1 Timeseries Multi-Step Multi-Output

Data treatment for model 1 Features:

- local lateral position Local_X, to account for different behaviors depending on the driving lane,
- local longitudinal position Local_Y, to account for different behaviors when approaching the merging lane,
- lateral and longitudinal velocities vx and vy,
- type (motorcycle, car or truck)

1.1 Import packages

In [5]:

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
import tensorflow as tf

plt.rcParams['figure.figsize'] = (8, 6)
```

executed in 9.88s, finished 15:16:12 2020-08-21

1.2 Load dataset

In [6]:

```
url_1 = 'https://github.com/duonghung86/Vehicle-trajectory-tracking/raw/master/
zip_path = tf.keras.utils.get_file(origin=url_1, fname=url_1.split('/')[-1], ex
csv_path = zip_path.replace('zip','csv')
csv_path
```

executed in 2.16s, finished 15:16:15 2020-08-21

Out[6]:

```
'C:\\Users\\DuongHung\\.keras\\datasets\\0750_0805_us101_smoothed_11_.csv'
```

Let's take a glance at the data. Here are the first few rows:

In [7]:

```
df = pd.read_csv(csv_path)
df.info()
df.head()
```

executed in 3.90s, finished 15:16:19 2020-08-21

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1048575 entries, 0 to 1048574
Data columns (total 18 columns):
 #   Column        Non-Null Count    Dtype  
---  ------        --------------    -----  
 0   Vehicle_ID    1048575 non-null  int64  
 1   Frame_ID      1048575 non-null  int64  
 2   Total_Frames  1048575 non-null  int64  
 3   Global_Time   1048575 non-null  int64  
 4   Local_X       1048575 non-null  float64
 5   Local_Y       1048575 non-null  float64
 6   Global_X      1048575 non-null  float64
 7   Global_Y      1048575 non-null  float64
 8   v_Length      1048575 non-null  float64
 9   v_Width       1048575 non-null  float64
 10  v_Class       1048575 non-null  int64  
 11  v_Vel         1048575 non-null  float64
 12  v_Acc         1048575 non-null  float64
 13  Lane_ID       1048575 non-null  int64  
 14  Preceeding    1048575 non-null  int64  
 15  Following     1048575 non-null  int64  
 16  Space_Hdwy    1048575 non-null  float64
 17  Time_Hdwy     1048575 non-null  float64
dtypes: float64(10), int64(8)
memory usage: 144.0 MB
```

Out[7]:

	Vehicle_ID	Frame_ID	Total_Frames	Global_Time	Local_X	Local_Y	Global_X	Glob
0	2	13	437	1118846980200	16.467196	35.380427	6451137.641	1873344
1	2	14	437	1118846980300	16.446594	39.381608	6451140.329	1873342
2	2	15	437	1118846980400	16.425991	43.381541	6451143.018	1873339
3	2	16	437	1118846980500	16.405392	47.380780	6451145.706	1873336
4	2	17	437	1118846980600	16.384804	51.379881	6451148.395	1873333

Next look at the statistics of the dataset:

In [8]:
```
df.describe().transpose().round(3)
```
executed in 1.51s, finished 15:16:20 2020-08-21

Out[8]:

