Project Report Datamining and Neurocomputing: Trajectories

Axel De Nardin, Cihan Yatbaz

June 2019

1 Introduction

The goal of the trajectory project was to build a model that, given multiple time series that represented the movement of molecular particles inside a cell, learned to predict the trajectory of said particles.

2 Data description

The data regarding the particles movements consisted in 2751 trajectories for a total of 53896 frame, the relevant attributes of the dataset are the following:

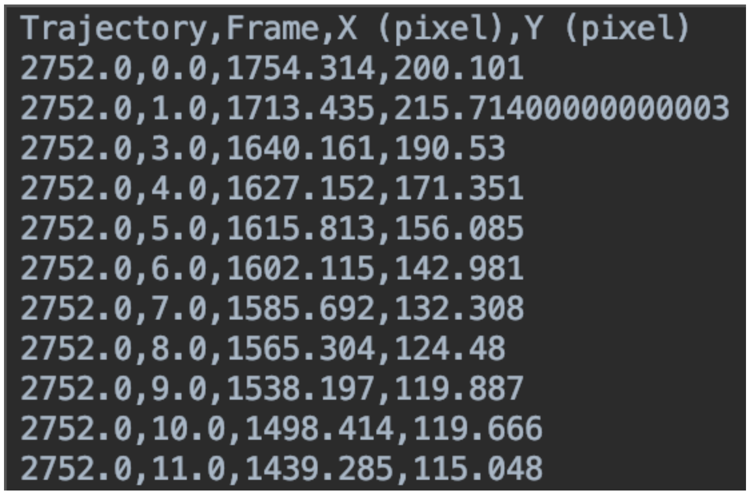
1. A trajectory id
2. The frame at which a certain position of the cell was recorded.
3. The X coordinate of the cell at a certain frame (in pixels).
4. The Y coordinate of the cell at a certain frame (in pixels).

a snippet of the dataset is reported in Fig. 1

3 Data selection and preparation

3.1 Data Preparation: Trajectories stitching

The data preparation for this dataset presented a peculiar problem. In some cases, probably due to an error occurring during the registration parts of the same trajectory has been given different ids. To solve this problem we needed to define a function that recognized the split trajectories and stitched them together. The first indicator that two different ids may refer to the same trajectory was that the last frame of one of the 2 trajectories had, as its first "Frame" value, the last frame value of the other trajectory + 1. Of course this aspect alone wasn't enough to decide to stitch 2 trajectories together, for this reason a similarity measure has been defined to determine if they were similar enough



```
Trajectory,Frame,X (pixel),Y (pixel)
2752.0,0.0,1754.314,200.101
2752.0,1.0,1713.435,215.71400000000003
2752.0,3.0,1640.161,190.53
2752.0,4.0,1627.152,171.351
2752.0,5.0,1615.813,156.085
2752.0,6.0,1602.115,142.981
2752.0,7.0,1585.692,132.308
2752.0,8.0,1565.304,124.48
2752.0,9.0,1538.197,119.887
2752.0,10.0,1498.414,119.666
2752.0,11.0,1439.285,115.048
```

Figure 1: Snippet of the trajectories dataset

to decide to apply the stitching. Given 2 trajectories ”A” and ”B”, for which $first_frame(B) = last_frame(A) + 1$, the aformentioned similarity measure took into consideration 3 elements:

- The vector representing the displacement between the last 2 positions of trajectory A (V1).
- The vector representing the displacement between the first 2 positions of trajectory B (V2).
- The vector representing the displacement between the last position of A and the first position of B (V3).

