Time series forecasting of positions of basket ball players

I have interpretted this as a state space problem hence not tried out Vector Autoregression (VAR) method (although, it would be among my next steps to try out.) Hence, the options I had were Kalman filter, RNN, particle filter, extended Kalman filter, generative RNNs or Variational Autoencoder. Since, this is a complex system, the underlying dynamical process can't be linear (and known before hand say for example Newtonian mechanics), hence there is no point trying out Kalman filter. Witnessing the tremendous success of RNN in object tracking and time series, I decided to implement a simple architecture of RNN to predict the next time step given past (25 and 100) time steps. Future time steps can be further computed by applying the learnt model from past known steps along with predicted previous step(s). Following is my implementation:

single time series

I have experimented with only one time series '2019040102_nba-bos_TRACKING.csv' in a way that player ID or jersey number and team ID are not a part of the training set and the information is implicit from the data structure used as training and test data. This gives rise to 3 features (x, y, z) and hence 11*3 states at each time step of the time series.

Understanding dataset

In [1]:

```
import pandas as pd
from pandas.plotting import autocorrelation_plot
import matplotlib.pyplot as plt
import os
import keras
```

```
C:\Users\verma_k\anaconda3\lib\site-packages\numpy\_distributor_init.py:30: UserWarn
ing: loaded more than 1 DLL from .libs:
C:\Users\verma_k\anaconda3\lib\site-packages\numpy\.libs\libopenblas.GK7GX5KEQ4F6UYO
3P26ULGBQYHGQO7J4.gfortran-win_amd64.dll
C:\Users\verma_k\anaconda3\lib\site-packages\numpy\.libs\libopenblas.wcdjnk7yvmpzq2m
e2zzhjjrj3jikndb7.gfortran-win_amd64.dll
  warnings.warn("loaded more than 1 DLL from .libs:"
```

In [2]:

```
os.chdir('C:/kinexon_task')
nba_bos = pd.read_csv('2019040102_nba-bos_TRACKING.csv')
nba_bos.head(11)
```

Out[2]:

	ts	player_id	jersey_nr	team	x	y	z
0	40:56.2	986b713a-b20b-4eb0-919e-c859d0508af7	0	team_0	0.182882	2.953548	0.000000
1	40:56.2	ff4221b5-89ef-11e6-b799-a45e60e298d3	11	team_0	6.647769	-0.234699	0.000000
2	40:56.2	ff411907-89ef-11e6-9c68-a45e60e298d3	36	team_0	0.438917	-2.990124	0.000000
3	40:56.2	ff427c2e-89ef-11e6-8d2a-a45e60e298d3	42	team_0	3.313216	-0.268227	0.000000
4	40:56.2	ff413e73-89ef-11e6-a5df-a45e60e298d3	46	team_0	0.143258	0.082297	0.000000
5	40:56.2	ff4024b0-89ef-11e6-a1ae-a45e60e298d3	11	team_1	-0.448061	-2.962692	0.000000

	ts	player_id	jersey_nr	team	x	y	z
6	40:56.2	61fc6b15-5ce2-4613-a67b-f15bada3aa57	13	team_1	-0.579127	-0.271275	0.000000
7	40:56.2	0511e7cf-e543-4547-aa9d-1c80e0ec3ad8	5	team_1	-0.871739	2.950500	0.000000
8	40:56.2	ff41920c-89ef-11e6-b546-a45e60e298d3	7	team_1	-7.135455	-0.091441	0.000000
9	40:56.2	ff426326-89ef-11e6-a607-a45e60e298d3	9	team_1	-4.572056	1.874543	0.000000
10	40:56.2	ball	ball	team_ball	-0.012192	-0.368812	1.868447

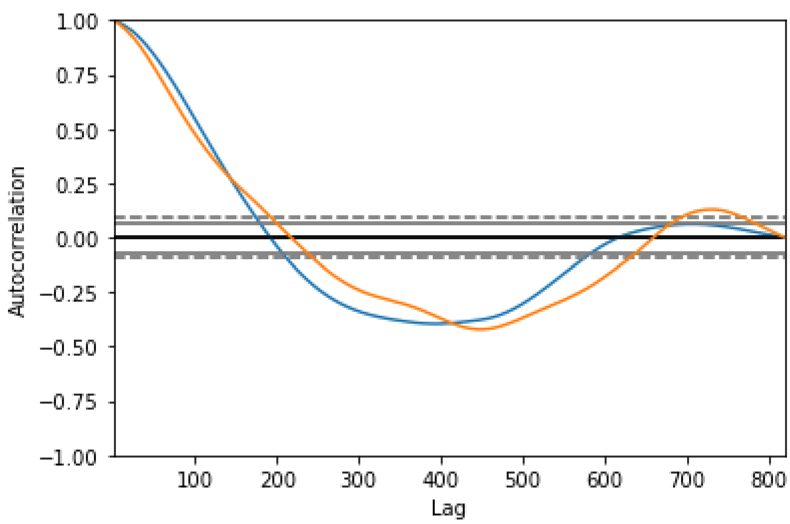
In [3]:

```
time_steps = nba_bos['ts']
player_id = nba_bos['player_id']
jersey_nr = nba_bos['jersey_nr']
team = nba_bos['team']
x = nba_bos['x']
y= nba_bos['y']
z = nba_bos['z']
```

From the following autocorrelation graph, one can interpret that the number of past time steps to forecast next time steps should be less than around 200. Hence, I started with 100.

In [6]:

```
autocorrelation_plot(x[0:9001:11])
autocorrelation_plot(x[1:9001:11])
plt.show()
```



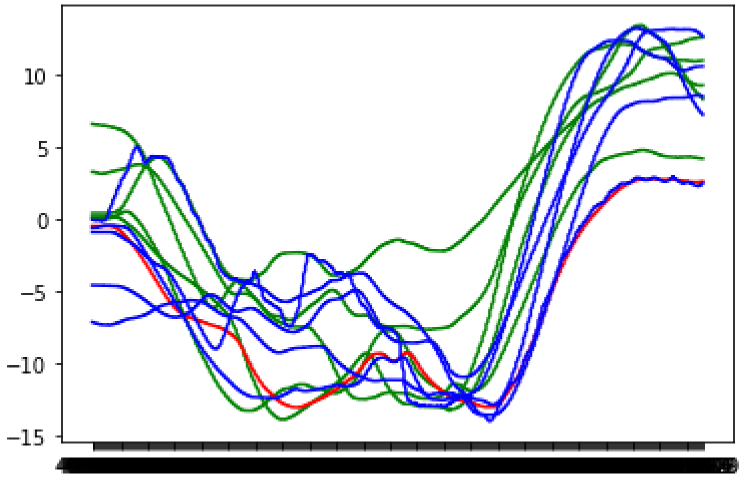
From the following plots, I can assume that the time series I am dealing with is stationary ie., shows no significant trends or seasonality. Hence, I proceed with the data as it is without any preprocessing steps like detrending or deseasonalising.