	count	mean	std	min	25%	50%	
Vehicle_ID	1048575.0	1.533080e+03	790.271	2.000000e+00	9.320000e+02	1.574000e+03	2.
Frame_ID	1048575.0	4.518249e+03	2412.479	8.000000e+00	2.455000e+03	4.586000e+03	6.
Total_Frames	1048575.0	5.608770e+02	146.577	1.770000e+02	4.640000e+02	5.180000e+02	6.
Global_Time	1048575.0	1.118847e+12	241247.914	1.118847e+12	1.118847e+12	1.118847e+12	1.
Local_X	1048575.0	2.940600e+01	16.666	5.340000e-01	1.728400e+01	2.955700e+01	4.
Local_Y	1048575.0	1.002056e+03	596.357	1.796600e+01	4.883960e+02	9.640280e+02	1.
Global_X	1048575.0	6.451838e+06	446.275	6.451107e+06	6.451450e+06	6.451808e+06	6.
Global_Y	1048575.0	1.872677e+06	397.006	1.871875e+06	1.872352e+06	1.872699e+06	1.
v_Length	1048575.0	1.463500e+01	4.870	4.000000e+00	1.200000e+01	1.450000e+01	1.
v_Width	1048575.0	6.132000e+00	1.037	2.000000e+00	5.400000e+00	6.000000e+00	6.
v_Class	1048575.0	2.009000e+00	0.191	1.000000e+00	2.000000e+00	2.000000e+00	2.
v_Vel	1048575.0	3.877400e+01	14.110	0.000000e+00	3.031700e+01	3.989800e+01	4.
v_Acc	1048575.0	3.610000e-01	5.852	-3.193080e+02	-1.752000e+00	1.700000e-02	2.
Lane_ID	1048575.0	2.956000e+00	1.469	1.000000e+00	2.000000e+00	3.000000e+00	4.
Preceeding	1048575.0	1.459864e+03	844.319	0.000000e+00	7.880000e+02	1.519000e+03	2.
Following	1048575.0	1.477269e+03	843.679	0.000000e+00	8.120000e+02	1.533000e+03	2.
Space_Hdwy	1048575.0	7.815800e+01	48.615	0.000000e+00	4.984000e+01	6.911000e+01	9.
Time_Hdwy	1048575.0	1.090800e+02	1027.551	0.000000e+00	1.460000e+00	1.970000e+00	2.

In [9]:
```
df.columns
```
executed in 13ms, finished 15:16:20 2020-08-21

Out[9]:
```
Index(['Vehicle_ID', 'Frame_ID', 'Total_Frames', 'Global_Time', 'Local_X',
       'Local_Y', 'Global_X', 'Global_Y', 'v_Length', 'v_Width', 'v_Class',
       'v_Vel', 'v_Acc', 'Lane_ID', 'Preceeding', 'Following', 'Space_Hdwy',
       'Time_Hdwy'],
      dtype='object')
```

In [10]:
```
#  keep only columns that are useful for now
kept_cols = ['Vehicle_ID', 'Frame_ID', 'Total_Frames', 'Local_X','Local_Y']
df = df[kept_cols]
df.head()
```
executed in 69ms, finished 15:16:20 2020-08-21

Out[10]:

	Vehicle_ID	Frame_ID	Total_Frames	Local_X	Local_Y
0	2	13	437	16.467196	35.380427
1	2	14	437	16.446594	39.381608
2	2	15	437	16.425991	43.381541
3	2	16	437	16.405392	47.380780
4	2	17	437	16.384804	51.379881

In [11]:
```
'the number of vehicles is {}'.format(len(df.Vehicle_ID.unique()))
```
executed in 44ms, finished 15:16:20 2020-08-21

Out[11]: 'the number of vehicles is 1993'

In [12]:
```
# let use only 1000 vehicle to reduce the computation workload
vehicle_list = df.Vehicle_ID.unique()
n_veh = 100 # number of vehicles
np.random.seed(48)
new_veh_list = np.random.choice(vehicle_list,n_veh)
print(new_veh_list)
```
executed in 42ms, finished 15:16:20 2020-08-21

```
[2149  901 1903 2061  567 1362 2593 1396  736 1841  570  346  394 2349
   10 1727 1339 2571 2741 2444 1176 1744 2587 2270 2328  869 2218 2167
  711 2167  389 1518  458 2255  686  190 1175 2589 1675  701  204 2567
 1690 2706  641   73 1785 1844 2126 1975 1942 2171  905 2161 1038  854
  212 1702  798 1259 2336  585 2691 1183 2202  729 2219  635 1106 2164
 1455 2136 1767 1044 2310   75  888 1605 1976 2209  950  790 1001 2249
 2255 2150 2172 1552  417 1149  923 1577 1368  825 1717 1953 2336 2211
  374 2384]
```

