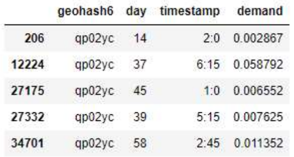
Testing the Model

How to use:

If the test dataset is in the same format as the train dataset provided,

Example:

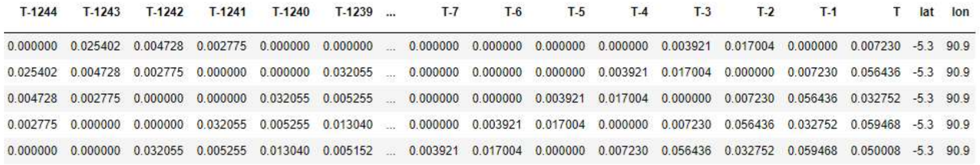


	geohash6	day	timestamp	demand
206	qp02yc	14	2:0	0.002867
12224	qp02yc	37	6:15	0.058792
27175	qp02yc	45	1:0	0.006552
27332	qp02yc	39	5:15	0.007625
34701	qp02yc	58	2:45	0.011352

Please change the file path under **section III**

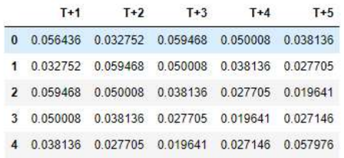
Else if the dataset is similar to the input and output of the model, of the following:

Input:
[demand on T-1248, demand on T-1247 ... demand on T, latitude, longitude] **1251** columns



T-1244	T-1243	T-1242	T-1241	T-1240	T-1239	...	T-7	T-6	T-5	T-4	T-3	T-2	T-1	T	lat	lon
0.000000	0.025402	0.004728	0.002775	0.000000	0.000000	...	0.000000	0.000000	0.000000	0.000000	0.003921	0.017004	0.000000	0.007230	-5.3	90.9
0.025402	0.004728	0.002775	0.000000	0.000000	0.032055	...	0.000000	0.000000	0.000000	0.003921	0.017004	0.000000	0.007230	0.056436	-5.3	90.9
0.004728	0.002775	0.000000	0.000000	0.032055	0.005255	...	0.000000	0.000000	0.003921	0.017004	0.000000	0.007230	0.056436	0.032752	-5.3	90.9
0.002775	0.000000	0.000000	0.032055	0.005255	0.013040	...	0.000000	0.003921	0.017004	0.000000	0.007230	0.056436	0.032752	0.059468	-5.3	90.9
0.000000	0.000000	0.032055	0.005255	0.013040	0.005152	...	0.003921	0.017004	0.000000	0.007230	0.056436	0.032752	0.059468	0.050008	-5.3	90.9

Output:
[demand on T+1,... demand on T+5] **5** columns



	T+1	T+2	T+3	T+4	T+5
0	0.056436	0.032752	0.059468	0.050008	0.038136
1	0.032752	0.059468	0.050008	0.038136	0.027705
2	0.059468	0.050008	0.038136	0.027705	0.019641
3	0.050008	0.038136	0.027705	0.019641	0.027146
4	0.038136	0.027705	0.019641	0.027146	0.057976

Please make them into float and pass directly into the prediction model under **section IV**

Else

Please try to process the test set into either format mentioned above. I apologize for any inconvenience

Code:

I. Import Libraries

In [1]:
```
### If missing any library, please uncomment the repective line below and pip install
#!pip install tensorflow --upgrade
#!pip install h5py
#!pip install numpy --upgrade
#!pip install pandas
#!pip install dask --upgrade

## Taken from https://pypi.org/project/pygeohash/
## Using this instead of the python-geohash by hiwi due to better documentation

#!pip install pygeohash
```

In [2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
import pygeohash as pgh
from tqdm._tqdm_notebook import tqdm_notebook
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error,r2_score,mean_absolute_error
import random
import math
import pickle
tqdm_notebook.pandas()
%matplotlib inline
```

```
C:\Users\Coddy\Anaconda3\lib\site-packages\h5py\__init__.py:34: FutureWarning: Conversion of the second argument of issubdtype from `float` to `np.floating` is deprecated. In future, it will be treated as `np.float64 == np.dtype(float).
type`.
  from ._conv import register_converters as _register_converters
```

II. Load Model

In [3]:
```
parameters,train_acc,dev_acc,train,k = pickle.load(open("training_parameters.pkl",'rb'))
```

III. Load Test Data and preparation

Here I use a sample 10% sample test set for debugging and sanity check.

Please replace the file path to actual test.csv 's path

The test set should be in the same format as the training data, i.e 4 columns of: [geohash6 , day , timestamp , demand]

In [4]:
```
df = pd.read_csv("sample_test.csv")
```

Data processing to match the input features

In [5]:

```
def string_to_time (string):
    x = string.split(":")
    timing = int(x[0]) * 60 + int(x[1])
    return timing/15
print("Converting timestamp and day to a single number")
df['time_stamp'] = df['timestamp'].progress_apply(string_to_time)
df['time_stamp'] = df['time_stamp'] + (df['day'] - 1)*96

de = df.groupby(['geohash6']).count()
hash_list = de.index.values

df = pd.pivot_table(df, values='demand', index=['time_stamp'],columns=['geohash6'])
df = df.fillna(0)
df = df.reindex(range(int(df.index.values[-1])), fill_value=0)

X = pd.DataFrame({"index":list(np.core.defchararray.add("T-",np.arange(1248,0,-1).astype("str")))+\
                       ["T","lat","lon"]})
Y = pd.DataFrame({"index":["T+1","T+2","T+3","T+4","T+5"]})

k=0
print("Converting to each row to record 13 days prior data as well as from T to T+5, latitude and longitude\nThis will take a while")
for geohash in tqdm_notebook(hash_list):
    for i in range(1248,df.shape[0]-5):
        if df[geohash].values[i] > 0:
            try:
                k+=1
                X[str(k)] = list(df[geohash].values[i-1248:i+1])+[pgh.decode(geohash)[0],pgh.decode(geohash)[1]]
            except:
                k+=1
                print(df[geohash].values[i])
                print(list(df[geohash].values[i-1248:i+6])+[geohash])
            try:
                Y[str(k)] = list(df[geohash].values[i+1:i+6])
            except:
                Y[str(k)] = [0,0,0,0,0]

X=np.array(X.T.drop(['index'])).astype("float")
Y=np.array(Y.T.drop(['index'])).astype("float")
```

```
Converting timestamp and day to a single number

Converting to each row to record 13 days prior data as well as from T to T+5, latitude and longitude
This will take a while
```

IV. Making Predictions

```
In [6]: def forward_propagation(X, parameters):
            '''
            This function obtain coefficient of various parameters and use them to predict a final cost(Z3)
            This process consists of  a linear function of X @ W1 + b1, @ being matrix multiplication,
                followed by a retilinear activation function
            '''
            W1 = parameters['W1']
            b1 = parameters['b1']
            W2 = parameters['W2']
            b2 = parameters['b2']
            W3 = parameters['W3']
            b3 = parameters['b3']

            Z1 = tf.add(tf.matmul(X,W1), b1)
            A1 = tf.nn.relu(Z1)
            Z2 = tf.add(tf.matmul(A1,W2), b2)
            A2 = tf.nn.relu(Z2)
            Z3 = tf.add(tf.matmul(A2,W3), b3)

            return Z3

        def predict(X, parameters):

            W1 = tf.convert_to_tensor(parameters["W1"])
            b1 = tf.convert_to_tensor(parameters["b1"])
            W2 = tf.convert_to_tensor(parameters["W2"])
            b2 = tf.convert_to_tensor(parameters["b2"])
            W3 = tf.convert_to_tensor(parameters["W3"])
            b3 = tf.convert_to_tensor(parameters["b3"])

            params = {"W1": W1,
                      "b1": b1,
                      "W2": W2,
                      "b2": b2,
                      "W3": W3,
                      "b3": b3}
            try:
                x = tf.placeholder("float", [X.shape[0],X.shape[1]])
            except:
                x = tf.placeholder("float", [1,X.shape[0]])
            z3 = forward_propagation(x, params)

            sess = tf.Session()
            prediction = sess.run(z3, feed_dict = {x: X})

            return prediction
```

**Using the above functions to predict the values of T+1 to T+5 for each row
by using 13 days (1248 time stamps) of demand data, demand data at time = T, latitude and longitude.**

```
In [7]: prediction = predict(X, parameters)
```

```
In [8]: actual = Y
```

V. Results

Output the mean square error

```
In [15]: results = mean_squared_error(actual,prediction)
         r = r2_score(actual,prediction)
         print("The mean squared error is ",results)

         The mean squared error is  0.0002942728594874626
```

```
In [16]: prediction

Out[16]: array([[0.00590695, 0.00590695, 0.00590695, 0.00590695, 0.00590695],
                [0.00633453, 0.00633483, 0.00633505, 0.00633342, 0.00633177],
                [0.00669677, 0.00669732, 0.00669772, 0.00669471, 0.00669167],
                ...,
                [0.01317704, 0.01318211, 0.01318577, 0.01315806, 0.01313012],
                [0.01380702, 0.01381253, 0.01381651, 0.01378639, 0.01375604],
                [0.01245268, 0.01245725, 0.01246054, 0.01243559, 0.01241044]],
               dtype=float32)
```

```
In [17]: sum(actual)

Out[17]: array([5.24399479, 3.58452554, 3.04824917, 3.02181072, 2.84064439])
```

```
In [18]: sum(prediction)
```

```
Out[18]: array([4.3983965, 4.3995695, 4.400417 , 4.394006 , 4.387548 ],
               dtype=float32)
```