The idea was to check whether movement from one trajectory to the other was similar enough the last movement of the first trajectory and the first movement of the second trajectory. The similarity measure was then defined as follows:

$$intensity\_change\_1 = \frac{max(\|V1\|, \|V2\|)}{min(\|V1\|, \|V2\|)} \tag{1}$$

$$intensity\_change\_2 = \frac{max(\|V2\|, \|V3\|)}{min(\|V2\|, \|V3\|)} \tag{2}$$

$$cos\_similarity\_1 = cosine\_similarity(V1, V2) \tag{3}$$

$$cos_similarity\_2 = cosine\_similarity(V2, V3) \tag{4}$$

$$final\_similarity\_1 = cos\_similarity\_1/intensity\_change\_1 \tag{5}$$

$$final\_similarity\_2 = cos\_similarity\_2/intensity\_change\_2 \tag{6}$$

In the above equation (1) and (2) represent, respectively , the difference in magnitude between V1 and V2, and the difference in magnitude between V2 and V3, while (3) and (4) represent, respectively, the cosine similarity between V1 and V2, and between V2 and V3. the final similarity measure takes combines these two elements. The idea behind this measure was that, if two trajectories belonged together, then the 2 closest movement to the "connection" and the connection itself should have similar direction (cosine_similarity) and magnitude and, the more one of the 2 differed the more the other should be similar, so that if we have a larger difference in direction we should have a smaller difference in magnitude and viceversa. Finally, to decide if the two trajectories needed to be stitched together a threshold of '0.5' was defined. if both the 'final_similarity' values were above this threshold then the 2 trajectories got stitched together. The choice of the threshold value was defined so that the two extreme for which the trajectories got stitched were the following:

- The cos_similarity was equal to 0.5 (difference in direction between the 2 displacements = 60 degrees) and the intensity_change was 1 (the two displacements had same intensity)
- The cos_similarity was equal to 1 (the two displacements had the same direction) and the intensity_change was 2 (the intensity of the bigger displacement was twice the other one)

After stitching together the original trajectories, the procedure was applied again by trying to stitch the new set of trajectories obtained with the original ones. this process was repeated until no new trajectories emerged and resulted in the creation of 133 new trajectories. Two examples of stitched trajectories are show in Fig.2.

3.2 Data preparation: missing data

By looking at the dataset we noticed that some frames were missing. To solve this problem we introduced the missing frames by interpolating them with the values of the coordinates for the frames immediately before and after them, with respect to the frame number. This operation resulted in the introduction of 1588 missing frames.

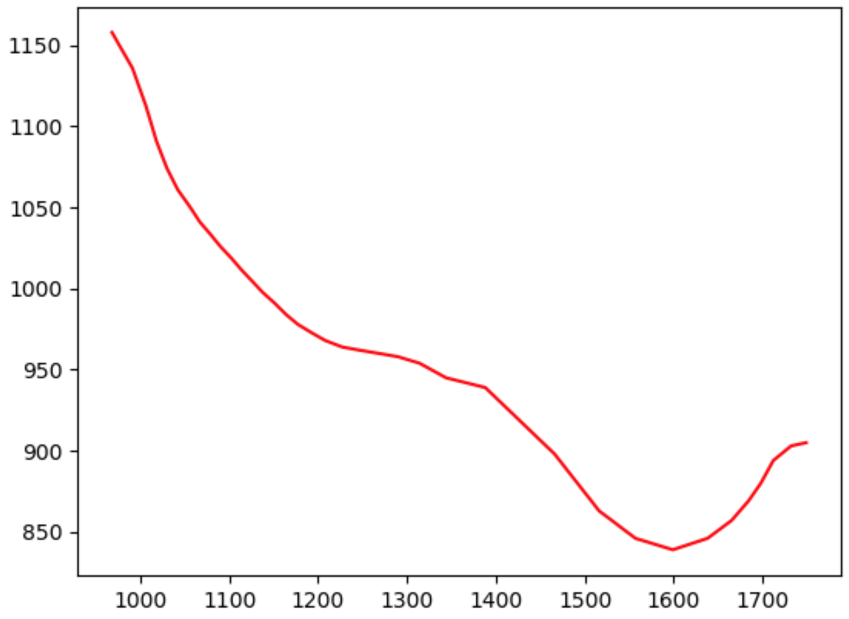
3.3 Data preparation: normalization

Another part of the data preparation regarded the normalization of the coordinate values of the positions of the particles. This operation was done by using the MinMaxScaler provided by the scikit learn library, which simply applies a minmax normalization, mapping all the values in the range (0,1), defined as:

$$x_{new} = \frac{x_{old} - min(x)}{max(x) - min(x)}$$

3.4 Data preparation: Train-test splitting

Finally the data was split into training and test data. Since the dataset represented multiple time series it couldn't simply be split randomly, instead all



