CSE 251B Project Final Report

Awies Mohammad Mulla
Electrical and Computer Engineering
University of California, San Diego
La Jolla, CA 92093
`amulla@ucsd.edu`

Abstract

Trajectory prediction of the vehicles is one of the major problems in the domain of autonomous navigation. The vehicles are identified from the data (Argoverse Dataset used) extracted using sensors like LiDAR, RGBD Stereo Camera which could be then transformed to a pose to be used for the task of trajectory prediction. This is an important problem for safe functioning of various agents simultaneously, along with safety of pedestrians. Not only should the results produced during this step be precise but also the corresponding controller should function with same level accuracy. We will focus on the prediction step in this paper. Over the past years various approaches and algorithm have been introduced to tackle this problem mainly along the lines of unsupervised learning. In this paper we will first introduce the problem and present an analysis on the dataset used for the said task. Moving on to discuss the details of the various algorithms experimented along with the implementation details. Finally, we will present the results produced. Since, this is a milestone report on the project, better results are expected to be presented in the final report by using the approaches whose overview are mentioned in the section future works.

1 Task Description and Background

For this project we are provided a simplified version of Argoverse Dataset recorded in Miami and Pittsburgh. Original dataset contains the raw sensor data from the sensors like LiDAR, RGBD Stereo Camera, GPS, IMU, etc. The simplified version of the dataset contains the 2D pose and velocities of the vehicles in the world frame along the information about the neccessary information about the environment like the coordinates of the lanes and directions represented by the lane markers. The link to scripts used to generate is here. And the background and applications of this project are mentioned in the abstract.
Our goal is to predict 2D pose of the agent for 30 timestep in the world frame, given the past 19 timestep information (2D pose and velocity in the world frame) of the agent and the the details of the environment. We are given the coordinates of the lanes on the road, direction of the lane markers (lane norms) and name of the city.

Objective: Minimize the following loss function -

$$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{30}\|\hat{y}_{i,j} - y_{i,j}\|_2 \tag{1}$$

where $\hat{y}_{i,j}$ is the predicted 2D pose of the agent at timestep j for the i^{th} sample and $y_{i,j}$ is the ground truth 2D pose of the agent at timestep j for the i^{th} sample. N is the number of samples in the dataset. $\hat{y}_{i,j}$ is defined as

$$\hat{y}_{i,j} = f(x_{i,j:(j-19)}, c_i) \tag{2}$$

where $f(.)$ is the model used for prediction, $x_{i,j:(j-19)}$ is the 2D pose and velocity of the agent at timestep j for the i^{th} sample and c_i is the information about the environment for the i^{th} sample.

The model defined above can potentially solve other tasks apart from motion forecasting of autonomous vehicles. For example, it can be used for trajectory prediction of multiple pedestrians, cyclists, etc. which is an important problem in the domain of autonomous navigation. The problem of multiple agents and pedestrians can be tackled by slight modification in the loss function. This model can be also used in the domain of Human-Robot Interaction (HRI) for predicting the trajectory of the human in the environment without any modification in the objective. The model can also for optimizing the delivery routes, by changing the given information as to only include initial position of the agents with the final position being the destination. This is plausible as we would need minimize the delivery time by considering the traffic conditions as the information about the environment.

2 Exploratory Data Analysis

- The dataset is of the form of a dictionary containing the different scenes. Each scene sample has a tracking id of the agent to be tracked along with information about other agents. The size of the given training set is 205942 and the size of the test data is 3200.
- We remodeled the data such that each sample only consist information about the agent to be tracked (the size of the dataset would still be same). This was done as for this project we are only focusing on the motion forecasting using information of single ganet. Of the 205942 samples we will be using 180000 samples for training and 25942 samples for validation after shuffling.
- Since, raw data consists the information about all agents (which is atmost 60 in a scene). The dimensions of the input and output 2D pose are $60 \times 19 \times 2$ and $60 \times 30 \times 2$ respectively. We are also provided with the input and output velocities of the agents. The dimensions of the input and output velocities are $60 \times 19 \times 2$ and $60 \times 30 \times 2$ respectively. For the agents not present in the scene, the corresponding input and output 2D pose and velocities are filled with zeros.
- Since, currently we are only using the 2D poses of the agent to be tracked the current dimension of the input is 19×2 (Vectorizing depends on the type of model to be analysed; for example in an MLP we will be using 38×1). The dimension of the required output is 30×2. In some cases, we also use the velocities of the agent as an input to the model. In that case the dimension of the input is 19×4 (and corresponding dimension of the vectorized version).
- We are also provided with the information about the environment. The information about the environment is of the form of a dictionary containing the coordinates of the lanes, the direction of the lane markers (lane norms) and the city in which the data is recorded (in form a string). The size of the arrays of lanes and lane norms are same. The dimension of this array is not fixed and is depended on the scene (with maximum of 1083 lane coordinates).
- One sample of the dataset is shown below (note that we are plotting the raw data so all the agents present in the scene are plotted) in Figure 1. The red lines indicate the input trajectories and the blue lines indicate the output trajectories.
- We will be analysing the datasets according to the city they are recorded in for better understanding of the distribution of the data. The dataset is recorded in two cities namely Miami and Pittsburgh. The heatmap of the distribution of the input data for the two cities are shown in the figure. Figure 2 shows the heatmap for the city of Pittsburgh and Figure 3 shows the heatmap for the city of Miami.
- We simply plotted the world frame coordinates of all the agents in the dataset for inputs and outputs seperately. For the city of Pittsburgh, many of the input coordinates is near the intersection. This shows that the agent can either go straight or take a turn. This can be deduced from the output distribution, which shows that we have enough samples (more than moving straight on the intersection) so that model would consider the turn and not overfit on agent going straight.

