In [1]:

```
import os
import tensorflow as tf
from tensorflow import keras

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error
from timeit import default_timer as timer
from IPython.display import clear_output
```

In [2]:

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

In [3]:

```
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), 2:4].to_numpy()
        Xs.append(v)
        ys.append(y.iloc[i+time_steps])
    return np.array(Xs), np.array(ys)
```

In [4]:

```
#carregando datasets
print('loading dataset...')
train = pd.read_csv('../datasets/com_concept_drift/sdn_train_unormalized.csv'
test = pd.read_csv('../datasets/com_concept_drift/sdn_test_unormalized.csv')

train = train[train.delay>=0]
test = test[test.delay>=0]

train = train[train.delay<=10000]
test = test[test.delay<=10000]
```

```
loading dataset...
```

In [5]:

```
start = timer()
print('creating window')
TIME_STEPS = 1
X_train,Y_train = create_dataset(train, train.delay, time_steps=TIME_STEPS)
X_test,Y_test = create_dataset(test, test.delay, time_steps=TIME_STEPS)

print('2D to 3D duration: ', round(timer() - start))
```

```
modeling to keras  97.01 %  10.566666666666666  seconds
```

setting LSTM

In [6]:

```
print('Init config LSTM')
model = keras.Sequential()
model.add(
    keras.layers.Bidirectional(
        keras.layers.LSTM(
            units=40,
            input_shape=(X_train.shape[1],X_train.shape[2])
        )
    ))
model.add(keras.layers.Dense(units=40))
model.add(keras.layers.Dense(units=40))
model.add(keras.layers.Dense(units=40))
model.add(keras.layers.Dense(units=40))
model.add(keras.layers.Dropout(rate=0.2))
model.add(keras.layers.Dense(units=1))
```

compiling

In [7]:

```
loss ="mse"
optim = tf.keras.optimizers.Adam(
    learning_rate=0.0001)
metrics=["accuracy"]

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

training

In [8]:

```
print('Init Train')
start = timer()
history = model.fit(
    X_train, Y_train,
    epochs=100,
    batch_size= 10,
    validation_split=0.1,
    shuffle=False,
#     callbacks=[tensorboard_callback]
)
print('trraining duration: ',round(timer() - start))
```

```
Epoch 23/100
4995/4995 [==============================] - 21s 4ms/step -
loss: 2243.3535 - accuracy: 0.5727 - val_loss: 13514.7246 -

val_accuracy: 0.0000e+00
Epoch 24/100
4995/4995 [==============================] - 21s 4ms/step -
loss: 2255.3489 - accuracy: 0.5802 - val_loss: 13346.8643 -
val_accuracy: 0.0000e+00
Epoch 25/100
4995/4995 [==============================] - 21s 4ms/step -
loss: 2198.2566 - accuracy: 0.5739 - val_loss: 12825.2744 -
val_accuracy: 0.0000e+00
Epoch 26/100
4995/4995 [==============================] - 21s 4ms/step -
loss: 2200.0188 - accuracy: 0.5761 - val_loss: 12803.1289 -
val_accuracy: 0.0000e+00
Epoch 27/100
4995/4995 [==============================] - 21s 4ms/step -
loss: 2272.0469 - accuracy: 0.5807 - val_loss: 12176.5693 -
```

saving model

In [9]:

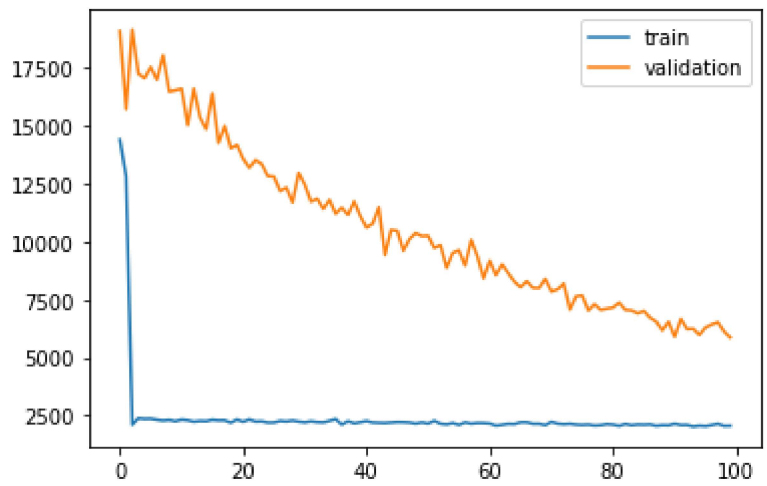
```
print('Saving Model')
model.save('models/lstm')
```