In [13]:

```
new_df = df[df.Vehicle_ID.isin(new_veh_list)]
new_df.info()
new_df.head()
```

executed in 136ms, finished 15:16:20 2020-08-21

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 51316 entries, 2454 to 1029946
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   Vehicle_ID    51316 non-null  int64  
 1   Frame_ID      51316 non-null  int64  
 2   Total_Frames  51316 non-null  int64  
 3   Local_X       51316 non-null  float64
 4   Local_Y       51316 non-null  float64
dtypes: float64(2), int64(3)
memory usage: 2.3 MB
```

Out[13]:

	Vehicle_ID	Frame_ID	Total_Frames	Local_X	Local_Y
2454	10	39	436	4.311965	35.406783
2455	10	40	436	4.289860	39.935881
2456	10	41	436	4.268287	44.330462
2457	10	42	436	4.247104	48.609480
2458	10	43	436	4.226170	52.791893

2 Data transformation

2.1 1 object and 1 target variable

2.1.1 Prepare the data set

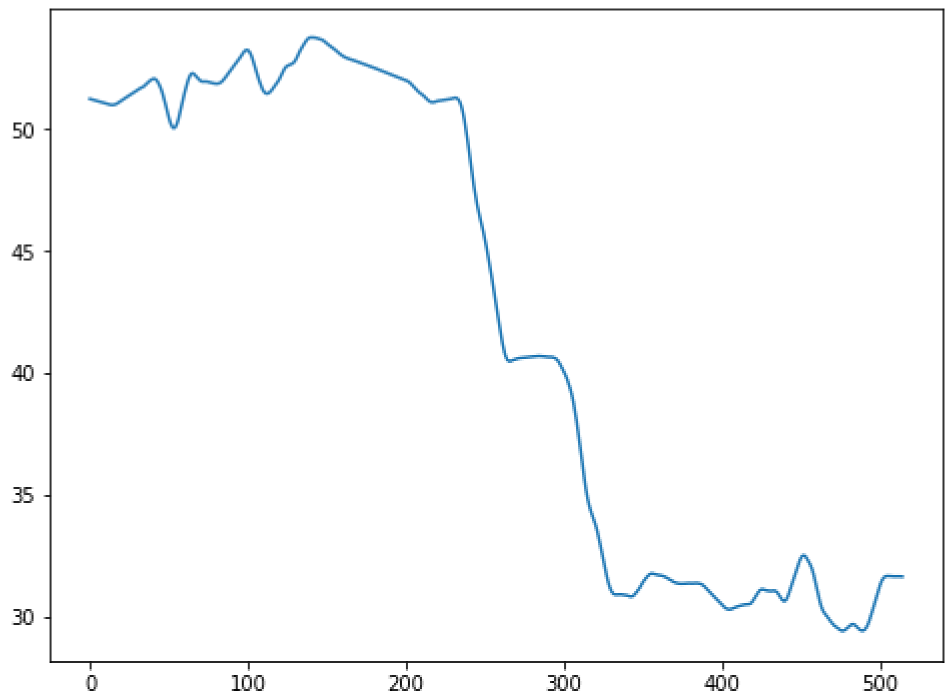
In [14]:

```
simple_df = new_df[new_df.Vehicle_ID == new_veh_list[0]].copy()
simple_df = simple_df[['Frame_ID','Local_X']]
simple_df.set_index('Frame_ID', inplace = True)
simple_df.sort_index(inplace=True)
simple_df.reset_index(drop=True, inplace=True)
plt.plot(simple_df)
simple_df.head()
```

executed in 421ms, finished 15:16:21 2020-08-21

Out[14]:

	Local_X
0	51.214252
1	51.196441
2	51.178494
3	51.160446
4	51.142331



In [ ]:

```

```

In [15]:

```
def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):
    """
    Frame a time series as a supervised learning dataset.
    Arguments:
    data: Sequence of observations as a list or NumPy array.
    n_in: Number of lag observations as input (X).
    n_out: Number of observations as output (y).
    dropnan: Boolean whether or not to drop rows with NaN values.
    Returns:
    Pandas DataFrame of series framed for supervised learning.
    """
    #n_vars = 1 if type(data) is list else data.shape[1]
    variables = list(data.columns)
    df = data.copy()
    cols, names = list(), list()
    # input sequence (t-n, ... t-1)
    for i in range(n_in, 0, -1):
        cols.append(df.shift(i))
        names += ['{}(t-{})'.format(j, i) for j in variables]
    # forecast sequence (t, t+1, ... t+n)
    for i in range(0, n_out):
        cols.append(df.shift(-i))
        if i == 0:
            names += ['{}(t)'.format(j) for j in variables]
        else:
            names += ['{}(t+{})'.format(j, i) for j in variables]
    # put it all together
    agg = pd.concat(cols, axis=1)
    agg.columns = names
    # drop rows with NaN values
    if dropnan:
        agg.dropna(inplace=True)
    return agg
```

executed in 29ms, finished 15:16:21 2020-08-21

In [30]:

```
series_to_supervised(simple_df, n_in=4, n_out=1, dropnan=False).head()
```

executed in 22ms, finished 15:22:16 2020-08-21

Out[30]:

	Local_X(t-4)	Local_X(t-3)	Local_X(t-2)	Local_X(t-1)	Local_X(t)
0	NaN	NaN	NaN	NaN	51.214252
1	NaN	NaN	NaN	51.214252	51.196441
2	NaN	NaN	51.214252	51.196441	51.178494
3	NaN	51.214252	51.196441	51.178494	51.160446
4	51.214252	51.196441	51.178494	51.160446	51.142331

In [33]:
```
transformed_df = series_to_supervised(simple_df, n_in=4, n_out=1, dropnan=True)
transformed_df.head()
```
executed in 32ms, finished 15:22:40 2020-08-21

Out[33]:

	Local_X(t-4)	Local_X(t-3)	Local_X(t-2)	Local_X(t-1)	Local_X(t)
4	51.214252	51.196441	51.178494	51.160446	51.142331
5	51.196441	51.178494	51.160446	51.142331	51.124182
6	51.178494	51.160446	51.142331	51.124182	51.106019
7	51.160446	51.142331	51.124182	51.106019	51.088021
8	51.142331	51.124182	51.106019	51.088021	51.069916

In [18]:
```
### Split the data set
from sklearn.model_selection import train_test_split
```
executed in 456ms, finished 15:16:21 2020-08-21

In [34]:
```
transformed_df.iloc[:,:-1]
```
executed in 25ms, finished 15:22:44 2020-08-21

Out[34]:

	Local_X(t-4)	Local_X(t-3)	Local_X(t-2)	Local_X(t-1)
4	51.214252	51.196441	51.178494	51.160446
5	51.196441	51.178494	51.160446	51.142331
6	51.178494	51.160446	51.142331	51.124182
7	51.160446	51.142331	51.124182	51.106019
8	51.142331	51.124182	51.106019	51.088021
...	...	...	...	...
510	31.668082	31.660184	31.653473	31.653089
511	31.660184	31.653473	31.653089	31.655713
512	31.653473	31.653089	31.655713	31.657416
513	31.653089	31.655713	31.657416	31.656984
514	31.655713	31.657416	31.656984	31.653203

511 rows × 4 columns

In [35]:
```
X_train, X_test, y_train, y_test = train_test_split(transformed_df.iloc[:,:-1],
                                                    test_size=0.3, random_state
print(X_train.shape,X_test.shape, y_train.shape, y_test.shape)
X_train.shape
```
executed in 27ms, finished 15:23:01 2020-08-21

```
(357, 4) (154, 4) (357,) (154,)
```

Out[35]:
```
(357, 4)
```

In [36]:
```
### Standardize the data
train_mean = X_train.mean()
train_std = X_train.std()

X_train = (X_train - train_mean) / train_std
X_test = (X_test - train_mean) / train_std
```
executed in 36ms, finished 15:23:03 2020-08-21

In [37]:
```
print(X_train.describe())
X_train.shape
```
executed in 55ms, finished 15:23:05 2020-08-21

```
       Local_X(t-4)  Local_X(t-3)  Local_X(t-2)  Local_X(t-1)
count  3.570000e+02  3.570000e+02  3.570000e+02  3.570000e+02
mean   1.475322e-15  1.174286e-15  2.308766e-15 -6.692437e-16
std    1.000000e+00  1.000000e+00  1.000000e+00  1.000000e+00
min   -1.367345e+00 -1.361572e+00 -1.358854e+00 -1.354419e+00
25%   -1.166080e+00 -1.161489e+00 -1.156922e+00 -1.152961e+00
50%    2.488613e-01  2.166698e-01  1.828833e-01  1.472498e-01
75%    9.506863e-01  9.562820e-01  9.600661e-01  9.638601e-01
max    1.136656e+00  1.140431e+00  1.144156e+00  1.147849e+00
```

Out[37]:
```
(357, 4)
```

2.1.2 Apply prediction model

In [28]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import LSTM
#from sklearn.preprocessing import MinMaxScaler
#from sklearn.metrics import mean_squared_error
```
executed in 15ms, finished 15:19:16 2020-08-21

2.1.2.1 Vanilla LSTM

A Vanilla LSTM is an LSTM model that has a single hidden layer of LSTM units, and an output layer used to make a prediction.

In [38]:
```
n_steps = 4
n_features = 1
```
executed in 12ms, finished 15:26:23 2020-08-21

In [42]:

```
X_train = X_train.values
X_test = X_test.values
X_train = X_train.reshape((X_train.shape[0], X_train.shape[1], n_features))
X_test = X_test.reshape((X_test.shape[0], X_test.shape[1], n_features))
```

executed in 9ms, finished 15:28:49 2020-08-21

In [45]:

```
# define model
model = Sequential()
model.add(LSTM(50, activation='relu', input_shape=(n_steps, n_features)))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
# fit model
model.fit(X_train, y_train, epochs=100,validation_data=(X_test, y_test), verbos
```

executed in 18.1s, finished 15:29:51 2020-08-21

```
357/357 [==============================] - 0s 431us/sample - loss: 0.2758 - v
al_loss: 0.2864
Epoch 96/100
357/357 [==============================] - 0s 403us/sample - loss: 0.2471 - v
al_loss: 0.2800
Epoch 97/100
357/357 [==============================] - 0s 403us/sample - loss: 0.2383 - v
al_loss: 0.2688
Epoch 98/100
357/357 [==============================] - 0s 468us/sample - loss: 0.2282 - v
al_loss: 0.2698
Epoch 99/100
357/357 [==============================] - 0s 426us/sample - loss: 0.2395 - v
al_loss: 0.2625
Epoch 100/100
357/357 [==============================] - 0s 409us/sample - loss: 0.2295 - v
al_loss: 0.2801
```

Out[45]:

```
<tensorflow.python.keras.callbacks.History at 0x16cf6200f88>
```

In [46]:

```
yhat = model.predict(X_test, verbose=1)
#print(yhat)
```

executed in 408ms, finished 15:30:43 2020-08-21

```
154/154 [==============================] - 0s 2ms/sample
```

In [89]:
```
plt.scatter(y_test.index,y_test, label = "true label",marker = 'X', )
plt.scatter(y_test.index,yhat, label = "prediction",marker = '.')
plt.legend()
plt.show()
```

executed in 265ms, finished 15:47:16 2020-08-21