In [7]:

```
plt.plot(time_steps[0:9000:11], x[0:9001:11], color='green')
plt.plot(time_steps[1:9000:11], x[1:9000:11], color='green')
plt.plot(time_steps[2:9000:11], x[2:9000:11], color='green')
plt.plot(time_steps[3:9000:11], x[3:9000:11], color='green')
plt.plot(time_steps[4:9000:11], x[4:9000:11], color='green')
plt.plot(time_steps[5:9000:11], x[5:9000:11], color='red')
plt.plot(time_steps[6:9000:11], x[6:9001:11], color='blue')
plt.plot(time_steps[7:9000:11], x[7:9000:11], color='blue')
```

```
plt.plot(time_steps[8:9000:11], x[8:9000:11], color='blue')
plt.plot(time_steps[9:9000:11], x[9:9000:11], color='blue')
plt.plot(time_steps[10:9000:11], x[10:9000:11], color='blue')
```

Out[7]: [<matplotlib.lines.Line2D at 0x2acc3c36df0>]



Implementation

The Jupyter file contains an interactive implementation. If you like, you can run the python script that I have sent, in terminal. Following is a sample command:
python3 training_and_evaluation.py -episodes 50 -num_past_steps 25 -batch_size 64 -begin 0 -end 1000 -load_model 1

I would recommend to load the pre-trained models, one corresponding to number of past time steps 100 and the other one 25.

In [4]:

```
import numpy as np
import os
import pandas as pd
from pandas.plotting import autocorrelation_plot
import matplotlib.pyplot as plt

from keras.utils import np_utils
from keras.models import Model
from keras.layers import LSTM, Activation, Dense, Dropout, Input, Embedding, BatchNo
from tensorflow.python.keras.callbacks import History
```

In [7]:

```
#Normalization
#x = (x-np.mean(x))/np.std(x)
#y = (y-np.mean(y))/np.std(y)
#z = (z-np.mean(z))/np.std(z)
```

In [9]:

```
num_time_steps = time_steps.shape[0]
data = np.concatenate((x,y,z))
data = data.reshape(3, num_time_steps)
player=[]
for i in range (11):
    player1 = data[:, i:num_time_steps:11]
    player.append(player1)
```

In [10]:

```
data =  np.array(player)
```

In [11]:
```
data.shape
```

Out[11]:
```
(11, 3, 95325)
```

In [12]:
```
data=data.reshape(33, 95325)
```

In [12]:
```
#in my previous implementation I used reshape instead of transpose and that created
def create_dataset(num_past_steps, data):
    x = []
    y = []
    for i in range(95224):                          #num_time_steps/11 = 95325
        #print(i)
        p = np.matrix.transpose(data[:,i:i+num_past_steps])    #training data
        x.append(p)
        q = np.matrix.transpose(data[:,i+num_past_steps])    #test data
        y.append(q)

    return np.array(x), np.array(y)
```

In [51]:
```
data_X, label = create_dataset(25, data)
```

In [52]:
```
train_x, train_label = data_X[0:95000][:][:], label[0:95000][:][:]
```

In [53]:
```
test_x, test_label = data_X[95000:][:][:], label[95000:][:][:]
```

In [56]:
```
test_label.shape
```

Out[56]:
```
(224, 33)
```

In [60]:
```
train_label=train_label.reshape(95000,1,33)
test_label=test_label.reshape(224,1,33)
```

Training

In [44]:
```
def RNN(past_time_steps):
    inputs = Input(name='inputs',shape=(past_time_steps, 33))
    #layer = BatchNormalization()(inputs)
    layer = LSTM(64, return_sequences=True)(inputs)
    layer = Activation('relu')(layer)
    layer = Dropout(0.2)(layer)
    #layer = LSTM(32, return_sequences=False)(layer)
    layer = Dense(128,name='FC2')(layer)
    layer = Activation('relu')(layer)


    layer = Dense(33,name='out_layer')(layer)
    model = Model(inputs=inputs,outputs=layer)
    return model

#Compile the model
model_25_ = RNN(25)   #or 100
model_25_.summary(25) #or 100
```

```
model_25_.compile(loss='mse',optimizer='adam',metrics=['accuracy']) #or 100

#Fit on training data
#model1.fit(trainx_array_new,label_array_new,batch_size=64,epochs=10, validation_spl
```

```
Model: "model_5"
_________________________
Layer (typ Output Sh Para
=========================
inputs (In [(None, 2 0
_________________________
lstm_5 (LS (None, 25 2508
_________________________
activation (None, 25 0
_________________________
dropout_1  (None, 25 0
_________________________
FC2 (Dense (None, 25 8320
_________________________
activation (None, 25 0
_________________________
out_layer  (None, 25 4257
=========================
Total params: 37,665
Trainable params: 37,665
Non-trainable params: 0
_________________________
```