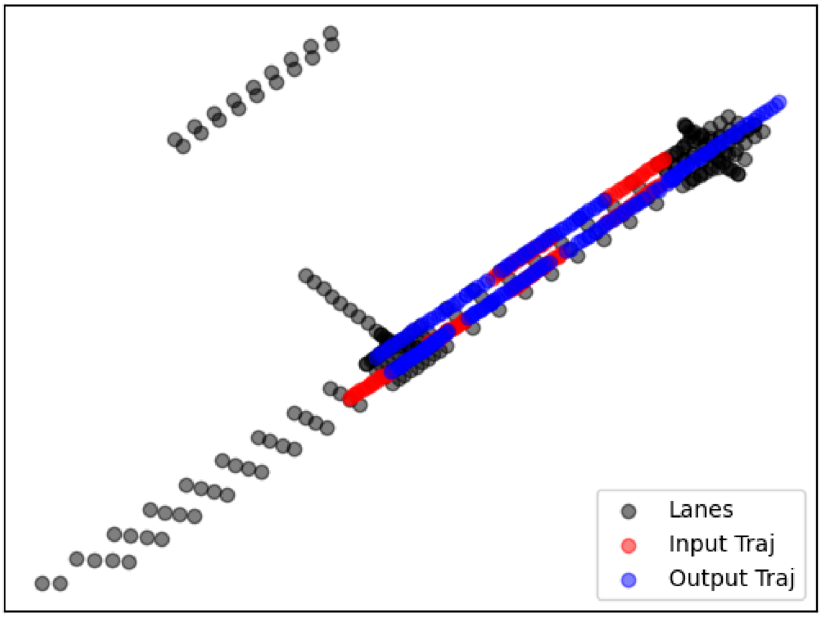


Figure 1: Red indicate input trajectories and blue indicate output trajectories

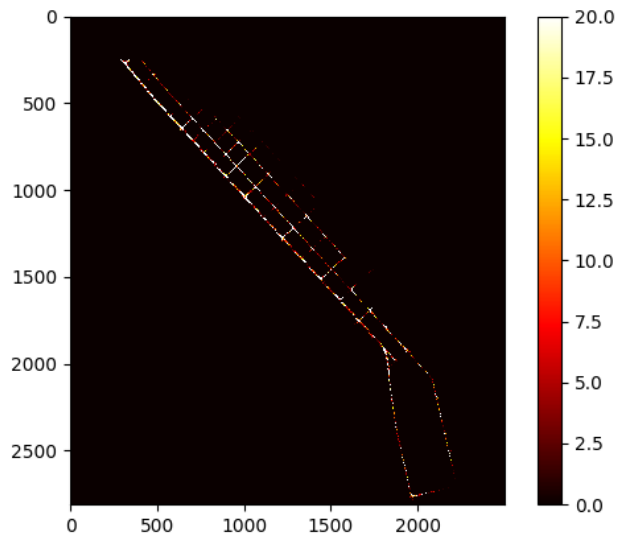


Figure 2: Heatmap of the distribution of the input data for the Pittsburgh

- For the city of Miami, the input and output distribution is concentrated on the straight road. This tells us that the agent can either move straight or change the lanes. The classificstion of these cases is explained in the next steps.

- From plotting the trajectories seperately for each agent we observed that there are four major cases for the agent: moves straight, is parked or at rest, taking a turn and finally changing the lanes. The figure shown below shows the distribution of the four cases for both the cities seperately.

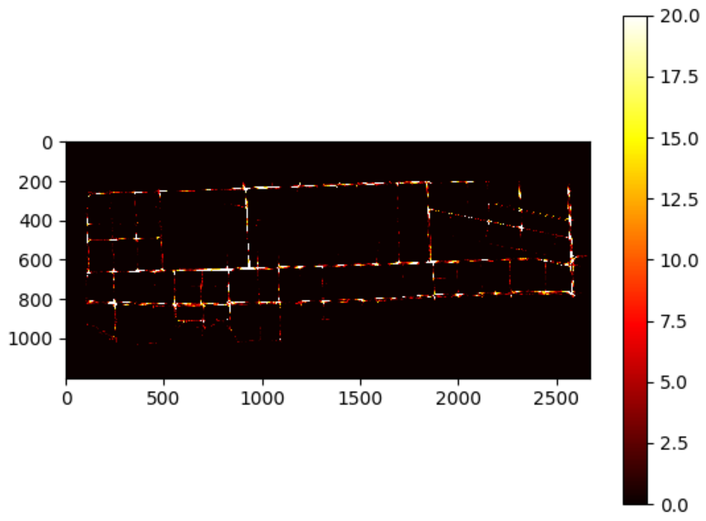


Figure 3: Heatmap of the distribution of the input data for the Miami

City	Straight	Turn	Lane Change	At Rest
Pittsburgh	92357	1917	1247	0
Miami	106420	2287	1714	0

Table 1: Classification of the agents

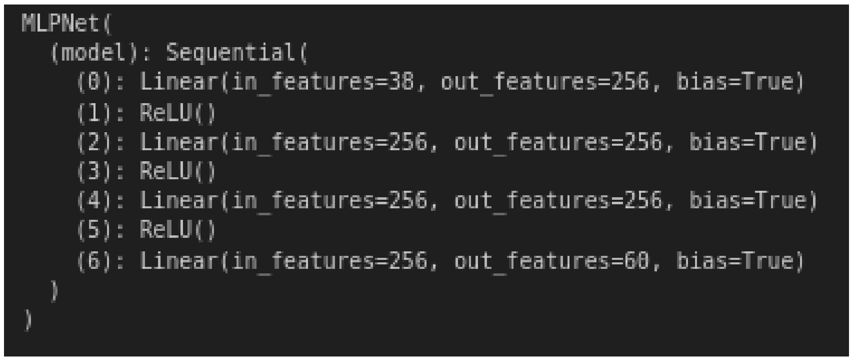
- To check whether a particular agent taking turn we check the angles made by the trajectories of the agent. Here threshold angle for a turn is taken as 60 degrees. Approach to classify the vehicle at rest is straightforward, we simply look through the velocity data of the agent. To classify agent moving in a straight line we try to fit the trajectory of the agent to a line and check the error. If the error is less than a threshold value it is traversing in a straight line. Finally, all the samples which does not fall in any of the above three categories are classified as changing lanes. The information of this classification is summarized in the Table 1.

- Since, the data provided is numerical in nature we did not any feature engineering for representation of the data. although as mentioned above we classified the data into four categories, this information was used only for analysis and hyperparamter tuning.

- As, the 2D pose provided in the dataset is in the world frame, we normalized each of the input and output 2D pose with respect to the first input 2D pose so that we can focus on the patterns of the trajectories rather than the absoluted values. This will help the model to learn the patterns more efficiently.

- We used the city information provided to train the model seperately based on the city. This was done as the distribution of the data is different for the two cities. And it would be easy for model to learn the patterns if the data is trained seperately.

- Initially we tried to normalize the data in the range of $[0, 1]$ but the model was not able to learn the patterns effectively. This was because the of the vanishing gradient problem. Since, the values of the input and output data were in the range of $[0, 1]$, the gradients were also in the same range resulting in very low loss and small gradients, hence the model was not able to learn the patterns.



```
MLPNet(
  (model): Sequential(
    (0): Linear(in_features=38, out_features=256, bias=True)
    (1): ReLU()
    (2): Linear(in_features=256, out_features=256, bias=True)
    (3): ReLU()
    (4): Linear(in_features=256, out_features=256, bias=True)
    (5): ReLU()
    (6): Linear(in_features=256, out_features=60, bias=True)
  )
)
```

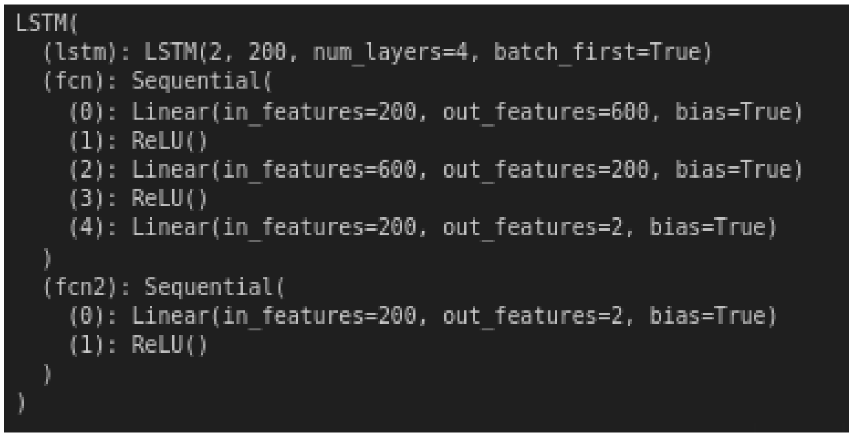
Figure 4: MLP Model Architecture

3 Deep Learning Model

As mentioned above we remodeled the given the dataset such that each sample consisted the information about the agent to be tracked. We described the preprocessing done on the dataset in the previous section including the normalization, transfromation of the coordinate axes and lastly classification based on the cities. Now, we fed the processed data directly to the model. We tried two main models in this project. The first model is a Multi-Layered Perceptron (MLP) model and the second model is based on LSTM architecture.

- For the MLP, we followed a straightforward approach as the input and output data are numerical in nature. And as we are only using 2D poses (and velocities in some cases) the size of input and output is fixed. We vectorized both the input and output data (processed) and fed it to the model.
- The model consists of 3 hidden layers with 256, 256 and 256 neurons respectively followed by ReLU activation function. We experimented with both L1 and MSE loss functions and concluded that MSE loss was better for training the model (intuitive). But we saw that L1 loss converged faster than MSE Loss, hence, we later used the L1 loss for testing purposes.
- For the LSTM based architecture, we remodeled the data such that the input and output data have same shape (19 timesteps). The label consisted of the 2D pose of the agent at the 20th timestep and so on.
- The model consists of an LSTM layer with 3 hidden layers with 200 cells in each which is then followed by a MLP with 4 layers with 600, 600, 200 and 2 neurons respectively. The output of which is again fed into an LSTM layer which is exactly same as the first one. This architecture is similar to the basic Encoder-Decoder architecture.
- We chose the model mentioned above because it is quite effective for sequence to sequence prediction. And as the input and output data is sequential in nature, we thought of using this architecture.
- We used the MSE loss function for training the model. We also tried using the L1 loss function but faced similar problem as mentioned for the MLP. Hence, we decided to go with the MSE loss function with Adam optimizer.
- We tried using the dropout layer and L1 regularization in the MLP model but it did not help in improving the performance of the model. This might be due to large size of the dataset compared to the number of parameters in the model.
- For LSTM model, we used the dropout layer with a dropout rate of 0.2. This helped in improving the performance of the model. As mentioned above, the LSTM was quite dense and hence, the dropout layer helped in regularizing the model.
- The model architecture for both the models is shown in the Figure 4 and Figure 5 respectively.

- We mainly focused on tuning the batch size, learning rate, number of layers (for MLP model) and number of cells (for LSTM model). We did hyperparameter tuning on a subset of the



```
LSTM(
  (lstm): LSTM(2, 200, num_layers=4, batch_first=True)
  (fcn): Sequential(
    (0): Linear(in_features=200, out_features=600, bias=True)
    (1): ReLU()
    (2): Linear(in_features=600, out_features=200, bias=True)
    (3): ReLU()
    (4): Linear(in_features=200, out_features=2, bias=True)
  )
  (fcn2): Sequential(
    (0): Linear(in_features=200, out_features=2, bias=True)
    (1): ReLU()
  )
)
```

Figure 5: LSTM Model Architecture

Hyperparameter	MLP	LSTM
Batch Size	512	256
Learning Rate	0.001	0.001
Decay Rate	0.95	0.95
Optimiser	Adam	Adam
Loss	MSE, L1	MSE, L1
Epochs	500	100
Number of Layers	3	3 + 3 + 3 + 1
Number of Cells	-	200

Table 2: Hyperparamters used for training the model

data (as mentioned above). Other hyperparamters were adopted from literature review. The details of the hyperparamaters used are mentioned in the Table 2.

4 Experiment Design and Results

The details of the implementation of a deep learning model and the analysis done to arrive at a particular model is mentioned in this section.

4.1 Experimental Setup

- We used mainly two devices for this project. The device used for small testing purposes and deciding on minute implementation is laptop with 8GB RAM and 12th Gen Intel(R) Core(TM) i7-1255U. For training the final model we used the device provided by the UCSD Datahub platform which contains Intel(R) Xeon(R) Gold 6230 CPU @ 2.10GHz, 16GB RAM and 1 GPU, GeForce RTX 2080 Ti with 11GB memory.
- We used the Adam optimizer with learning rate of 0.001, decaying at each step with rate of 0.05. We did hyperparameter tuning on a subset of the data. As mentioned above we classified the data into four categories of the motion of the agent. So, we sampled few samples from each of the four categories to form a subset of the data for hyperparameter tuning. The values with which we experimented were chosen based on the literature review done in the past.
- We also experimented on various batch size. For testing purposes we used batch size of 32 on the subset of training data and trained the model for 100-500 epochs (500 for MLP and 100 for the LSTM) for the loss to converge. For the final model we used batch size of 512 and trained the model for 500 epochs (or early stopping) which used up lot of resources and time.
- MLP takes about 6 seconds to go over the complete training data for one epoch whereas, LSTM takes about 200 seconds to go over the complete training data for one epoch.

- As mentioned above for the MLP we predicted the 2D pose of the agent as a vector of length 60, whereas, for the LSTM we broke the complete trajectory into subsequences and predicted the last term of the subsequence. Hence, we had to remodel the data for this purpose.
- The number of parameters used in the MLP model is about 6.5 million whereas, the number of parameters used in the LSTM model is about 1 million.

4.2 Results

- The training and validation loss for models are mentioned in Figures 6 and 7 respectively.

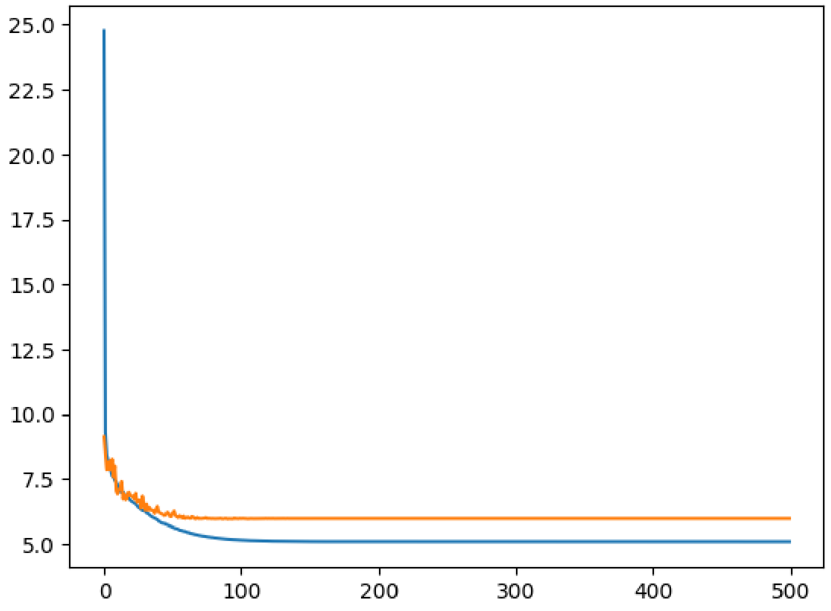


Figure 6: MLP Training and Validation Loss

- Few training examples are shown in the figures below (Red refers to the input trajectory, Blue refers to the predicted trajectory, and Green refers to the ground truth trajectory):

- Current ranking (Awies Mulla) - 10th on the leaderboard. Final MSE - 628.

5 Discussion and Future Work

- For this project most effective strategy for me turned out to be transformation of the coordinate axes as this step reduced the variance in the data by a large margin and hence, we were significantly able to reduce the MSE.
- The other strategy that worked for me was to use the LSTM model. The LSTM model was able to capture the temporal dependencies in the data and hence, was able to predict the trajectory better than the MLP model, hence reducing the MSE by a large margin.
- One of bottleneck of this project was when we were training on a MLP model, trying to achieve results which were beyond the capacity of the model.
- We should always start by analysing the data and building a mini dataset which should be used for most of the analysis. Later, train a model on the complete dataset once we get satisfactory results on the mini dataset. Understand the bottleneck of the model with respect to data and one's implementation.

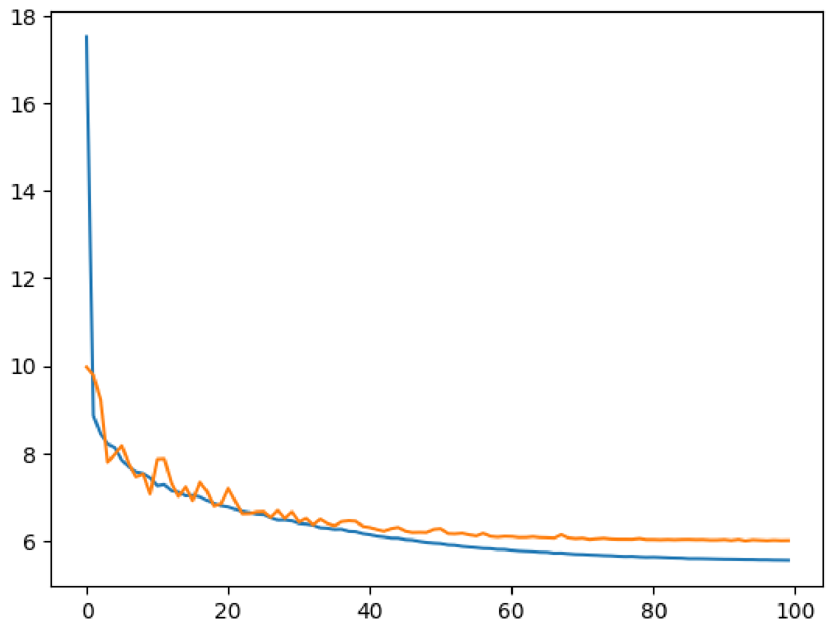


Figure 7: LSTM Training and Validation Loss

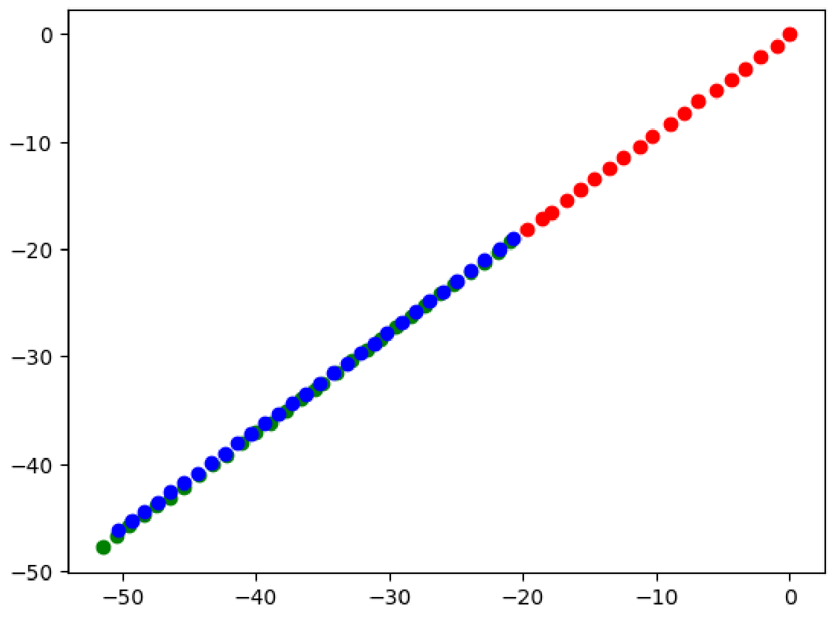


Figure 8: MLP Training Example

- We would later like to experiment with Attention mechanism and see if it improves the performance of the model. We would also like to try implementing models openly available in the literature and see if we can improve the performance of the model.
- One of main idea with which we would like to experiment is trying to incorporate the Kalman Filter with the output of the model. The output from the could be labeled as the observations obtained and use the Euler's discretized equations of motion as the motion model to predict the trajectory. We ae confident that this would improve the performance of the model significantly. We were not able to implement this idea due to time constraints.

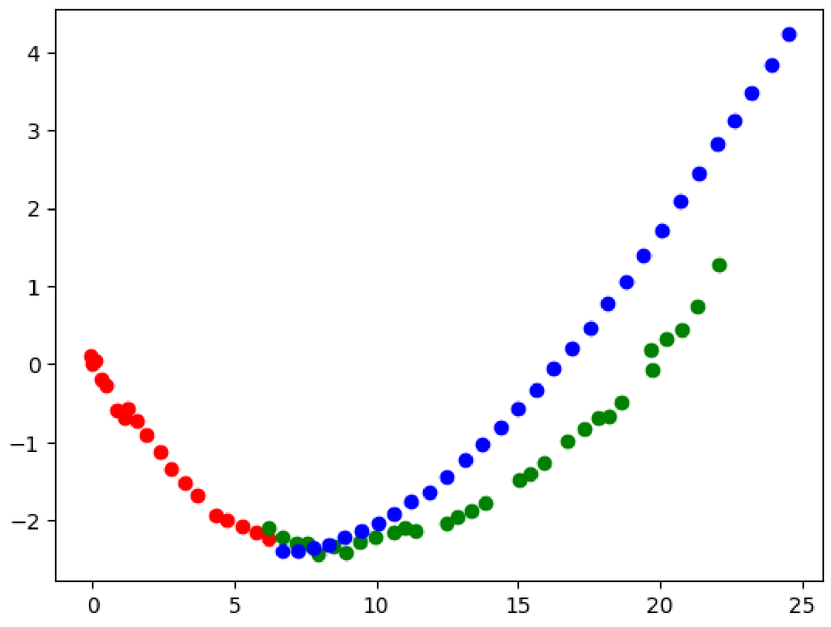


Figure 9: LSTM Training Example

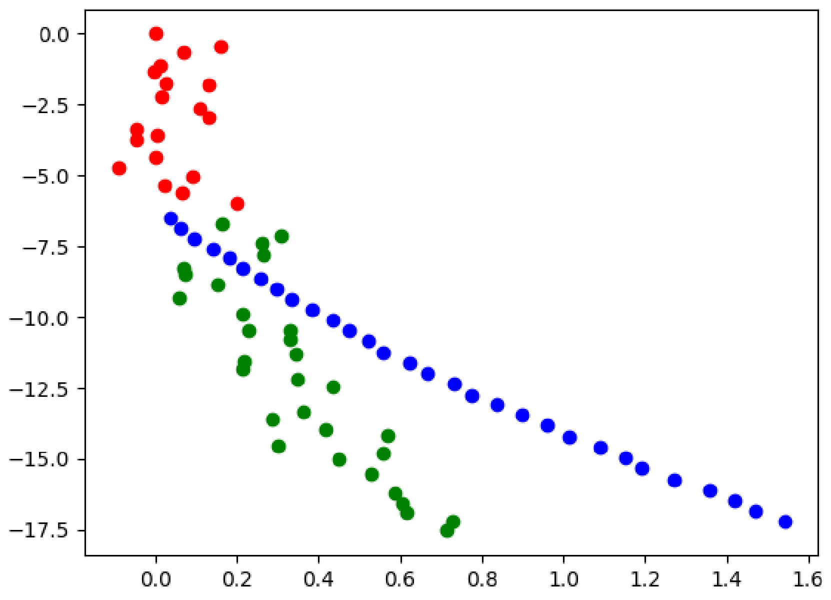


Figure 10: MLP Training Example

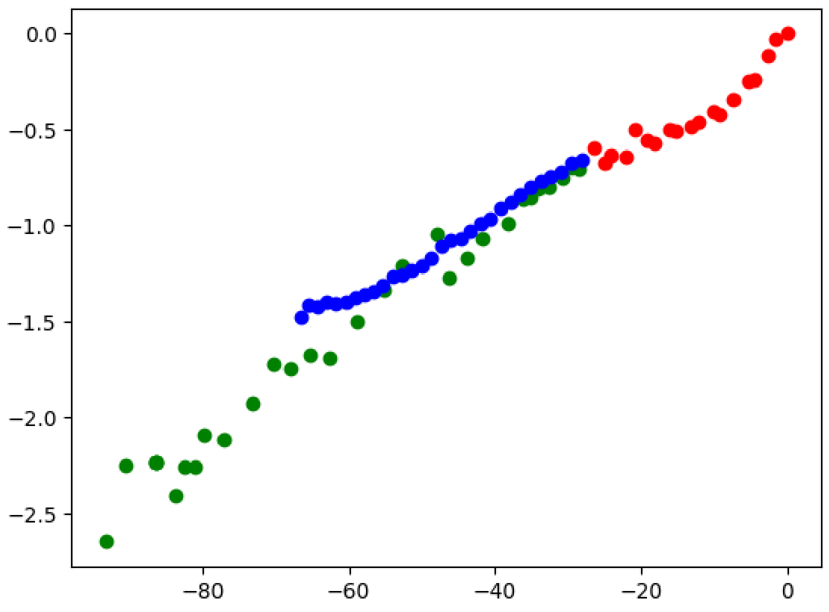


Figure 11: LSTM Training Example

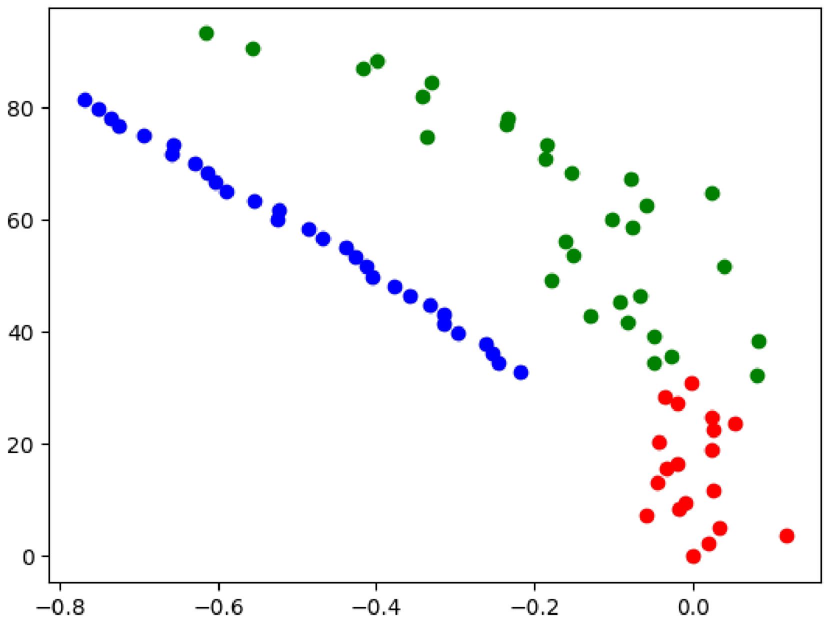


Figure 12: MLP Training Example

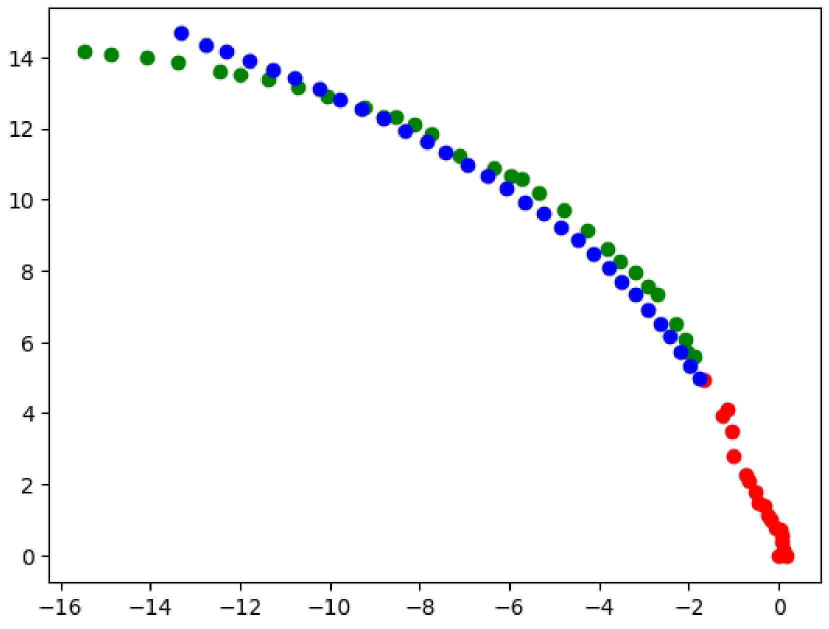


Figure 13: LSTM Training Example

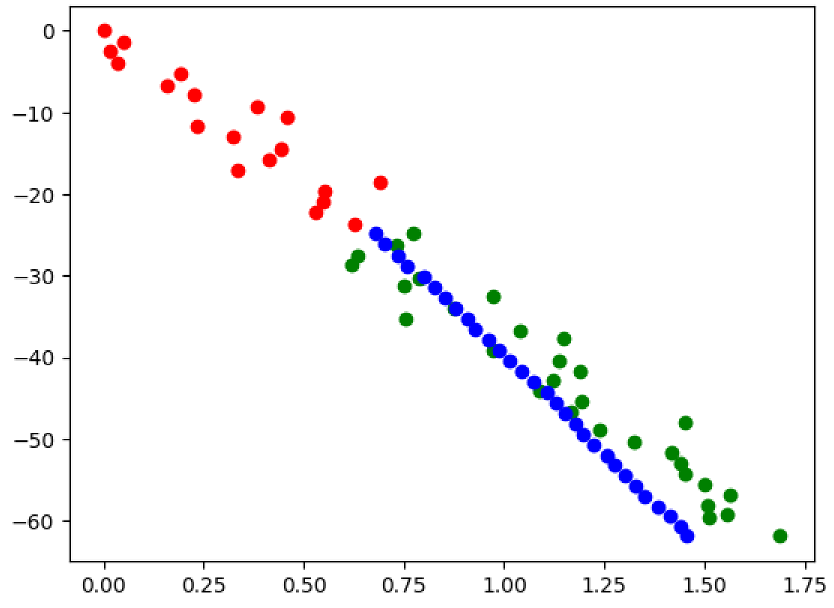


Figure 14: MLP Training Example

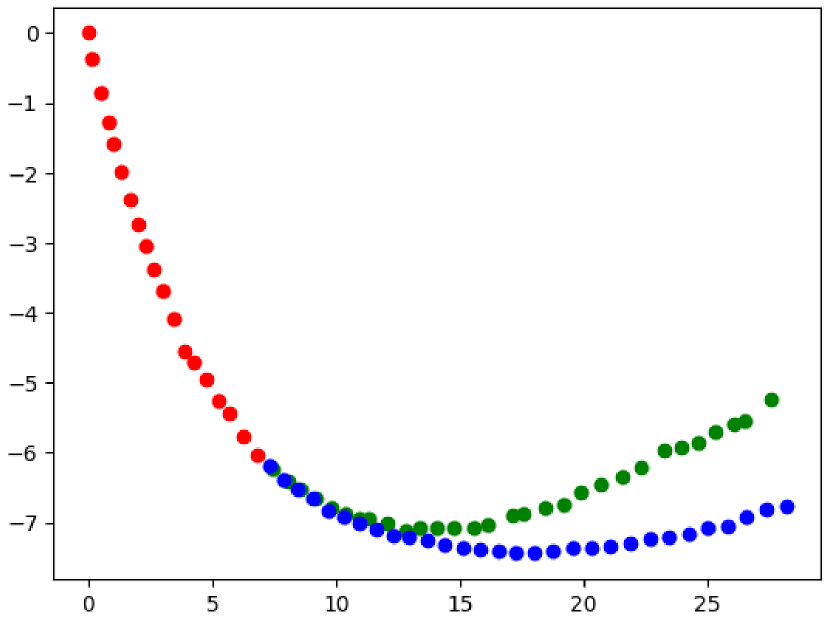


Figure 15: LSTM Training Example