In [1]:

```
import os
os.environ["TF_CPP_MIN_LOG_LEVEL"] = "2"
```

In [2]:

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import matplotlib.pyplot as plt
import seaborn as sns
import csv
import datetime

import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import StandardScaler
from threading import Timer
from timeit import default_timer as timer
from IPython.display import clear_output
```

checking GPU erros

In [3]:

```
# physical_devices = tf.config.list_physical_devices("GPU")
# physical_devices
# tf.config.experimental.set_memory_growth(physical_devices[0], True)
```

In [4]:

```
def preprocessiing():

    prep_dataset1 = pd.read_csv('../datasets/dataset_test_02_07.csv', delimit
    df = prep_dataset1.iloc[:,1:4]
    WINDOW = 35

    for i in np.arange(df.shape[0]):
        init = i*WINDOW
        init2 = (i+1)*WINDOW
        if(init2<df.shape[0]):
            df.iloc[init:init+WINDOW,2] =  df.iloc[init2,2]
    df2 = normalizing(df)
    train_size = int(len(df2) * 0.8)
    test_size = len(df2) - train_size
    return df2.iloc[0:train_size], df2.iloc[train_size:len(df2)]
```

In [5]:

```
def normalizing(dataset):
    df_norm = pd.read_csv('../datasets/dataset_test_02_07.csv', delimiter=","
    df_norm = df_norm.iloc[:,1:4]
    scaler = StandardScaler().fit(df_norm)

    scaler = scaler.fit(df_norm[['delay']])

    dataset['delay'] = scaler.transform(dataset[['delay']])
    return dataset


def unormalizing(Y_test,y_pred ):
    df_norm = pd.read_csv('../datasets/dataset_test_02_07.csv', delimiter=","
    df_norm = df_norm.iloc[:,1:4]
    scaler = StandardScaler().fit(df_norm)
    scaler = scaler.fit(df_norm[['delay']])
    y_test_inv = scaler.inverse_transform(Y_test.reshape(1,-1))
    y_pred_inv = scaler.inverse_transform(y_pred)

    return y_test_inv, y_pred_inv
```

In [6]:

```
def create_dataset(X, y, time_steps=1):
    Xs, ys = [], []
    for i in range(len(X) - time_steps):
        clear_output(wait=True)
        print('modeling to keras ',round((i/(len(X) - time_steps))*100,2), ('
        s = round(timer() - start)
        if(s>60):
            s /=60
            print(' ', s, ' seconds')
        v = X.iloc[i: (i+time_steps), 2:3].to_numpy()
        Xs.append(v)
        ys.append(y.iloc[i+time_steps])
    return np.array(Xs), np.array(ys)
```

In [7]:

```
def LSTMconf(X_train):
    print('Init config LSTM')
    model = keras.Sequential()
    model.add(
            keras.layers.LSTM(
                units=512,
                input_shape=(X_train.shape[1],X_train.shape[2]),
                 kernel_initializer="glorot_uniform",
                unit_forget_bias=True,
                recurrent_dropout=0.75,
            )
        )

    model.add(keras.layers.Dense(units=1024, ))
    model.add(keras.layers.Dropout(rate=0.75))
    model.add(keras.layers.Dense(units=512, ))
#     model.add(keras.layers.Dropout(rate=0.5))
    model.add(keras.layers.Dense(units=1024, ))
#     model.add(keras.layers.Dropout(rate=0.3))
    model.add(keras.layers.Dense(units=512, ))
#     model.add(keras.layers.Dropout(rate=0.2))
    model.add(keras.layers.Dense(units=512, ))
    model.add(keras.layers.Dense(units=1))

    loss ="mse"
    optim = tf.keras.optimizers.Adam(
    learning_rate=0.0001)


    model.compile(loss=loss, optimizer=optim,
             )


    return model
```

In [8]:

```
# batch_size = round(X_train.shape[0]*0.1)
# print(batch_size)
```

In [9]:

```
def LSTMfit(model,X_train,Y_train):
    print('Init Train')
    start = timer()
    log_dir = "logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
    tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir, hi
    batch_size = round(X_train.shape[0]*0.08)
    history = model.fit(
        X_train, Y_train,
        epochs=10,
        batch_size= batch_size,
        validation_split=0.1,
        shuffle=False,
        callbacks=[tensorboard_callback]
    )
    return history
```

In [10]:

```
start = timer()
train, test = preprocessiing()
```

In [11]:

```
X_train,Y_train = create_dataset(train, train.delay)
model = LSTMconf(X_train)
history = LSTMfit(model,X_train, Y_train)
```

```
modeling to keras  99.97 %Init config LSTM
Init Train
Epoch 1/10
12/12 [==============================] - 4s 150ms/step - loss:
0.5472 - val_loss: 0.0160
Epoch 2/10
12/12 [==============================] - 1s 60ms/step - loss:
0.2487 - val_loss: 0.0024
Epoch 3/10
12/12 [==============================] - 1s 60ms/step - loss:
0.2921 - val_loss: 0.0796
Epoch 4/10
12/12 [==============================] - 1s 63ms/step - loss:
0.0662 - val_loss: 0.0023
Epoch 5/10
12/12 [==============================] - 1s 64ms/step - loss:
0.0799 - val_loss: 7.6086e-05
Epoch 6/10
12/12 [==============================] - 1s 64ms/step - loss:
0.0807 - val_loss: 0.0130
Epoch 7/10
12/12 [==============================] - 1s 70ms/step - loss:
0.0139 - val_loss: 0.0018
Epoch 8/10
12/12 [==============================] - 1s 69ms/step - loss:
0.0124 - val_loss: 2.5564e-04
Epoch 9/10
12/12 [==============================] - 1s 69ms/step - loss:
0.0137 - val_loss: 1.5448e-04
Epoch 10/10
12/12 [==============================] - 1s 70ms/step - loss:
0.0122 - val_loss: 2.6952e-05
```

In [12]:

```
# Load the TensorBoard notebook extension
# %load_ext tensorboard
# %tensorboard --logdir logs/fit
```

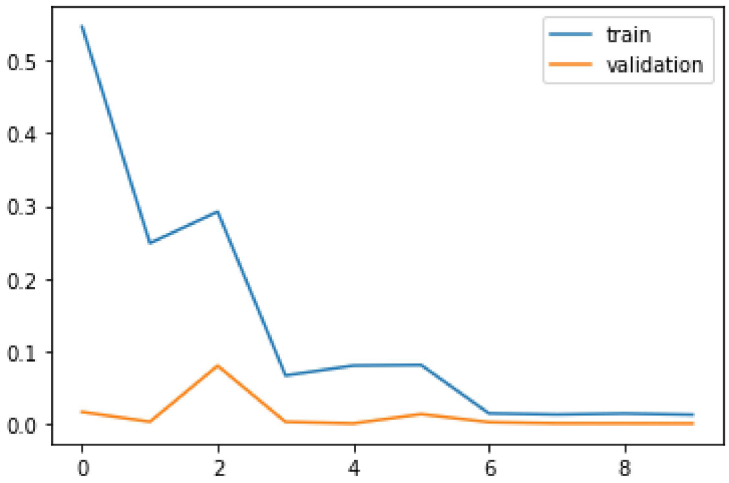
In [13]:

```
# print('Saving Model')
# model.save('models/lstm_mininet')
```

loss training

In [14]:

```
fig1 = plt.figure()
ax1 = fig1.add_subplot(1,1,1)
ax1.plot(history.history['loss'], label='train')
ax1.plot(history.history['val_loss'], label='validation')
ax1.legend();
```



In [15]:

```
X_test,Y_test = create_dataset(test, test.delay)
```

```
modeling to keras  99.9 %
```

predicting

In [16]:

```
y_pred = model.predict(X_test)
```

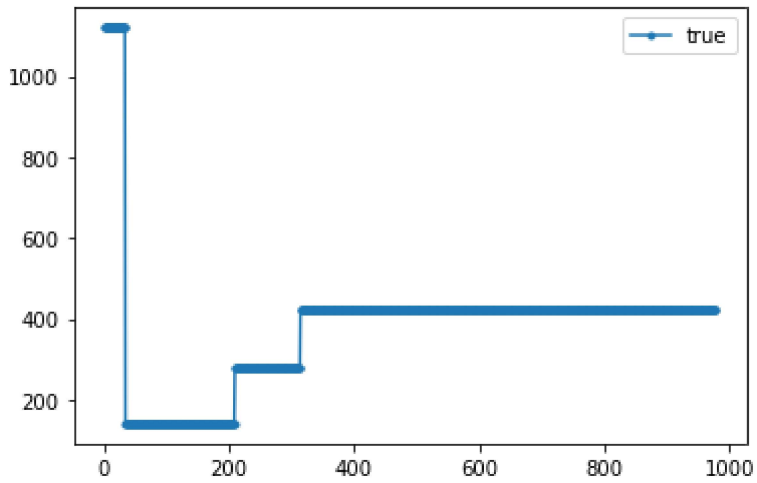
In [17]:

```
y_test_inv, y_pred_inv = unormalizing(Y_test, y_pred)
```

unormalizing

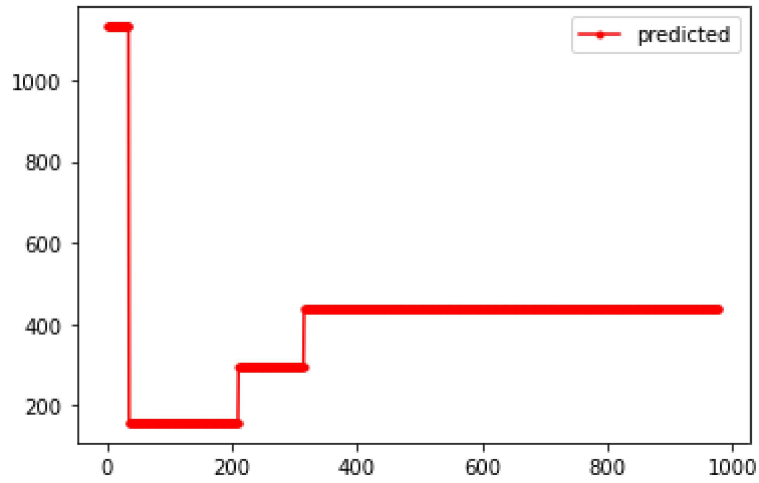
In [18]:

```
fig2 = plt.figure()
a2 = fig2.add_subplot(1,1,1)
a2.plot(y_test_inv.flatten(), marker='.', label='true')
a2.legend();
```



In [19]:

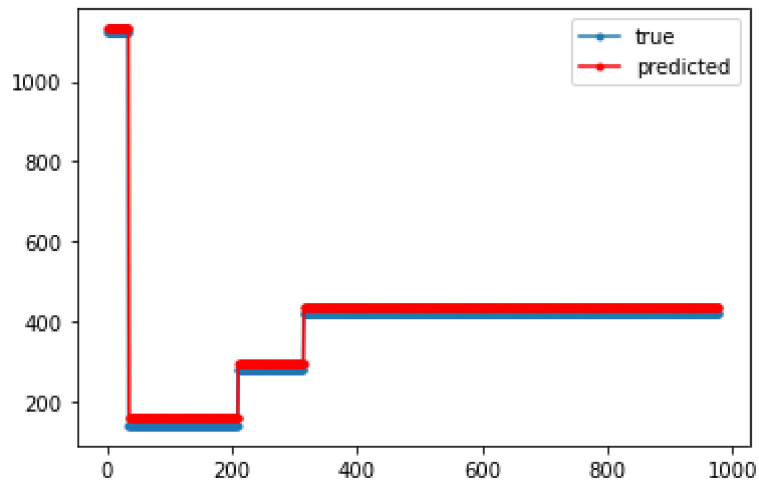
```
fig3 = plt.figure()
a3 = fig3.add_subplot(1,1,1)
a3.plot(y_pred_inv.flatten(),'r',marker='.', label='predicted')
a3.legend();
```



In [20]:

```
fig4 = plt.figure()
a4 = fig4.add_subplot(1,1,1)

a4.plot(y_test_inv.flatten(), marker='.', label='true')
a4.plot(y_pred_inv.flatten(),'r',marker='.', label='predicted')
a4.legend();
```



In [21]:

```
from sklearn.metrics import mean_squared_error
from sklearn.metrics import median_absolute_error
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_log_error
from sklearn.metrics import explained_variance_score
```

In [22]:

```
size = np.min([y_pred_inv.shape[0],y_test_inv.shape[0] ])
rmse =  mean_squared_error(y_test_inv.flatten()[0:size], y_pred_inv.flatten()
mae =  mean_absolute_error(y_test_inv.flatten()[0:size], y_pred_inv.flatten()
median_mae = median_absolute_error(y_test_inv.flatten()[0:size], y_pred_inv.f
evs = explained_variance_score(y_test_inv.flatten()[0:size], y_pred_inv.flatt

print(rmse)
print(mae)
print(median_mae)
print('Explained Variance Score: ',evs)
```

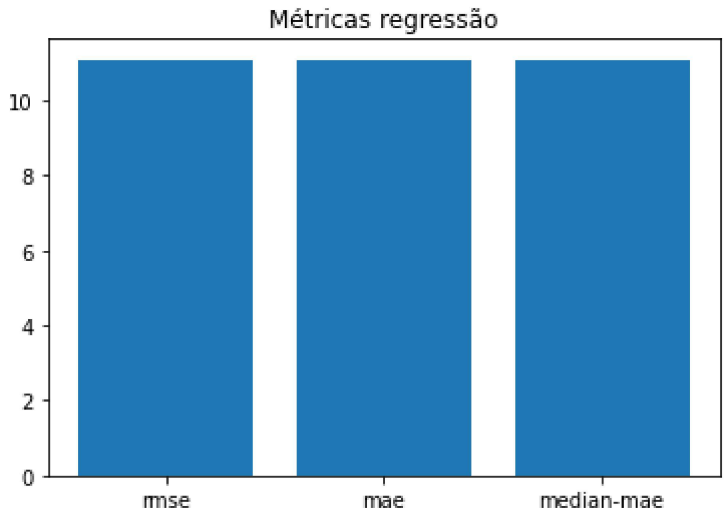
```
11.088162660598528
11.088162660598528
11.088162660598528
Explained Variance Score:  1.0
```

In [23]:

```
objects = ('rmse', 'mae', 'median-mae')
y_pos = np.arange(3)
performance = [rmse,mae,median_mae]

plt.bar(y_pos, performance, align='center')
plt.xticks(y_pos, objects)
#plt.ylabel('Usage')
plt.title('Métricas regressão')

plt.show()
```



In [24]:

```
batch_size = round(X_train.shape[0]*0.08)
batch_size
```

Out[24]:

```
314
```

In [ ]:

In [ ]:

In [ ]:

```
```