In [45]:

```
model_25_.fit(train_x, train_label,batch_size=64,epochs=100, validation_split=0.2, s
```

```
Epoch 1/100
1188/1188 [==============================] - 36s 24ms/step - loss: 9.4564 - accurac
y: 0.2410 - val_loss: 3.0982 - val_accuracy: 0.5603
Epoch 2/100
1188/1188 [==============================] - 33s 28ms/step - loss: 3.0189 - accurac
y: 0.4494 - val_loss: 2.9364 - val_accuracy: 0.5545
Epoch 3/100
1188/1188 [==============================] - 30s 26ms/step - loss: 2.6847 - accurac
y: 0.4675 - val_loss: 2.9820 - val_accuracy: 0.5587
Epoch 4/100
1188/1188 [==============================] - 32s 27ms/step - loss: 2.4826 - accurac
y: 0.4777 - val_loss: 2.8698 - val_accuracy: 0.5666
Epoch 5/100
1188/1188 [==============================] - 32s 27ms/step - loss: 2.3076 - accurac
y: 0.4877 - val_loss: 2.9338 - val_accuracy: 0.5507
Epoch 6/100
1188/1188 [==============================] - 36s 30ms/step - loss: 2.2094 - accurac
y: 0.4958 - val_loss: 2.9588 - val_accuracy: 0.5322
Epoch 7/100
1188/1188 [==============================] - 33s 28ms/step - loss: 2.1440 - accurac
y: 0.5002 - val_loss: 3.0362 - val_accuracy: 0.5451
Epoch 8/100
1188/1188 [==============================] - 33s 28ms/step - loss: 2.0792 - accurac
y: 0.5052 - val_loss: 3.0341 - val_accuracy: 0.5330
Epoch 9/100
1188/1188 [==============================] - 33s 28ms/step - loss: 2.0157 - accurac
y: 0.5081 - val_loss: 3.1002 - val_accuracy: 0.5184
Epoch 10/100
1188/1188 [==============================] - 36s 30ms/step - loss: 1.9770 - accurac
y: 0.5130 - val_loss: 3.0878 - val_accuracy: 0.5247
Epoch 11/100
1188/1188 [==============================] - 33s 28ms/step - loss: 1.9191 - accurac
y: 0.5169 - val_loss: 3.1539 - val_accuracy: 0.5019
Epoch 12/100
1188/1188 [==============================] - 33s 28ms/step - loss: 1.9176 - accurac
y: 0.5171 - val_loss: 3.1709 - val_accuracy: 0.5261
Epoch 13/100
1188/1188 [==============================] - 36s 30ms/step - loss: 1.8744 - accurac
```

```
y: 0.5201 - val_loss: 3.2502 - val_accuracy: 0.5224
Epoch 14/100
1188/1188 [==============================] - 34s 28ms/step - loss: 1.8488 - accurac
y: 0.5233 - val_loss: 3.3340 - val_accuracy: 0.5071
Epoch 15/100
1188/1188 [==============================] - 33s 28ms/step - loss: 1.8272 - accurac
y: 0.5252 - val_loss: 3.3380 - val_accuracy: 0.5085
Epoch 16/100
1188/1188 [==============================] - 34s 29ms/step - loss: 1.8045 - accurac
y: 0.5266 - val_loss: 3.3752 - val_accuracy: 0.5066
Epoch 17/100
1188/1188 [==============================] - 35s 30ms/step - loss: 1.7906 - accurac
y: 0.5289 - val_loss: 3.4727 - val_accuracy: 0.5040
Epoch 18/100
1188/1188 [==============================] - 33s 28ms/step - loss: 1.7653 - accurac
y: 0.5298 - val_loss: 3.3544 - val_accuracy: 0.4875
Epoch 19/100
1188/1188 [==============================] - 35s 29ms/step - loss: 1.7487 - accurac
y: 0.5316 - val_loss: 3.4259 - val_accuracy: 0.4955
Epoch 20/100
1188/1188 [==============================] - 36s 30ms/step - loss: 1.7221 - accurac
y: 0.5319 - val_loss: 3.4890 - val_accuracy: 0.4855
Epoch 21/100
1188/1188 [==============================] - 34s 29ms/step - loss: 1.7212 - accurac
y: 0.5353 - val_loss: 3.6947 - val_accuracy: 0.5017
Epoch 22/100
1188/1188 [==============================] - 34s 28ms/step - loss: 1.7063 - accurac
y: 0.5351 - val_loss: 3.5286 - val_accuracy: 0.4794
Epoch 23/100
1188/1188 [==============================] - 34s 28ms/step - loss: 1.6982 - accurac
y: 0.5354 - val_loss: 3.6237 - val_accuracy: 0.4884
Epoch 24/100
1188/1188 [==============================] - 39s 33ms/step - loss: 1.6777 - accurac
y: 0.5371 - val_loss: 3.7282 - val_accuracy: 0.4895
Epoch 25/100
1188/1188 [==============================] - 38s 32ms/step - loss: 1.6597 - accurac
y: 0.5391 - val_loss: 3.6482 - val_accuracy: 0.4832
Epoch 26/100
1188/1188 [==============================] - 35s 30ms/step - loss: 1.6600 - accurac
y: 0.5372 - val_loss: 3.5184 - val_accuracy: 0.4991
Epoch 27/100
1188/1188 [==============================] - 58s 49ms/step - loss: 1.6488 - accurac
y: 0.5401 - val_loss: 3.4778 - val_accuracy: 0.4811
Epoch 28/100
1188/1188 [==============================] - 32s 27ms/step - loss: 1.6387 - accurac
y: 0.5395 - val_loss: 3.6605 - val_accuracy: 0.4891
Epoch 29/100
1188/1188 [==============================] - 30s 25ms/step - loss: 1.6401 - accurac
y: 0.5415 - val_loss: 3.6223 - val_accuracy: 0.4953
Epoch 30/100
1188/1188 [==============================] - 31s 26ms/step - loss: 1.6100 - accurac
y: 0.5437 - val_loss: 3.5956 - val_accuracy: 0.4848
Epoch 31/100
1188/1188 [==============================] - 30s 25ms/step - loss: 1.6038 - accurac
y: 0.5437 - val_loss: 3.6245 - val_accuracy: 0.4885
Epoch 32/100
1188/1188 [==============================] - 31s 26ms/step - loss: 1.6026 - accurac
y: 0.5456 - val_loss: 3.8042 - val_accuracy: 0.4851
Epoch 33/100
1188/1188 [==============================] - 32s 27ms/step - loss: 1.5994 - accurac
y: 0.5447 - val_loss: 3.7163 - val_accuracy: 0.4916
