Progress of Shuo
03/06/2020

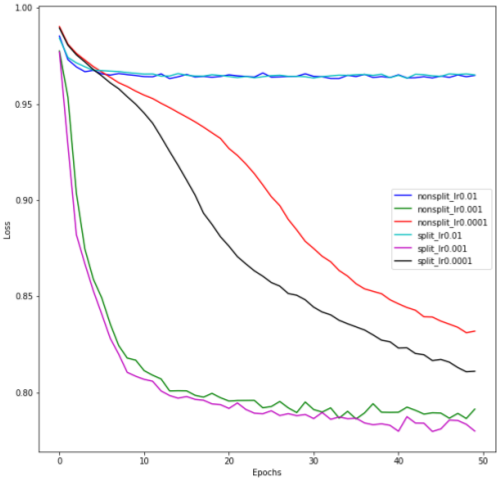
So far done (Next Plan is on the final page):

- Data Process of my own data collected
- Naive NN training with different setups of hyperparameter
    1. Learning rate= [0.01, 0.001, 0.0001]
    2. NN Type=[Split model for pos and load separately, Non-split model]
    3. Number of Hidden Layer Nodes=[200 for each layer, 512 for each layer]
    4. Other hyperparameters of network, such as activation function, number of layers, etc are the same as Liam's and Avishai's. Training were done for 50 epochs.
- Three modes of trajectory prediction

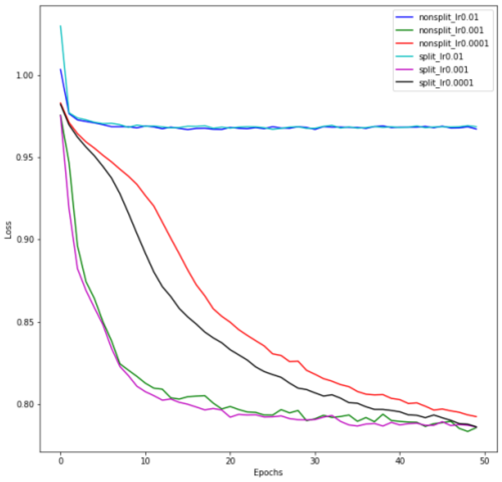
Hyperparameter search conclusions (plots below):

- Number of hidden layer nodes, 200 or 512, do not affect much.
- Split model might be slightly better than non-split model
- Learning rate of 0.0001 is the best among [0.01, 0.001, 0.0001].

Training Loss Plots:



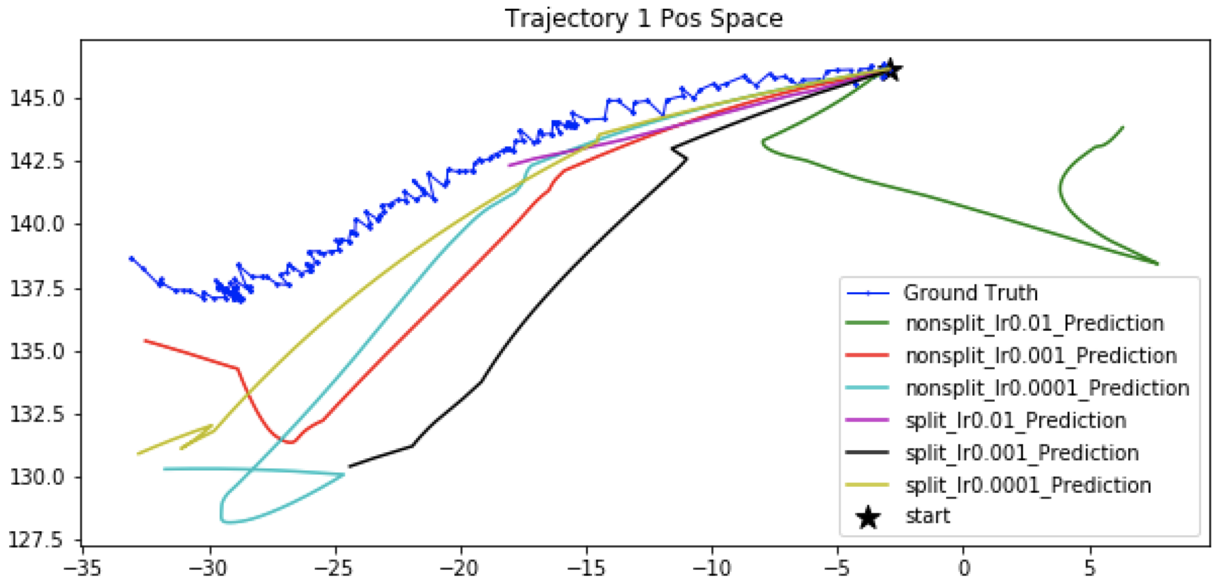
200 nodes



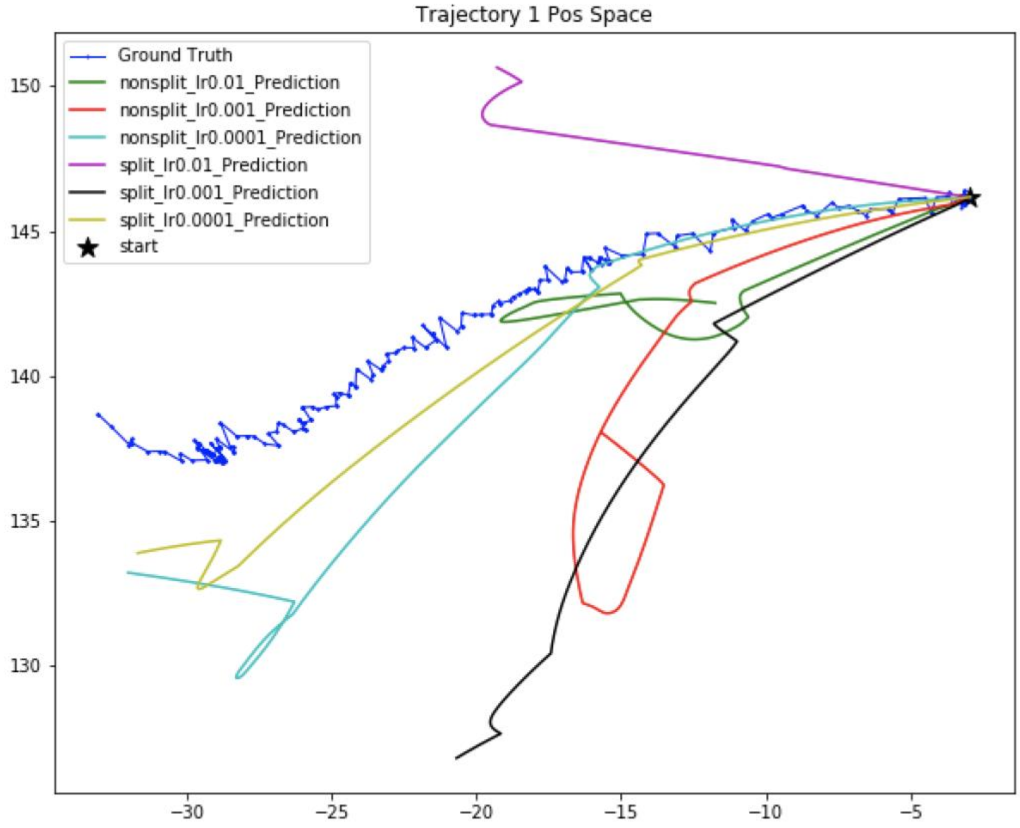
512 nodes

Trajectory Prediction Plots:

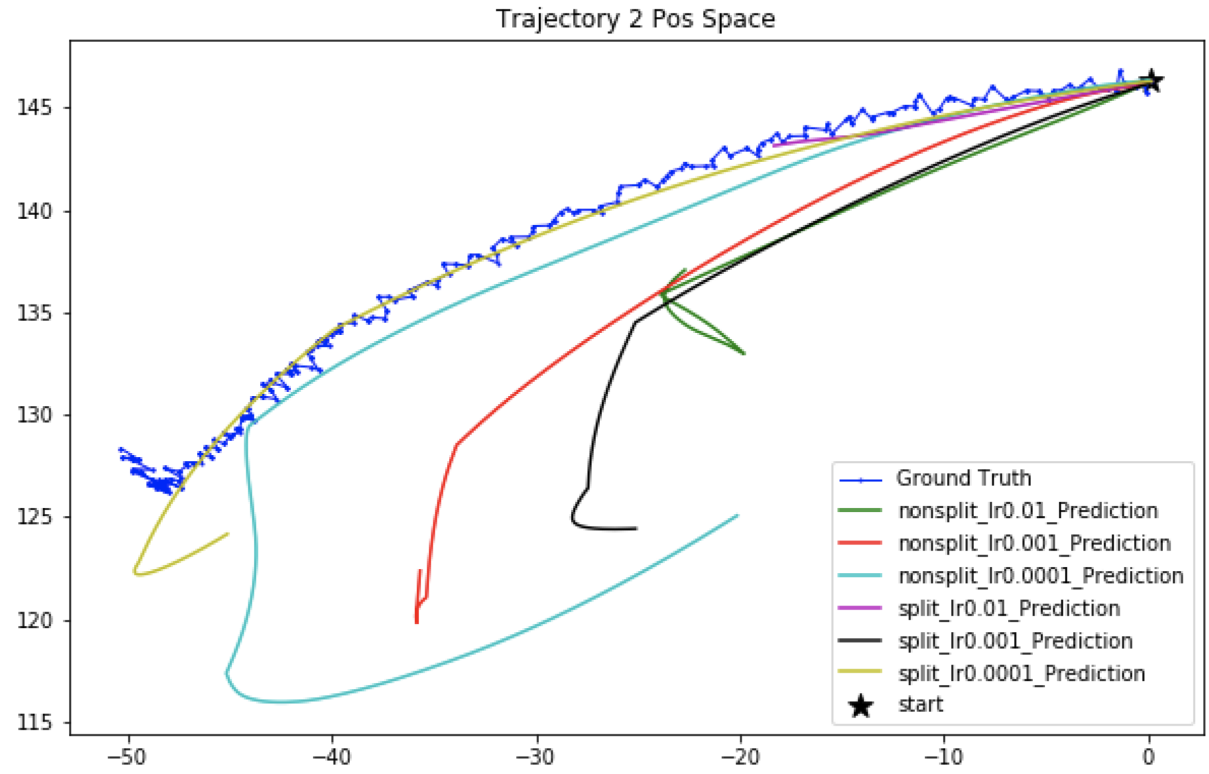
Given only the true state of the first step in a test episode, then predict the whole trajectory. The two test paths were shown below.



