CSE151B Final Report: ML Noobs Group

Jeffrey Feng || Dennis Wu || Tat Hei Tsin || Yujian He
A15507377 || A15537107 || A15541127 || A15386248

Abstract

This is the mile stone report for CSE151B deep learning project for Autonomous vehicles(AV). We are given 19 timestamps information for all agent such as position and velocity and expected to predict the next 30 time steps. We include my approach using deep learning model and linear regression with supportive data.

1 Task Description and Background

1.1 Describe in your own words what the deep learning task is and why it is important. Provide some real-world examples where solving this task can have great impact on our daily life and the society.

The background of the task focuses on a spatial-temporal task, where the deep learning model is required to predict future trajectories of moving objects like cars on roads based on their previous positions and velocity. By improving the accuracy of the prediction, it could benefit the forecasting technology of fully autonomous driving vehicles, which would increase the safety of driving without supervision. Therefore, solving this problem can boost the progression of autonomous driving technology. Moreover, in the domain of traffic forecasting, predicting the trajectories of vehicles could give a better prediction and understanding of traffic conditions in major cities, thus benefiting efficiency of highway and public transportation planning.

1.2 Use Google Scholar or other internet resources to research on this task. What type of methods have been examined before? Include some references and discuss their ideas in a few sentences. You can use Bibtex to manage your bibliography

Across the methods that have been applied on this task before, convolutional neural networks (CNN) (Ma et al., 2017) (Yu et al., 2017), recurrent neural networks (RNN) (Cui et al, 2018), variants like long-short term memory model (LSTM) (Feng et al., 2019), autoencoder like Seq2seq, and graphical convolutional networks (GCN) (Zhao et al., 2017) are all applied on this task. We would focus on works that are more interesting to use, which includes the T-GCN by Lehai Feng that predicts urban traffic flow using Graph Convolutional Network (Feng et al., 2019), the work by Zhao et al. that predicts the short-term traffic using LSTM network (Zhao et al., 2017), the model built by Yu et al. that forecasts the geospatial coordinates of traffic using Convolutional Neural Network (Yu et al., 2017). Correspondingly, we thought of building our model based on CNN, RNN and Linear Regression. Feng's T-GCN transforms the dataset into a graph, which can be further learned by the CNN model that enables us to predict the car's trajectory. The work by Zhao et al. made us realize the power of the RNN by training the model using LSTM.

1.3 Define the input and output in mathematical language and formulate your prediction task. From the abstraction, do you think your model can potentially solve other tasks beyond this project? If so, list some examples and explain your rationale

For this task, we are given a dataset that consists of individual scenes. Within each scene, the input matrix for the model can be described as $mxnxp$, where m is the number of autonomous vehicles being tracked within the scene. $m = 60$ in all our cases, but there are cases where the number of tracked vehicles is smaller than 60. N equals the number of time steps recorded, where the recording interval is between 2 seconds. In all the cases, $n = 19$. p equals to the features the model is going to intake, which includes p_in and v_in p_in contains a vector $[x_i, y_i]$ which records the vehicle's positions for x and y coordinates. v_in contains a vector $[v_x, v_y]$ which recorded the vehicle's position for x and y velocity. We are given the first 19 times steps of vehicles' information to predict the next 30 time steps of a corresponding target vehicle. The output matrix would have a similar form of $(mxnxp)$, where $n = 30$.

Other than prediction of the car coordinates, this deep learning model can solve any other tasks that record the position and velocity of objects along time, since this is what our deep learning model is designed for. For example, we can transform the electricity usage over a number of households, and predict the next 30 day dynamic electricity usage using our model. We can also predict fluctuations of the stock market or generate new sentences according to previous records.

2 Exploratory Data Analysis : Perform exploratory data analysis and report your findings with texts/tables/figures. If you include more exploratory analysis beyond the listed questions that provides insights into the data, you will receive bonus points.

2.1 Perform exploratory data analysis and report your findings with texts/tables/figures. If you include more exploratory analysis beyond the listed questions that provides insights into the data, you will receive bonus points.

2.1.1 What is the train/test data size?

Train set dimension is 205942 * 60 * 19 * 2, the test set dimension is 3200 * 60 * 19 * 2

	city	lane	lane_norm	scene_idx	agent_id	car_mask	p_in	v_in	track_id
0	scalar	(144, 3)	(144, 3)	scalar	scalar	(60, 1)	(60, 19, 2)	(60, 19, 2)	(60, 30, 1)
1	scalar	(810, 3)	(810, 3)	scalar	scalar	(60, 1)	(60, 19, 2)	(60, 19, 2)	(60, 30, 1)
2	scalar	(450, 3)	(450, 3)	scalar	scalar	(60, 1)	(60, 19, 2)	(60, 19, 2)	(60, 30, 1)
3	scalar	(252, 3)	(252, 3)	scalar	scalar	(60, 1)	(60, 19, 2)	(60, 19, 2)	(60, 30, 1)
4	scalar	(126, 3)	(126, 3)	scalar	scalar	(60, 1)	(60, 19, 2)	(60, 19, 2)	(60, 30, 1)
5	scalar	(99, 3)	(99, 3)	scalar	scalar	(60, 1)	(60, 19, 2)	(60, 19, 2)	(60, 30, 1)

This is the dimension mock-up of the first sixth scene.