Epoch 34/100
1188/1188 [==============================] - 40s 33ms/step - loss: 1.5717 - accurac
y: 0.5468 - val_loss: 3.6695 - val_accuracy: 0.4831
Epoch 35/100
1188/1188 [==============================] - 31s 26ms/step - loss: 1.5740 - accurac
y: 0.5468 - val_loss: 3.6818 - val_accuracy: 0.4827s: 1.5740 - ac
Epoch 36/100
1188/1188 [==============================] - 36s 30ms/step - loss: 1.5567 - accurac
```

```
y: 0.5473 - val_loss: 3.6870 - val_accuracy: 0.4903
Epoch 37/100
1188/1188 [==============================] - 33s 28ms/step - loss: 1.5814 - accurac
y: 0.5464 - val_loss: 3.6975 - val_accuracy: 0.4878
Epoch 38/100
1188/1188 [==============================] - 37s 31ms/step - loss: 1.5472 - accurac
y: 0.5477 - val_loss: 3.7310 - val_accuracy: 0.4725
Epoch 39/100
1188/1188 [==============================] - 31s 26ms/step - loss: 1.5549 - accurac
y: 0.5464 - val_loss: 3.6649 - val_accuracy: 0.4862
Epoch 40/100
1188/1188 [==============================] - 35s 29ms/step - loss: 1.5438 - accurac
y: 0.5495 - val_loss: 3.6745 - val_accuracy: 0.4873
Epoch 41/100
1188/1188 [==============================] - 39s 33ms/step - loss: 1.5406 - accurac
y: 0.5478 - val_loss: 3.7103 - val_accuracy: 0.5024
Epoch 42/100
1188/1188 [==============================] - 33s 28ms/step - loss: 1.5288 - accurac
y: 0.5501 - val_loss: 3.6992 - val_accuracy: 0.4912
Epoch 43/100
1188/1188 [==============================] - 36s 30ms/step - loss: 1.5226 - accurac
y: 0.5506 - val_loss: 3.6695 - val_accuracy: 0.4897
Epoch 44/100
1188/1188 [==============================] - 43s 36ms/step - loss: 1.5234 - accurac
y: 0.5517 - val_loss: 3.6613 - val_accuracy: 0.4991
Epoch 45/100
1188/1188 [==============================] - 38s 32ms/step - loss: 1.5168 - accurac
y: 0.5533 - val_loss: 3.6994 - val_accuracy: 0.4950
Epoch 46/100
1188/1188 [==============================] - 41s 35ms/step - loss: 1.5091 - accurac
y: 0.5530 - val_loss: 3.6555 - val_accuracy: 0.4910
Epoch 47/100
1188/1188 [==============================] - 34s 28ms/step - loss: 1.5044 - accurac
y: 0.5533 - val_loss: 3.6537 - val_accuracy: 0.4948
Epoch 48/100
1188/1188 [==============================] - 33s 27ms/step - loss: 1.5055 - accurac
y: 0.5554 - val_loss: 3.6330 - val_accuracy: 0.4856
Epoch 49/100
1188/1188 [==============================] - 32s 27ms/step - loss: 1.4810 - accurac
y: 0.5565 - val_loss: 3.6557 - val_accuracy: 0.4918
Epoch 50/100
1188/1188 [==============================] - 31s 26ms/step - loss: 1.4872 - accurac
y: 0.5549 - val_loss: 3.6496 - val_accuracy: 0.4893
Epoch 51/100
1188/1188 [==============================] - 34s 29ms/step - loss: 1.4965 - accurac
y: 0.5537 - val_loss: 3.6375 - val_accuracy: 0.4832
Epoch 52/100
1188/1188 [==============================] - 32s 27ms/step - loss: 1.4776 - accurac
y: 0.5591 - val_loss: 3.6736 - val_accuracy: 0.4879
Epoch 53/100
1188/1188 [==============================] - 32s 27ms/step - loss: 1.4767 - accurac
y: 0.5594 - val_loss: 3.6988 - val_accuracy: 0.4938
Epoch 54/100
1188/1188 [==============================] - 33s 28ms/step - loss: 1.4718 - accurac
y: 0.5602 - val_loss: 3.6602 - val_accuracy: 0.4916
Epoch 55/100
1188/1188 [==============================] - 31s 26ms/step - loss: 1.4656 - accurac
y: 0.5603 - val_loss: 3.7002 - val_accuracy: 0.4849
Epoch 56/100
1188/1188 [==============================] - 32s 27ms/step - loss: 1.4548 - accurac
y: 0.5597 - val_loss: 3.6292 - val_accuracy: 0.4943
Epoch 57/100
1188/1188 [==============================] - 32s 27ms/step - loss: 1.4558 - accurac
y: 0.5606 - val_loss: 3.6778 - val_accuracy: 0.5030
Epoch 58/100
1188/1188 [==============================] - 32s 27ms/step - loss: 1.4596 - accurac
y: 0.5611 - val_loss: 3.6834 - val_accuracy: 0.4938
Epoch 59/100
1188/1188 [==============================] - 34s 29ms/step - loss: 1.4483 - accurac
```

```
y: 0.5618 - val_loss: 3.6913 - val_accuracy: 0.4931
Epoch 60/100
1188/1188 [==============================] - 32s 27ms/step - loss: 1.4615 - accurac
y: 0.5621 - val_loss: 3.6860 - val_accuracy: 0.4942
Epoch 61/100
1188/1188 [==============================] - 34s 29ms/step - loss: 1.4345 - accurac
y: 0.5638 - val_loss: 3.6660 - val_accuracy: 0.4930
Epoch 62/100
1188/1188 [==============================] - 33s 28ms/step - loss: 1.4366 - accurac
y: 0.5617 - val_loss: 3.7378 - val_accuracy: 0.4860
Epoch 63/100
1188/1188 [==============================] - 33s 27ms/step - loss: 1.4183 - accurac
y: 0.5638 - val_loss: 3.7161 - val_accuracy: 0.4854
Epoch 64/100
1188/1188 [==============================] - 34s 28ms/step - loss: 1.4331 - accurac
y: 0.5637 - val_loss: 3.7203 - val_accuracy: 0.4969
Epoch 65/100
1188/1188 [==============================] - 39s 33ms/step - loss: 1.4170 - accurac
y: 0.5646 - val_loss: 3.8863 - val_accuracy: 0.5020
Epoch 66/100
1188/1188 [==============================] - 35s 30ms/step - loss: 1.4471 - accurac
y: 0.5619 - val_loss: 3.7467 - val_accuracy: 0.4980
Epoch 67/100
1188/1188 [==============================] - 37s 31ms/step - loss: 1.4227 - accurac