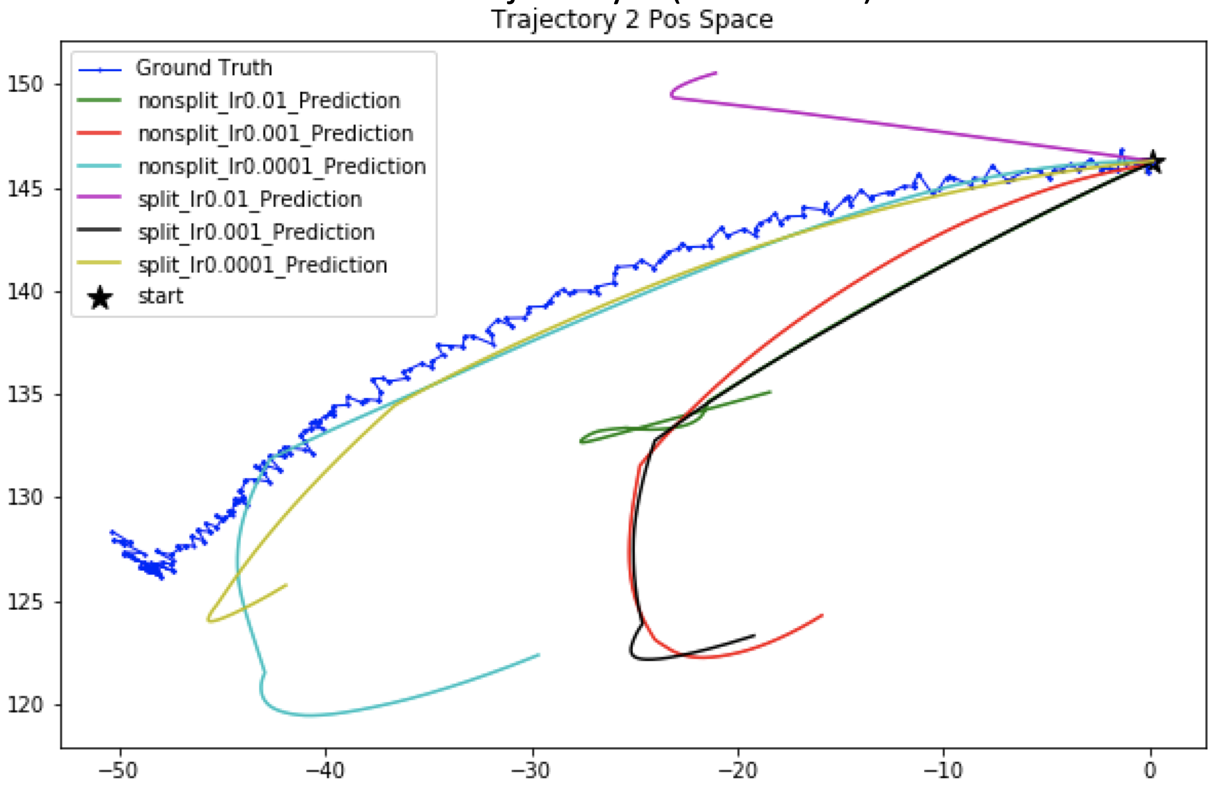
Test Trajectory 1 (200 nodes)



Test Trajectory 1 (512 nodes)



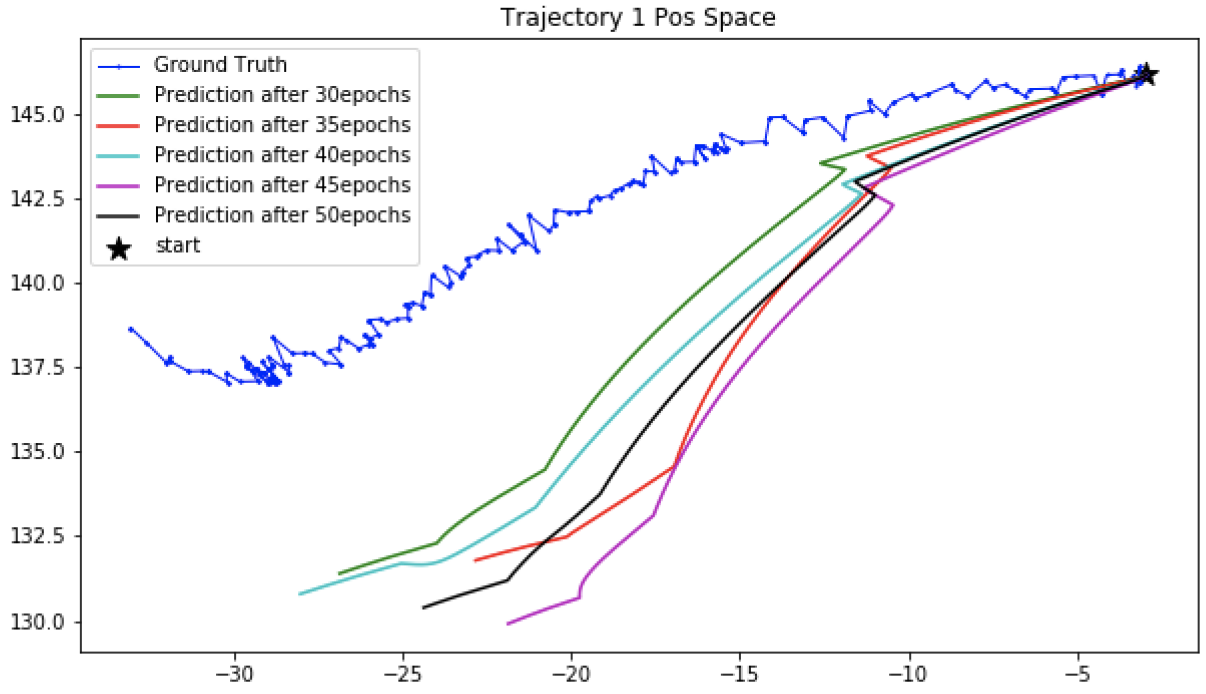
Test Trajectory 2 (200 nodes)