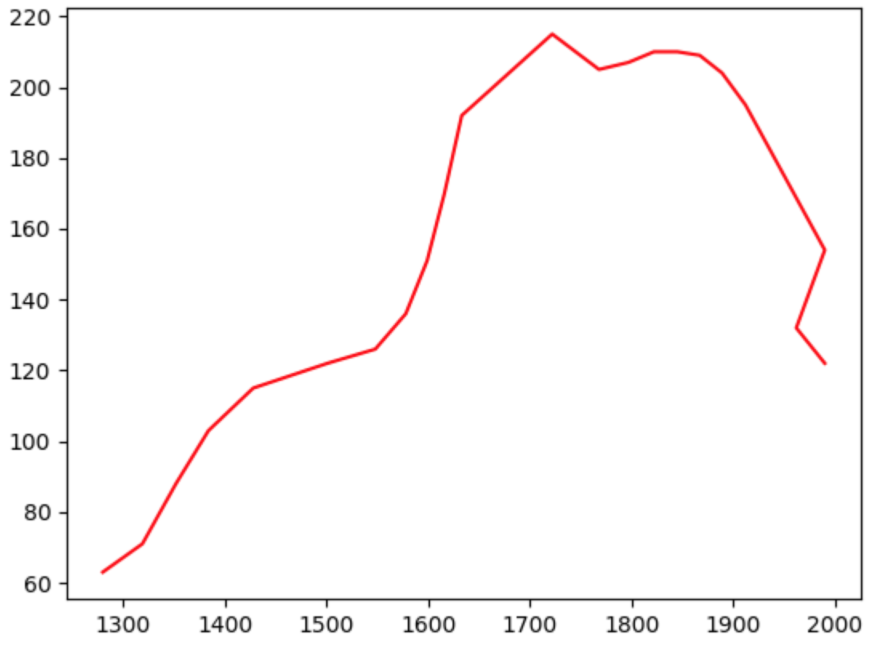


Figure 2: Stitched trajectories examples

the frames belonging to the same trajectory needed to be kept together in the same set. For this reason the approach we adopted was to extract the set of all unique trajectory ids from the dataset and then split it into the training (80% of the total trajectories) and the test sets (the remaining 20% of trajectories). After the 2 sets of ids were obtained we could simply retrieve the corresponding trajectories data for both of them.

3.5 Data selection

The only form of data selection implemented was removing all the trajectories that presented less then a certain threshold of points. where this threshold was determined by a parameter called 'look_back' that defined how many point were considered to predict the next one. If, for example the 'look_back' is equal to 3 then all the trajectories with length lower than 4 would be removed, because to make a prediction would be needed at least 'look_back' coordinates + 1.

4 Problem analysis

The problem presented itself as a typical multiple time series forecasting problem, in which, after training a model on the historical data regarding the time series we want to use it for predicting the behaviour of new time series representing the same type of data. In this specific case the time series represented the movement of molecular particles inside a cell, so we wanted to analyze the behaviour of these movements over time and then try to predict movements of new particles.

5 Mathematical description of proposed models

5.1 Network architecture

The network architecture built to solve the describes problem consists in 2 hidden layer: the first one is a LSTM layer consisting in 4 nodes, which takes as input the data regarding 'look_back' frames at a time while the second one is a Dense layer consisting in 2 nodes. The model summary provided by Keras is reported in Fig.3.