y: 0.5644 - val_loss: 3.8000 - val_accuracy: 0.5025
Epoch 68/100
1188/1188 [==============================] - 33s 28ms/step - loss: 1.4146 - accurac
y: 0.5654 - val_loss: 3.7707 - val_accuracy: 0.5037
Epoch 69/100
1188/1188 [==============================] - 36s 30ms/step - loss: 1.4156 - accurac
y: 0.5667 - val_loss: 3.7729 - val_accuracy: 0.4913
Epoch 70/100
1188/1188 [==============================] - 35s 29ms/step - loss: 1.4102 - accurac
y: 0.5667 - val_loss: 3.7685 - val_accuracy: 0.5022
Epoch 71/100
1188/1188 [==============================] - 36s 30ms/step - loss: 1.4108 - accurac
y: 0.5645 - val_loss: 3.7973 - val_accuracy: 0.5060
Epoch 72/100
1188/1188 [==============================] - 39s 33ms/step - loss: 1.3969 - accurac
y: 0.5651 - val_loss: 3.8227 - val_accuracy: 0.4984
Epoch 73/100
1188/1188 [==============================] - 37s 31ms/step - loss: 1.4002 - accurac
y: 0.5655 - val_loss: 3.7701 - val_accuracy: 0.5146
Epoch 74/100
1188/1188 [==============================] - 36s 30ms/step - loss: 1.4171 - accurac
y: 0.5670 - val_loss: 3.8178 - val_accuracy: 0.5038
Epoch 75/100
1188/1188 [==============================] - 39s 33ms/step - loss: 1.4100 - accurac
y: 0.5664 - val_loss: 3.8543 - val_accuracy: 0.5061
Epoch 76/100
1188/1188 [==============================] - 35s 30ms/step - loss: 1.4056 - accurac
y: 0.5679 - val_loss: 3.8036 - val_accuracy: 0.5112
Epoch 77/100
1188/1188 [==============================] - 33s 28ms/step - loss: 1.3887 - accurac
y: 0.5666 - val_loss: 3.7686 - val_accuracy: 0.5061
Epoch 78/100
1188/1188 [==============================] - 36s 30ms/step - loss: 1.3918 - accurac
y: 0.5675 - val_loss: 3.8103 - val_accuracy: 0.5016
Epoch 79/100
1188/1188 [==============================] - 39s 33ms/step - loss: 1.3902 - accurac
y: 0.5686 - val_loss: 3.8197 - val_accuracy: 0.5104
Epoch 80/100
1188/1188 [==============================] - 37s 31ms/step - loss: 1.3918 - accurac
y: 0.5667 - val_loss: 3.8420 - val_accuracy: 0.5146
Epoch 81/100
1188/1188 [==============================] - 35s 30ms/step - loss: 1.3760 - accurac
y: 0.5709 - val_loss: 3.8599 - val_accuracy: 0.5208
Epoch 82/100
1188/1188 [==============================] - 33s 28ms/step - loss: 1.3757 - accurac
```

```
y: 0.5711 - val_loss: 3.8374 - val_accuracy: 0.5189
Epoch 83/100
1188/1188 [==============================] - 36s 30ms/step - loss: 1.3771 - accurac
y: 0.5710 - val_loss: 3.8606 - val_accuracy: 0.5190
Epoch 84/100
1188/1188 [==============================] - 36s 31ms/step - loss: 1.3799 - accurac
y: 0.5715 - val_loss: 3.7874 - val_accuracy: 0.5186
Epoch 85/100
1188/1188 [==============================] - 37s 31ms/step - loss: 1.3734 - accurac
y: 0.5721 - val_loss: 3.8072 - val_accuracy: 0.5231curacy: 0.57
Epoch 86/100
1188/1188 [==============================] - 35s 29ms/step - loss: 1.3637 - accurac
y: 0.5714 - val_loss: 3.8323 - val_accuracy: 0.5179
Epoch 87/100
1188/1188 [==============================] - 34s 29ms/step - loss: 1.3737 - accurac
y: 0.5708 - val_loss: 3.8727 - val_accuracy: 0.5122
Epoch 88/100
1188/1188 [==============================] - 35s 29ms/step - loss: 1.3639 - accurac
y: 0.5717 - val_loss: 3.7914 - val_accuracy: 0.5093
Epoch 89/100
1188/1188 [==============================] - 37s 31ms/step - loss: 1.3725 - accurac
y: 0.5707 - val_loss: 3.8461 - val_accuracy: 0.5123
Epoch 90/100
1188/1188 [==============================] - 35s 29ms/step - loss: 1.3633 - accurac
y: 0.5724 - val_loss: 3.8100 - val_accuracy: 0.5207
Epoch 91/100
1188/1188 [==============================] - 35s 30ms/step - loss: 1.3596 - accurac
y: 0.5712 - val_loss: 3.7470 - val_accuracy: 0.5152
Epoch 92/100
1188/1188 [==============================] - 37s 31ms/step - loss: 1.3553 - accurac
y: 0.5715 - val_loss: 3.7624 - val_accuracy: 0.5097
Epoch 93/100
1188/1188 [==============================] - 40s 34ms/step - loss: 1.3650 - accurac
y: 0.5728 - val_loss: 3.9057 - val_accuracy: 0.5108
Epoch 94/100
1188/1188 [==============================] - 30s 25ms/step - loss: 1.3587 - accurac
y: 0.5738 - val_loss: 3.8598 - val_accuracy: 0.5131
Epoch 95/100
1188/1188 [==============================] - 32s 27ms/step - loss: 1.3514 - accurac
y: 0.5723 - val_loss: 3.8271 - val_accuracy: 0.5068
Epoch 96/100
1188/1188 [==============================] - 33s 28ms/step - loss: 1.3492 - accurac
y: 0.5731 - val_loss: 3.8154 - val_accuracy: 0.5084
Epoch 97/100
1188/1188 [==============================] - 36s 30ms/step - loss: 1.3576 - accurac
y: 0.5717 - val_loss: 3.8459 - val_accuracy: 0.5023
Epoch 98/100
1188/1188 [==============================] - 35s 29ms/step - loss: 1.3470 - accurac
y: 0.5728 - val_loss: 3.8606 - val_accuracy: 0.5198
Epoch 99/100
1188/1188 [==============================] - 37s 31ms/step - loss: 1.3457 - accurac
y: 0.5751 - val_loss: 3.8252 - val_accuracy: 0.5046
Epoch 100/100
1188/1188 [==============================] - 35s 30ms/step - loss: 1.3445 - accurac
y: 0.5748 - val_loss: 3.7688 - val_accuracy: 0.5113
```