Test Trajectory 2 (512 nodes)

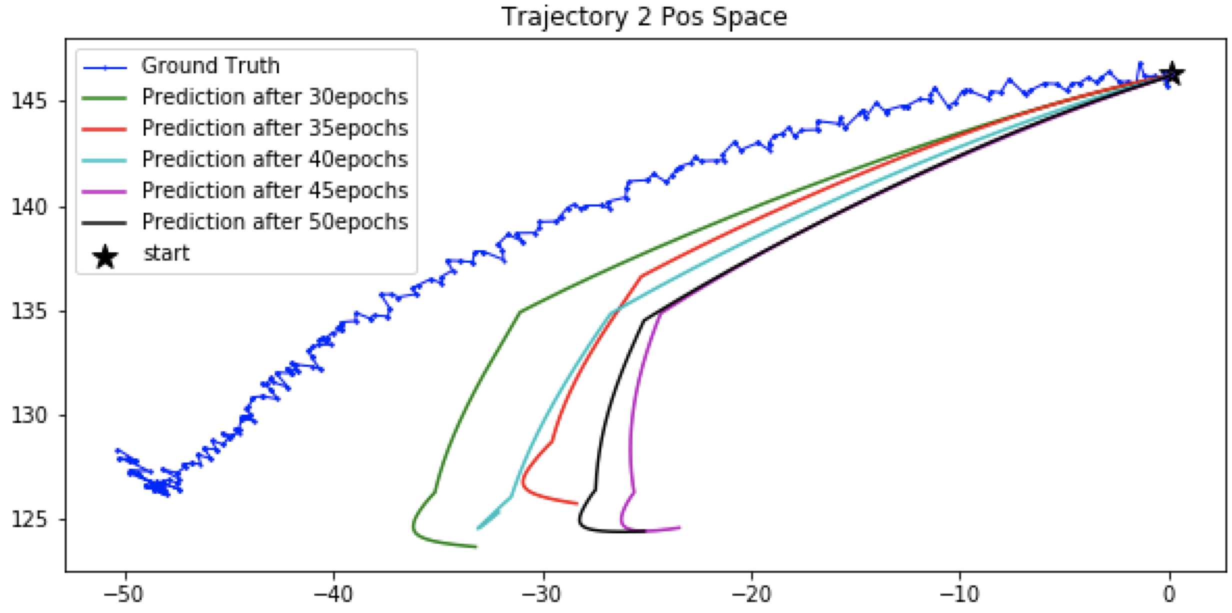
<u>Overfitting Case:</u>

With a faster learning rate, such as 0.001, overfitting was confirmed even within 50 epochs. Below are cases using learning rate 0.001, split model and 200 nodes. Trajectory prediction are plotted for after 30, 35, 40, 45 and 50 training epochs.



Trajectory 1 Pos Space

Test Trajectory 1 Overfit



Trajectory 2 Pos Space

Test Trajectory 2 Overfit

Three modes of trajectory prediction:

To find out why the prediction of the whole trajectory is not very precise, I plotted prediction in 3 ways totally.

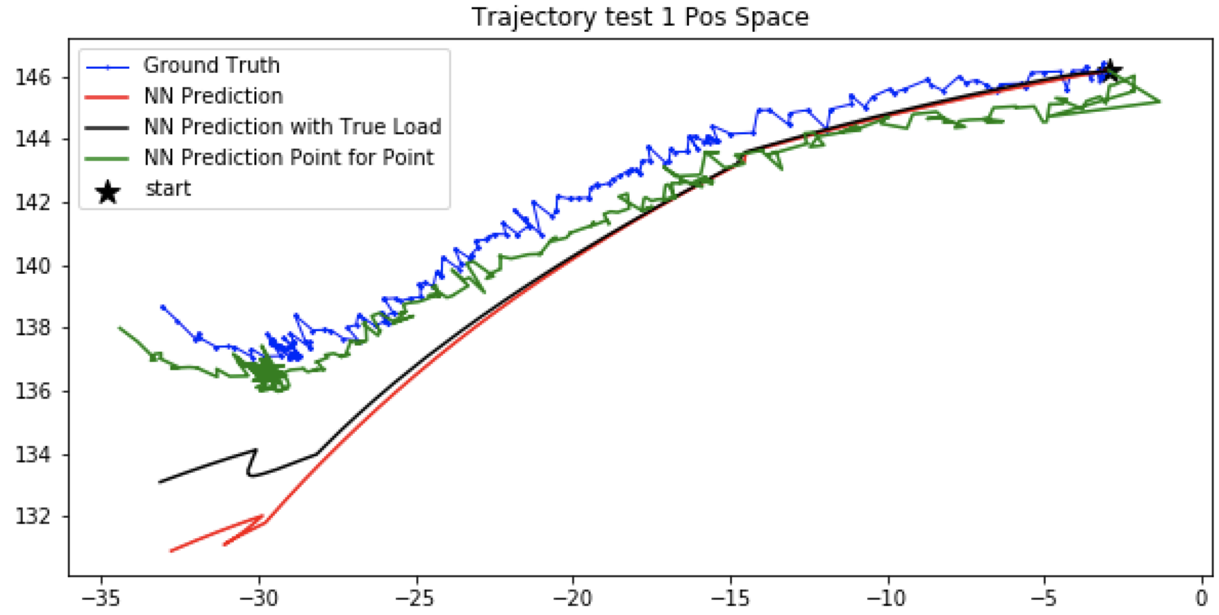
The first mode(shown as red lines below) is that only given the first step state of an episode, then prediction of the whole trajectory position. In this case, there would be error propagation since each state (both load and position) after the initial state is predicted one by one without any other ground truth information.