```
Saving Model

WARNING:absl:Found untraced functions such as lstm_cell_1_layer
_call_and_return_conditional_losses, lstm_cell_1_layer_call_fn,
lstm_cell_2_layer_call_and_return_conditional_losses, lstm_cell
_2_layer_call_fn, lstm_cell_1_layer_call_fn while saving (showi
ng 5 of 10). These functions will not be directly callable afte
r loading.

INFO:tensorflow:Assets written to: models/lstm\assets

INFO:tensorflow:Assets written to: models/lstm\assets
```

loss training

In [10]:

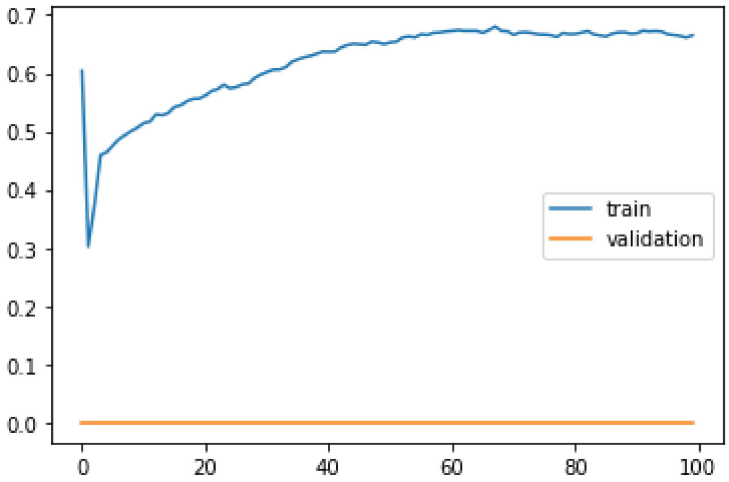
```
fig1 = plt.figure()
ax1 = fig1.add_subplot(1,1,1)
ax1.plot(history.history['loss'], label='train')
ax1.plot(history.history['val_loss'], label='validation')
ax1.legend();
```



accuracy

In [11]:

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



predicting

In [12]:

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

unormalizing

In [13]:

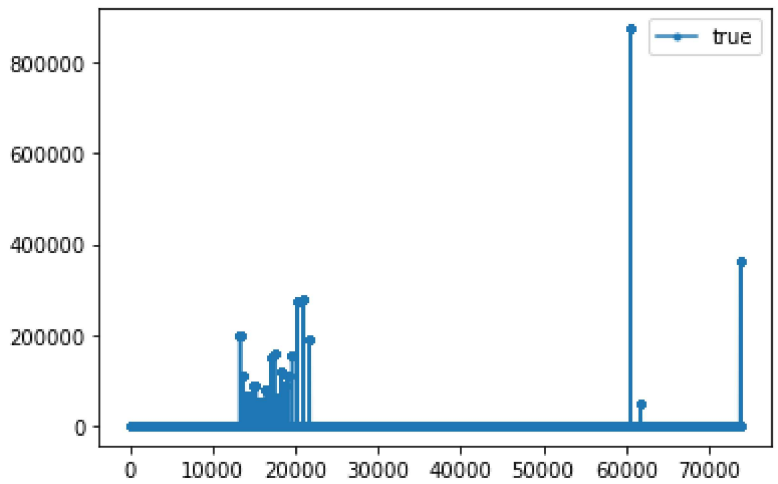
```
f_columns = ['temperature','label']
scaler1 = StandardScaler().fit(train[f_columns])
scaler2 = StandardScaler().fit(train[f_columns])

scaler1= scaler1.fit(train[f_columns].to_numpy())
scaler2 = scaler2.fit(train[['delay']])


#normalizando test
scaler3 = StandardScaler().fit(test[f_columns])
scaler4 = StandardScaler().fit(test[f_columns])

scaler3 = scaler3.fit(test[f_columns].to_numpy())
scaler4 = scaler4.fit(test[['delay']])
```

In [14]:

```
y_test_inv = scaler4.inverse_transform(Y_test.reshape(1,-1))
y_pred_inv = scaler4.inverse_transform(y_pred)
```

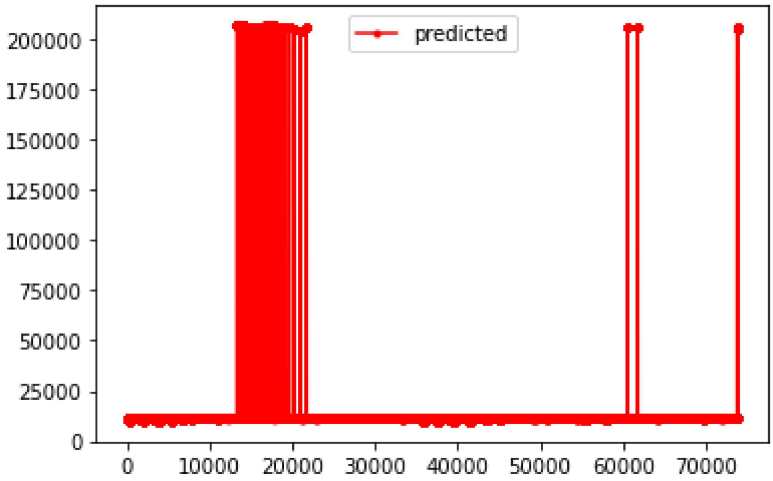
In [15]:

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



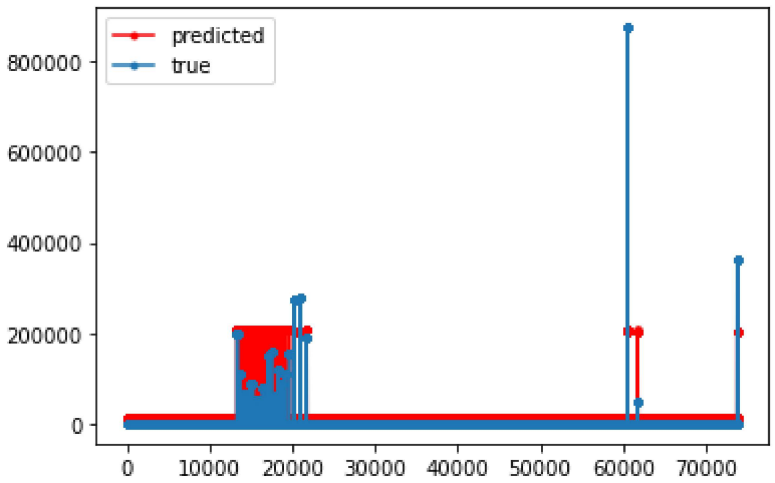
In [16]:

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



In [17]:

```
fig4 = plt.figure()
a4 = fig4.add_subplot(1,1,1)
a4.plot(y_pred_inv.flatten(),'r',marker='.', label='predicted')
a4.plot(y_test_inv.flatten(), marker='.', label='true')
a4.legend();
```



In [18]:

```
from sklearn.metrics import mean_squared_error
```

In [19]:

```
mean_squared_error(y_test_inv[0], y_pred_inv[:,0])
```

Out[19]:

```
668760452.3832275
```

In [ ]:

```

```

In [ ]:

```

```