Out[45]: `<keras.callbacks.History at 0x1977d92a9a0>`

In [61]:

```
#Test result:
model_25_.evaluate(test_x, test_label)
```

```
7/7 [==============================] - 5s 7ms/step - loss: 3.7754 - accuracy: 0.7379
```

Out[61]: `[3.7754178047180176, 0.7378571629524231]`

In [62]:

```
#Data preparation for model_100_
```

In [21]:
```
data_X, label = create_dataset(100, data)
train_x, train_label = data_X[0:95000][:][:], label[0:95000][:][:]
test_x, test_label = data_X[95000:][:][:], label[95000:][:][:]
train_label=train_label.reshape(95000,1,33)
```

In [26]:
```
test_label=test_label.reshape(224,1,33)
```

In [20]:
```
test_label=test_label.reshape(224,1,3)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-20-43d8ef69b2c7> in <module>
----> 1 test_label=test_label.reshape(224,1,3)

ValueError: cannot reshape array of size 7392 into shape (224,1,3)
```

In [16]:
```
import keras
model_100_ = keras.models.load_model("my_model100")
```

In [48]:
```
#Compile the model
model_100_ = RNN(100)
model_100_.summary(100)
model_100_.compile(loss='mse',optimizer='adam',metrics=['accuracy'])
model_100_.fit(train_x, train_label,batch_size=64,epochs=100, validation_split=0.2,
```

```
Model: "model_7"
________________________________________________________________________________
________________
Layer (type)                                 Output Shape
Param #
================================================================================
================
inputs (InputLayer)                          [(None, 100, 33)]
0

________________________________________________________________________________
________________
lstm_7 (LSTM)                                (None, 100, 64)
25088

________________________________________________________________________________
________________
activation_10 (Activation)                   (None, 100, 64)
0

________________________________________________________________________________
________________
dropout_3 (Dropout)                          (None, 100, 64)
0

________________________________________________________________________________
________________
FC2 (Dense)                                  (None, 100, 128)
8320

________________________________________________________________________________
________________
activation_11 (Activation)                   (None, 100, 128)
0

________________________________________________________________________________
________________
out_layer (Dense)                            (None, 100, 33)
4257
================================================================================
================
Total params: 37,665
Trainable params: 37,665
```

```
Non-trainable params: 0
__________________________________________________________________________________
__________________
Epoch 1/100
1188/1188 [==============================] - 131s 96ms/step - loss: 12.1563 - accura
cy: 0.1731 - val_loss: 11.3957 - val_accuracy: 0.3487
Epoch 2/100
1188/1188 [==============================] - 115s 97ms/step - loss: 5.5186 - accurac
y: 0.3357 - val_loss: 11.8278 - val_accuracy: 0.3074
Epoch 3/100
1188/1188 [==============================] - 121s 102ms/step - loss: 4.8076 - accura
cy: 0.3621 - val_loss: 12.1690 - val_accuracy: 0.3052
Epoch 4/100
1188/1188 [==============================] - 117s 98ms/step - loss: 4.4408 - accurac
y: 0.3747 - val_loss: 12.9700 - val_accuracy: 0.2880
Epoch 5/100
1188/1188 [==============================] - 121s 102ms/step - loss: 4.1417 - accura
cy: 0.3840 - val_loss: 13.4743 - val_accuracy: 0.2632
Epoch 6/100
1188/1188 [==============================] - 118s 100ms/step - loss: 3.9686 - accura
cy: 0.3931 - val_loss: 13.7336 - val_accuracy: 0.2685
Epoch 7/100
1188/1188 [==============================] - 119s 100ms/step - loss: 3.7665 - accura
cy: 0.3984 - val_loss: 13.3627 - val_accuracy: 0.2608
Epoch 8/100
1188/1188 [==============================] - 125s 105ms/step - loss: 3.7010 - accura
cy: 0.4043 - val_loss: 13.5105 - val_accuracy: 0.2720
Epoch 9/100
1188/1188 [==============================] - 135s 113ms/step - loss: 3.5891 - accura
cy: 0.4096 - val_loss: 14.2213 - val_accuracy: 0.2346
Epoch 10/100
1188/1188 [==============================] - 130s 109ms/step - loss: 3.4924 - accura
cy: 0.4118 - val_loss: 13.8957 - val_accuracy: 0.2374
Epoch 11/100
1188/1188 [==============================] - 122s 103ms/step - loss: 3.4124 - accura
cy: 0.4151 - val_loss: 13.6711 - val_accuracy: 0.2456
Epoch 12/100
1188/1188 [==============================] - 125s 105ms/step - loss: 3.3109 - accura
cy: 0.4197 - val_loss: 13.9854 - val_accuracy: 0.2379
Epoch 13/100
1188/1188 [==============================] - 124s 105ms/step - loss: 3.2600 - accura
cy: 0.4213 - val_loss: 13.9211 - val_accuracy: 0.2379
Epoch 14/100
1188/1188 [==============================] - 125s 105ms/step - loss: 3.2312 - accura
cy: 0.4226 - val_loss: 13.7727 - val_accuracy: 0.2358
Epoch 15/100
1188/1188 [==============================] - 123s 103ms/step - loss: 3.1851 - accura
cy: 0.4267 - val_loss: 14.0479 - val_accuracy: 0.2346
Epoch 16/100
1188/1188 [==============================] - 129s 109ms/step - loss: 3.1439 - accura
cy: 0.4276 - val_loss: 13.7969 - val_accuracy: 0.2413
Epoch 17/100
1188/1188 [==============================] - 120s 101ms/step - loss: 3.1007 - accura
cy: 0.4291 - val_loss: 13.7476 - val_accuracy: 0.2401
Epoch 18/100
1188/1188 [==============================] - 127s 107ms/step - loss: 3.1259 - accura
cy: 0.4280 - val_loss: 14.1215 - val_accuracy: 0.2283
Epoch 19/100
1188/1188 [==============================] - 119s 100ms/step - loss: 3.0431 - accura
cy: 0.4304 - val_loss: 14.1177 - val_accuracy: 0.2309
Epoch 20/100
1188/1188 [==============================] - 121s 102ms/step - loss: 3.0091 - accura
cy: 0.4326 - val_loss: 13.9611 - val_accuracy: 0.2252
Epoch 21/100
1188/1188 [==============================] - 125s 105ms/step - loss: 3.0402 - accura
cy: 0.4333 - val_loss: 14.1823 - val_accuracy: 0.2283
Epoch 22/100