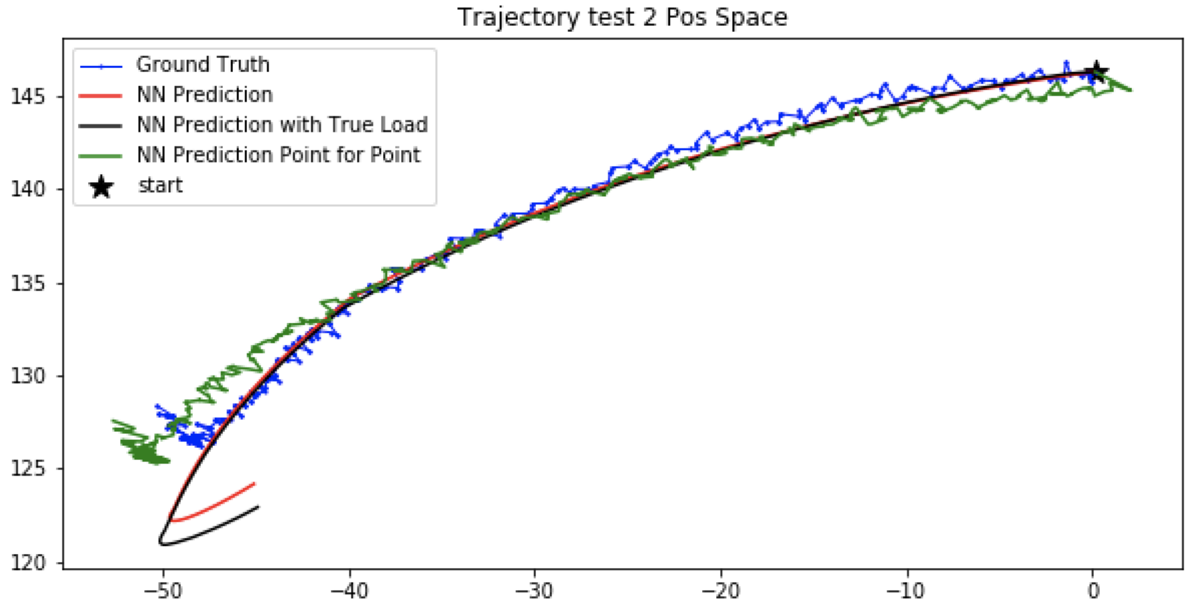
The second mode(shown as black lines) is prediction with true load information, meaning that given not only first step state of an episode, but also the true load information for each step afterwards, then prediction of the whole trajectory position. In this case, there would be less error propagation and the prediction should be better than the first mode, since we have the information of true load for trajectory's each step.

The third mode(shown as green lines) is point-for-point prediction, meaning that for every step the true state, both load and position, is given, then prediction of the next state position for each step (point for point). Thus, there is no error propagation in this case and the result should be the best.