LSTM: LSTM's are particularly indicated when the model we need to build must learn from data sets in which the time component is included, meaning that there is a short or long term correlation between the data units that are fed to the network as a sequence. The reason for which LSTM work well for this type of datasets is that Each node is composed of the following elements:

- **A memory**: where the relevant information for the current sequence of data is stored

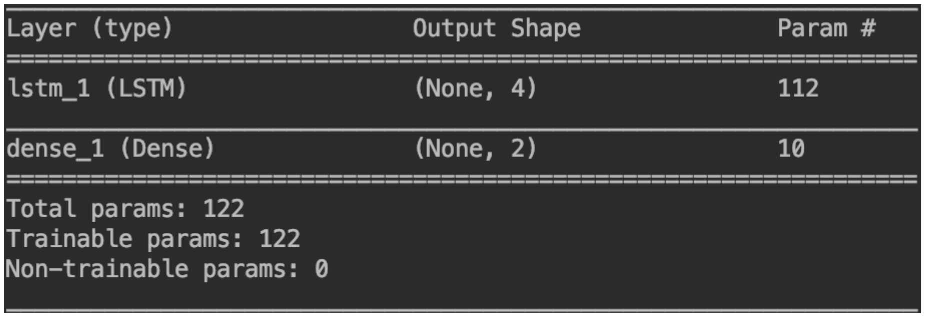

```
Layer (type)                 Output Shape              Param #
=================================================================
lstm_1 (LSTM)                (None, 4)                 112
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 10
=================================================================
Total params: 122
Trainable params: 122
Non-trainable params: 0
_________________________________________________________________
```

Figure 3: Model summary

- **An input gate**: Is used to update the hidden state of the cell. It takes as input the previous hidden state and the current input and pass them as input to a sigmoid function which outputs a value between 0 and 1, where 0 means that the information is not important while 1 means it is and determines which values need to be updated.
- **A state**: At each step the state gets updated in the following way: first it gets multiplied point wise by the forget vector, this operation may result in some values of the state dropping. The output of the previous operation is then added to the output from the input gate, this operation determines which values the network finds relevant and gives the new cell state.
- **An output gate**: The output gate decides what the next hidden state will be. First the current input and the previous hidden state are passed into a sigmoid function. Then the previously modified cell state is passed to a tanh function. Finally the output of the two functions is multiplied to decide which information should be kept in the hidden state which will be carried to the next time step.
- **A forget gate**: which decides what information must be kept into the memory and which must be forgotten. To do so it uses a sigmoid function that takes as input the information about the cell's previous hidden state and the information about the current state. A value of the sigmoid close to 0 means a propension to forget while a value close to 1 means a propension of the cell to keep the information.

Because of their structure the LTSM prevent the vanishing gradient problem, which consist in the gradient vanishing as it back propagates through time. It is a problem because if the gradient becomes too small it doesn't contribute too much to the learning and this affects especially the early layers of a network which tend to forget long term correlations.

Dense Layer: A dense layer is simply a network layer in which all of the neurons receive the input from all the neurons in the previous layer of the network itself. In this case the Dense layer is used as an output layer to match the requested output size of a bi-dimensional coordinate.

Loss function: The loss function used for the network is the mean squared error which a scale based measure that punishes larger errors and should lead to forecasts of the mean.

Optimizer: The optimizer used for the network is "ADAM" which is an adaptive learning rate method that computes individual learning rates for the different parameters instead of using a common one. It also uses the squared gradient to scale the learning rates.

5.2 Training

The training of the model was carried out by taking one trajectory of the training set at a time and splitting into sequences of 3 frames so that the first sequence was composed of the first 3 frames of the trajectory, the second sequence was composed of frames 2,3 and 4 and so on, until the final one which was composed of the frames n-3,n-2 and n-1 (where n was the total number of frames for that trajectory). Each of these sequences was fed to the model and the output compared to the first frame after those contained in the sequence (so if the input was the first sequence the output would be compared to the 4th frame of the trajectory) to compute the MSE that will then be used for the backpropagation step. The model was trained on the data of each trajectory for 50 epochs.