1188/1188 [==============================] - 125s 105ms/step - loss: 2.9907 - accura
cy: 0.4363 - val_loss: 14.6518 - val_accuracy: 0.2352
```

```
Epoch 23/100
1188/1188 [==============================] - 118s 99ms/step - loss: 2.9713 - accurac
y: 0.4332 - val_loss: 14.3322 - val_accuracy: 0.2238
Epoch 24/100
1188/1188 [==============================] - 127s 107ms/step - loss: 2.9178 - accura
cy: 0.4371 - val_loss: 14.5957 - val_accuracy: 0.2184
Epoch 25/100
1188/1188 [==============================] - 123s 104ms/step - loss: 2.8987 - accura
cy: 0.4378 - val_loss: 14.8738 - val_accuracy: 0.2231
Epoch 26/100
1188/1188 [==============================] - 130s 109ms/step - loss: 2.9033 - accura
cy: 0.4397 - val_loss: 14.7893 - val_accuracy: 0.2256
Epoch 27/100
1188/1188 [==============================] - 123s 104ms/step - loss: 2.8921 - accura
cy: 0.4395 - val_loss: 15.1844 - val_accuracy: 0.2135
Epoch 28/100
1188/1188 [==============================] - 129s 109ms/step - loss: 2.9727 - accura
cy: 0.4356 - val_loss: 15.0634 - val_accuracy: 0.2131
Epoch 29/100
1188/1188 [==============================] - 125s 106ms/step - loss: 2.8672 - accura
cy: 0.4397 - val_loss: 15.2148 - val_accuracy: 0.2109
Epoch 30/100
1188/1188 [==============================] - 122s 102ms/step - loss: 2.8185 - accura
cy: 0.4448 - val_loss: 15.2125 - val_accuracy: 0.2062
Epoch 31/100
1188/1188 [==============================] - 127s 107ms/step - loss: 2.8577 - accura
cy: 0.4415 - val_loss: 15.2733 - val_accuracy: 0.2087
Epoch 32/100
1188/1188 [==============================] - 121s 102ms/step - loss: 2.8096 - accura
cy: 0.4439 - val_loss: 15.3282 - val_accuracy: 0.2151
Epoch 33/100
1188/1188 [==============================] - 122s 103ms/step - loss: 2.8480 - accura
cy: 0.4432 - val_loss: 15.3400 - val_accuracy: 0.2146
Epoch 34/100
1188/1188 [==============================] - 123s 104ms/step - loss: 2.7770 - accura
cy: 0.4447 - val_loss: 15.6134 - val_accuracy: 0.2094
Epoch 35/100
1188/1188 [==============================] - 127s 107ms/step - loss: 2.7769 - accura
cy: 0.4464 - val_loss: 15.6778 - val_accuracy: 0.2037
Epoch 36/100
1188/1188 [==============================] - 126s 106ms/step - loss: 2.7833 - accura
cy: 0.4452 - val_loss: 15.0316 - val_accuracy: 0.2233
Epoch 37/100
1188/1188 [==============================] - 122s 102ms/step - loss: 2.8310 - accura
cy: 0.4423 - val_loss: 15.6454 - val_accuracy: 0.2077
Epoch 38/100
 136/1188 [==>...........................] - ETA: 1:46 - loss: 2.9058 - accuracy: 0.
4367
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-48-9fd62560dd82> in <module>
      3 model_100_.summary(100)
      4 model_100_.compile(loss='mse',optimizer='adam',metrics=['accuracy'])
----> 5 model_100_.fit(train_x, train_label,batch_size=64,epochs=100, validation_spl
it=0.2, shuffle='True')

~\AppData\Roaming\Python\Python38\site-packages\keras\engine\training.py in fit(sel
f, x, y, batch_size, epochs, verbose, callbacks, validation_split, validation_data,
 shuffle, class_weight, sample_weight, initial_epoch, steps_per_epoch, validation_st
eps, validation_batch_size, validation_freq, max_queue_size, workers, use_multiproce
ssing)
   1156                 _r=1):
   1157               callbacks.on_train_batch_begin(step)
-> 1158               tmp_logs = self.train_function(iterator)
   1159               if data_handler.should_sync:
   1160                 context.async_wait()

~\AppData\Roaming\Python\Python38\site-packages\tensorflow\python\eager\def_functio
n.py in __call__(self, *args, **kwds)
```

```
    887
    888         with OptionalXlaContext(self._jit_compile):
--> 889           result = self._call(*args, **kwds)
    890
    891         new_tracing_count = self.experimental_get_tracing_count()

~\AppData\Roaming\Python\Python38\site-packages\tensorflow\python\eager\def_functio
n.py in _call(self, *args, **kwds)
    915         # In this case we have created variables on the first call, so we run
 the
    916         # defunned version which is guaranteed to never create variables.
--> 917         return self._stateless_fn(*args, **kwds)  # pylint: disable=not-callab
le
    918       elif self._stateful_fn is not None:
    919         # Release the lock early so that multiple threads can perform the call

~\AppData\Roaming\Python\Python38\site-packages\tensorflow\python\eager\function.py
 in __call__(self, *args, **kwargs)
   3021         (graph_function,
   3022          filtered_flat_args) = self._maybe_define_function(args, kwargs)
-> 3023       return graph_function._call_flat(
   3024           filtered_flat_args, captured_inputs=graph_function.captured_inputs)
# pylint: disable=protected-access
   3025

~\AppData\Roaming\Python\Python38\site-packages\tensorflow\python\eager\function.py
 in _call_flat(self, args, captured_inputs, cancellation_manager)
   1958           and executing_eagerly):
   1959         # No tape is watching; skip to running the function.
-> 1960         return self._build_call_outputs(self._inference_function.call(
   1961             ctx, args, cancellation_manager=cancellation_manager))
   1962       forward_backward = self._select_forward_and_backward_functions(

~\AppData\Roaming\Python\Python38\site-packages\tensorflow\python\eager\function.py
 in call(self, ctx, args, cancellation_manager)
    589         with _InterpolateFunctionError(self):
    590           if cancellation_manager is None:
--> 591             outputs = execute.execute(
    592                 str(self.signature.name),
    593                 num_outputs=self._num_outputs,

~\AppData\Roaming\Python\Python38\site-packages\tensorflow\python\eager\execute.py i
n quick_execute(op_name, num_outputs, inputs, attrs, ctx, name)
     57   try:
     58     ctx.ensure_initialized()
---> 59     tensors = pywrap_tfe.TFE_Py_Execute(ctx._handle, device_name, op_name,
     60                                         inputs, attrs, num_outputs)
     61   except core._NotOkStatusException as e:

KeyboardInterrupt:
```