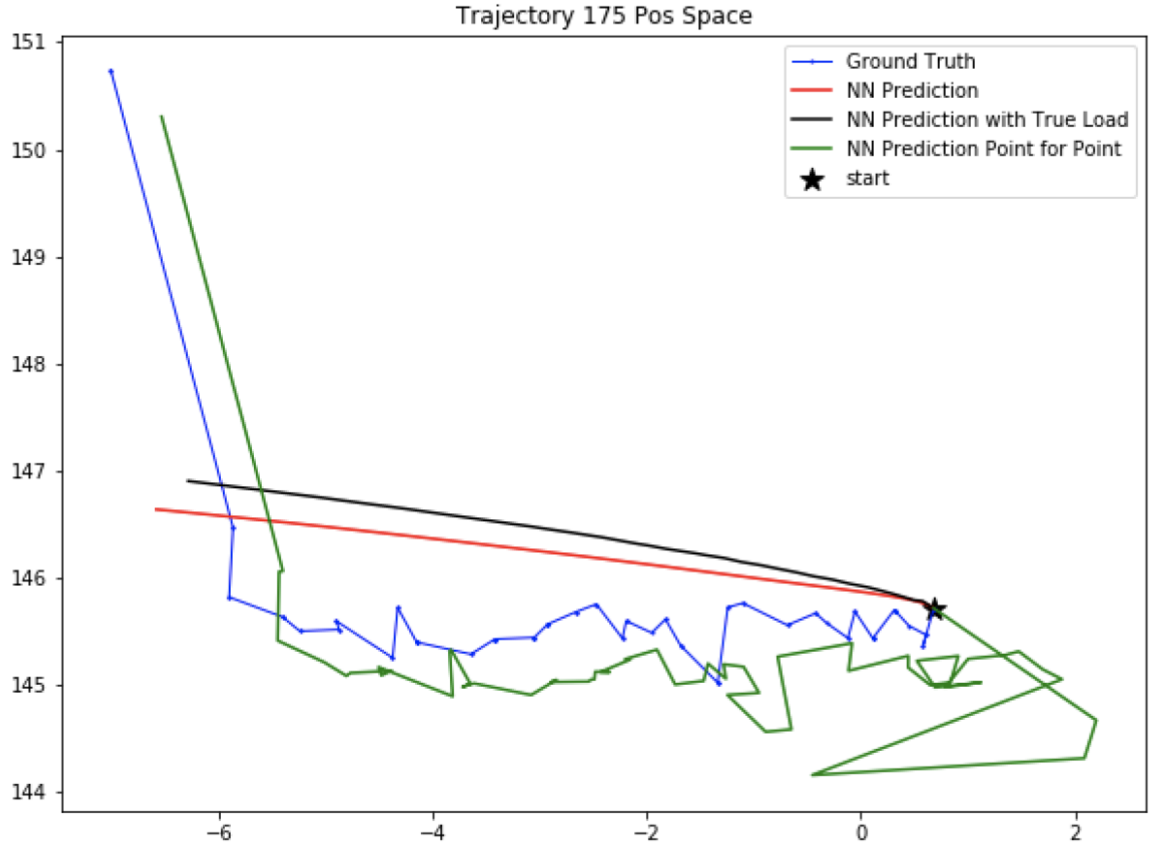
The plots below are done with the best hyperparameter setup, i.e. learning rate 0.0001, split-model, 200 nodes. We can see that the point-for-point prediction is the most precise one, and has only slight difference from the ground truth(blue lines). So, this means my NN model of predicting postions is generally working correctly and has learned something. However, the second mode and the first mode(black and red lines) are not as good as the point-for-point prediction due to the error propagation. However, they are close to each other, which means the model of predicting loads is also working. Otherwise, the second mode and the first mode prediction would differ much from each other.

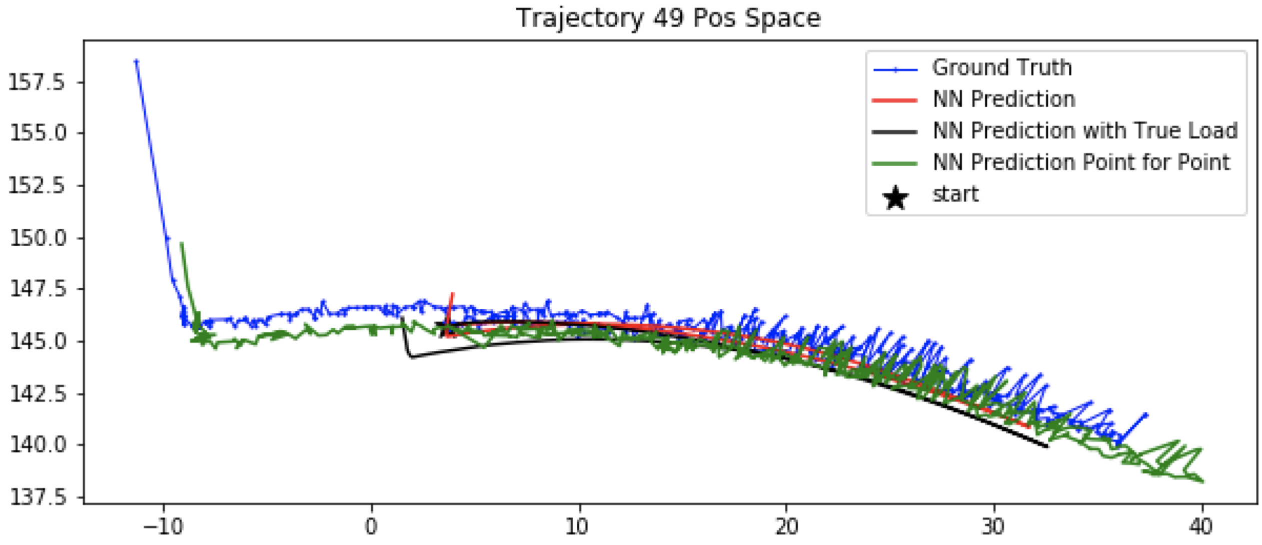
Therefore, both of my position model and load model are working, while the prediction performance is unsatisfactory since we want to finally predict a trajectory purely from the initial state only (i.e. the first mode). So, my guess would be the training data is too noisy and need to be further post-processed.