6 Performance and Evaluation

The performance of the model was evaluated by using a rolling window approach in which for each trajectory in the test set the first 3 frame were considered as a baseline to predict the next ones and the final result would then be compared to the real trajectory and the MSE calculated over all the predictions made for the tests set. The model was evaluated both on single step prediction and multistep prediction for steps of size varying from to 2 to 9. The single step prediction consisted in taking the 3 real coordinates to predict the next coordinate of the test trajectory, this process was repeated until all the point had been predicted. For the multistep prediction, on the other end, the real coordinates values were used only for the first three steps, while for the next ones the coordinates previously predicted by the model were used as input for the prediction. As expected the performance of the model decreased quickly as the number of prediction steps increased.

7 Results obtained

The results for the multistep predictions obtained by the model are the following: As is show in the graph in Fig.4 the performance of the model increases rapidly

Steps	Mean Square Error
1-Step	0.0001745
2-Step	0.0007155
3-Step	0.0016397
4-Step	0.0029084
5-step	0.0044803
6-Step	0.0063320
7-Step	0.0084526
8-Step	0.0108246
9-Step	0.0134342

as the number of prediction steps increasing, from 1-step prediction to 9-step the increase of the MSE for the results produced by the model is of 2 orders of magnitude. This is likely due to the fact the the initially small errors commited in the prediction first prediction steps propagates and gets amplified during the later steps predictions. To have a better idea of the meaning of these result is useful to visualize the comparison between the real trajectory and the predicted one for different levels of multi-step prediction. The results for 1,3, 5, 9-step predictions are reported in Fig.5. As we can see, as the number of steps increases the model seems to predict coordinates progressively closer to the previous ones.

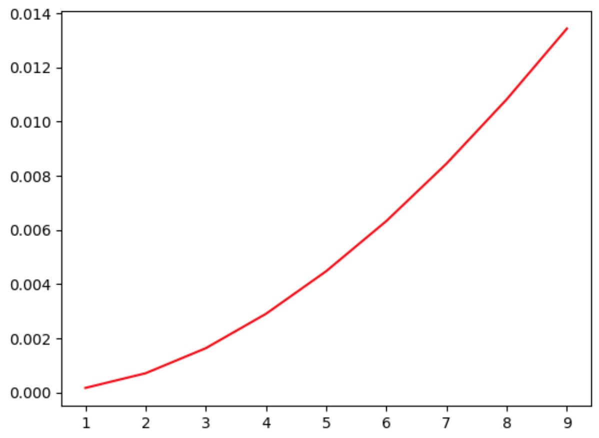


Figure 4: MSE (X-axis) of the N-step prediction (Y-axis)

We tried also to increase the number of both the hidden layers and the hidden nodes in each layer, but the performance of the model decreased for each of this configuration compared to the one described in the previous sections.

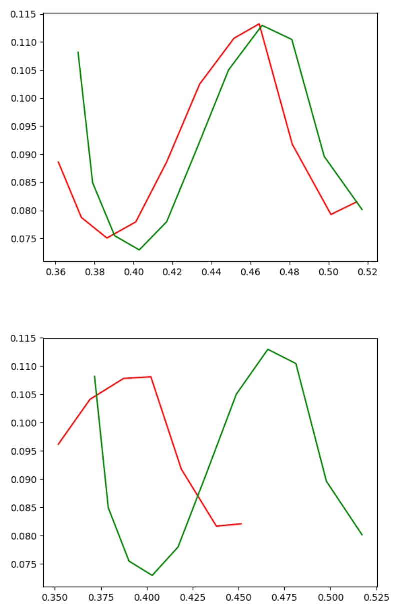


Figure 5: Comparison between predicted and real trajectory for increasing number of n-step multistep prediction (n= 1, 3, 5, 9

8 Conclusions

The results of the model on 1 and 2 step predictions present MSE that seems to be low enough to be considered good, while for multi-step prediction with a number of prediction steps ¿= 3 the performance is probably too low for any type of real application. Out guess is that to give more accurate long-term predictions about the movement of a molecule inside a cell we need a more detailed description of each Frame that includes other information other than just the position of the particle in that Frame.