In [27]:

```python
#Test results:
model_100_.evaluate(test_x, test_label)
```

```
7/7 [==============================] - 9s 23ms/step - loss: 6.7734 - accuracy: 0.120
4
```

Out[27]:

```
[6.773382663726807, 0.12044642865657806]
```

In [50]:

```python
model_100_.save('my_model100')
model_25_.save('my_model25')
```

```
WARNING:absl:Found untraced functions such as lstm_cell_7_layer_call_and_return_cond
itional_losses, lstm_cell_7_layer_call_fn, lstm_cell_7_layer_call_fn, lstm_cell_7_la
yer_call_and_return_conditional_losses, lstm_cell_7_layer_call_and_return_conditiona
l_losses while saving (showing 5 of 5). These functions will not be directly callabl
e after loading.
```

```
INFO:tensorflow:Assets written to: my_model100\assets
INFO:tensorflow:Assets written to: my_model100\assets
WARNING:absl:Found untraced functions such as lstm_cell_5_layer_call_and_return_cond
itional_losses, lstm_cell_5_layer_call_fn, lstm_cell_5_layer_call_fn, lstm_cell_5_la
yer_call_and_return_conditional_losses, lstm_cell_5_layer_call_and_return_conditiona
l_losses while saving (showing 5 of 5). These functions will not be directly callabl
e after loading.
INFO:tensorflow:Assets written to: my_model25\assets
INFO:tensorflow:Assets written to: my_model25\assets
```

Variance Test

In [5]:

```
nba_uta = pd.read_csv('2019040129_nba-uta_TRACKING.csv')
```

In [6]:

```
nba_gsw = pd.read_csv('2019051610_nba-gsw_TRACKING.csv')
```

In [8]:

```
def data_preprocessing(raw_data):

    #team = raw_data['team']
    #data = 0

    #for i in range (len(raw_data['ts'])):
        #if team[i]=='team_ball':
            #team[i]='team_2'

    #team = np.array(team)
    #team_processed = np.zeros([len(team)])
    #for i in range (len(team)):
        #team_processed[i] = int(team[i][5])

    num_time_steps = time_steps.shape[0]
    data = np.concatenate((raw_data['x'],raw_data['y'],raw_data['z']))
    data = data.reshape(3, num_time_steps)
    player=[]
    for i in range (11):
        player1 = data[:, i:num_time_steps:11]
        player.append(player1)

    data =  np.array(player)

    return data
```

In [18]:

```
data = data_preprocessing(nba_gsw)
```

In [19]:

```
data=data.reshape(33,95325)
```

In [25]:

```
x, y = create_dataset(25, data)
```

In [28]:

```
len(y)
```

Out[28]: 95224

In [26]:

```
y=y.reshape(95224,1,33)
```

NBA_UTA

In [44]:
```
model_25_.evaluate(x[0:6000], y[0:6000], batch_size=64)
```

```
94/94 [==============================] - 6s 8ms/step - loss: 3.1988 - accuracy: 0.50
74
```

Out[44]:
```
[3.198753595352173, 0.5073666572570801]
```

In [16]:
```
model_100_.evaluate(x[0:6000], y[0:6000], batch_size=64)
```

```
94/94 [==============================] - 17s 26ms/step - loss: 12.2466 - accuracy:
0.2411
```

Out[16]:
```
[12.246556282043457, 0.24110166728496552]
```

NBA_GSW

In [22]:
```
model_100_.evaluate(x[0:6000], y[0:6000], batch_size=64)
```

```
94/94 [==============================] - 3s 27ms/step - loss: 15.7371 - accuracy: 0.
2476
```

Out[22]:
```
[15.73711109161377, 0.24758833646774292]
```

In [27]:
```
model_25_.evaluate(x[0:6000], y[0:6000], batch_size=64)
```

```
94/94 [==============================] - 2s 8ms/step - loss: 3.1692 - accuracy: 0.43
47
```

Out[27]:
```
[3.1692137718200684, 0.434746652841568]
```

Further steps:

1. First of all, training the same model with around 200 to 400 epochs as there is a clear scope of increasing the training accuracy.
2. Combining all the three time series and evaluating with the same RNN architecture.
3. Architectural tweaks, like batch normalization, adding LSTM layers etc.
4. Predicting more than 1 future time steps either in a regressive way (feeding network outputs to the model and predicting the next step), or by directly making labels 2-dimensional.
5. Model player interacrions like euclidean distances, playing position etc. (giving rise to a graph), and then apply min-cost flow for time series forecasting.
6. Trying out generative RNNs and Variational Autoencoders, to perform uncertainty quantification and generate scenarios that were never seen before, but have the same posterior distribution p(z|x), z:=latent variables describing space dynamics, x:=states. This can also be used for data augmentation.
7. In case the underlying dynamics can also be learnt, then particle filter is also a solution. I am not sure how else to model the underlying dynamics.

In [ ]: