e o melhor score
print("Melhores parâmetros encontrados: ", best_params)
print("Melhor score encontrado: ", best_score)
```