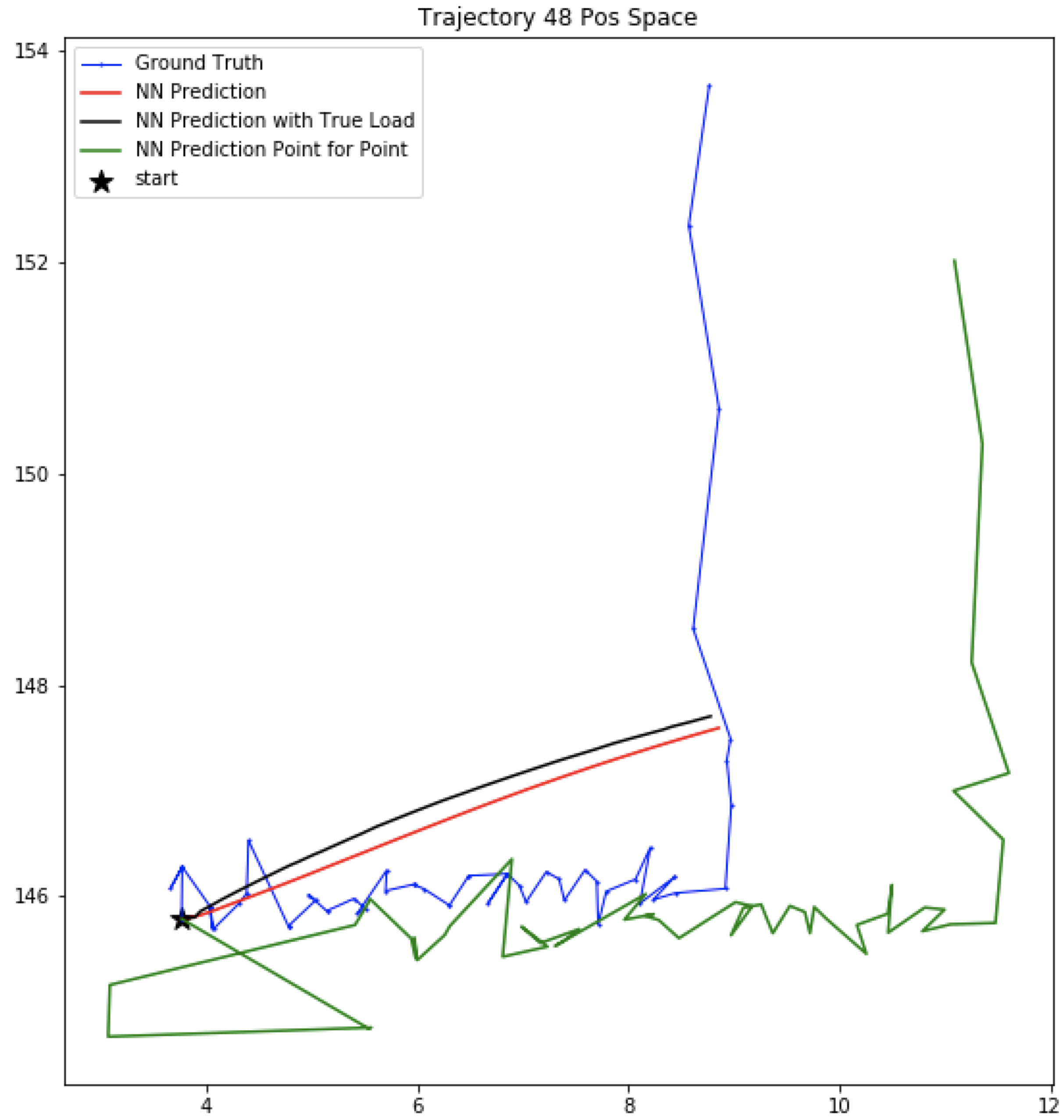




For reference, I also plotted some other trajectories with bad performance, which are actually even training trajectories. You can also see that I did not filter out the strong position change at the ending phase of the trajectory. So, filtering out such big change from one state to next state would be one idea of performance improvement.

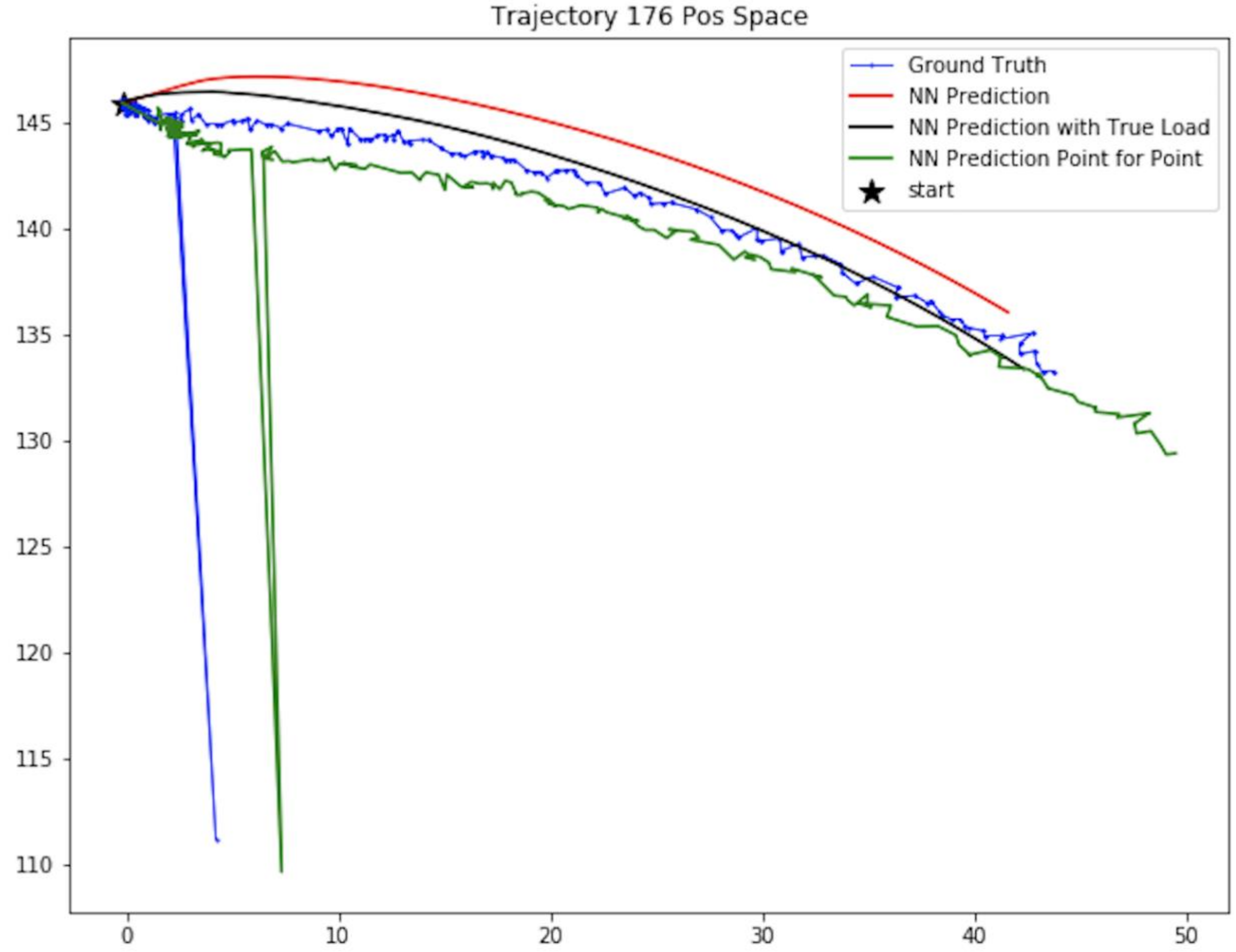


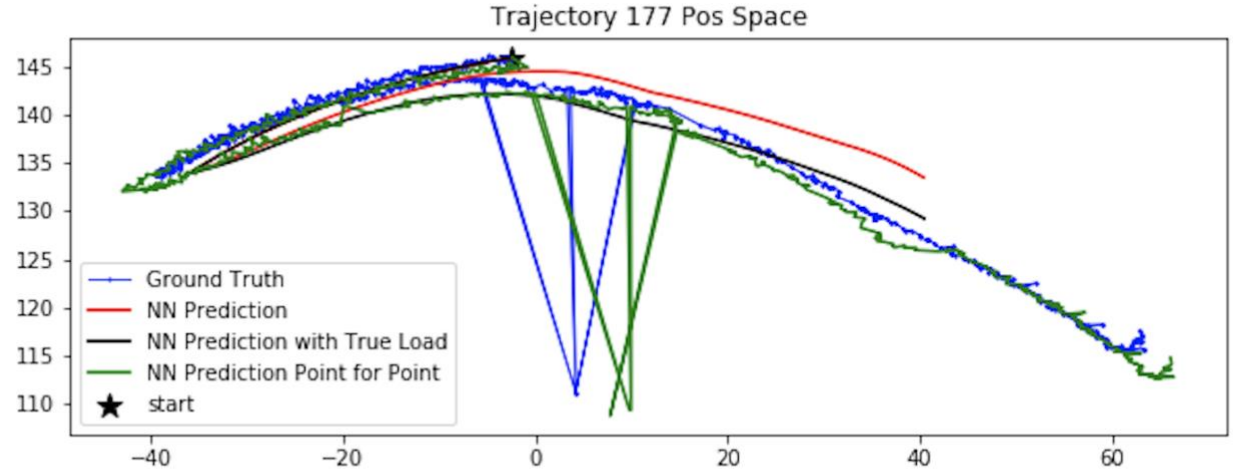




Evidence that my data was not clean:

I just recognized today that I should definitely set a threshold to filter out the big change from one state to the next state. I thought data would clean enough if I could filter out the noise caused by instability from visual detection, rotation matrix, object drop detection etc. But apparently I was wrong. I need to filter out more outliers using some value to limit the position change from one state to the next state. Actually, I also recognized today that Avishai used 1.2mm to clean such outliers. So, I would also further clean data with this value as a test. Below are some outlier cases even in the ground truth.





Next Plan:

1. Definitely Do: Further filter out outliers using a value (perhap 1.2mm) to limit the change from one state to the next state

2. Definitely Do: Then retrain NN with learning rate options[0.0002, 0.0001, 0.00005].

3. Would Do parallel: For newly processed data, train with Non-Split Model vs. Split Model; 200 Hidden Nodes vs. 512 Hidden Nodes.

4. Maybe Do (depending on the new results, we can discuss about this after I get new/better results):
    - Medfilter (which Liam seemed to not have used, but Avishai used. I personally prefer not to use because I think averaging 40 consecutive raw data to be one training data, would not the real position in the transition model.)

    - Training with one trajectory/episode as one input data. (I saw only in Liam's code that he also had another training mode for prediction of the whole trajectory and the loss would be the average deviation from the true trajectory while starting trajectory prediction from only the initial state, instead of naïve datapoint to datapoint prediction.)