2.1.2 how many dimensions of inputs/outputs in the raw data?

inputs: 60 * 19 * 2
outputs: 60 * 30 * 2

2.1.3 what are the meanings of these input/output dimensions?

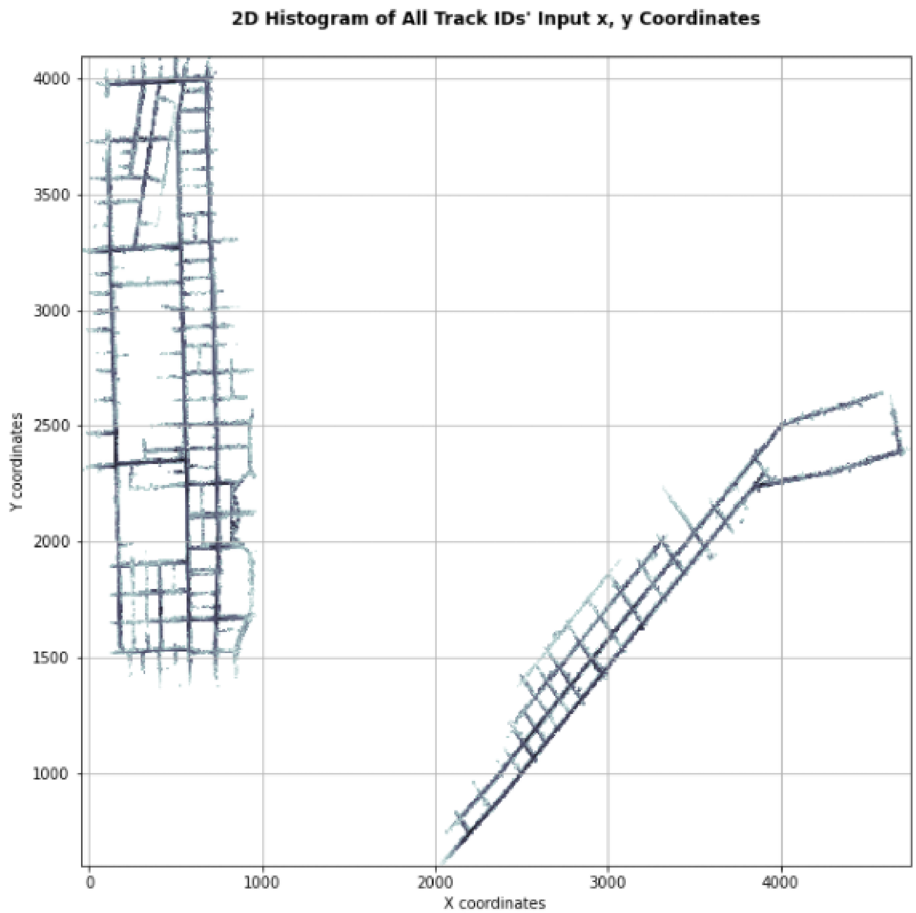
This means that for each scene, there are 60 cars in total (although some of the 60 cars will be empty), and for each of the car in the scene, there will be 19/30 frames of data depending whether it's input or output. Last but not least, for each frame of the cars, there will be 2 inputs in total, x and y coordinates.

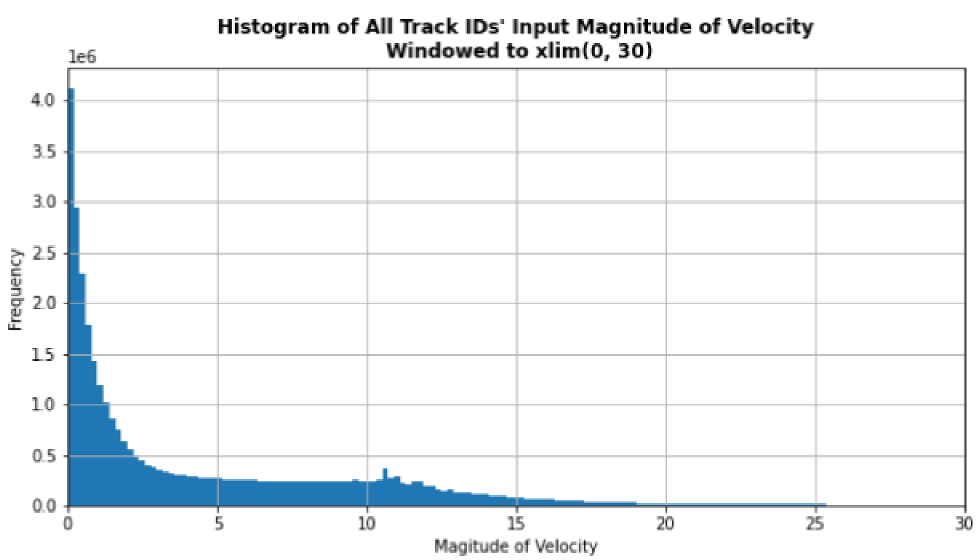
2.1.4 what does one data sample looks like?

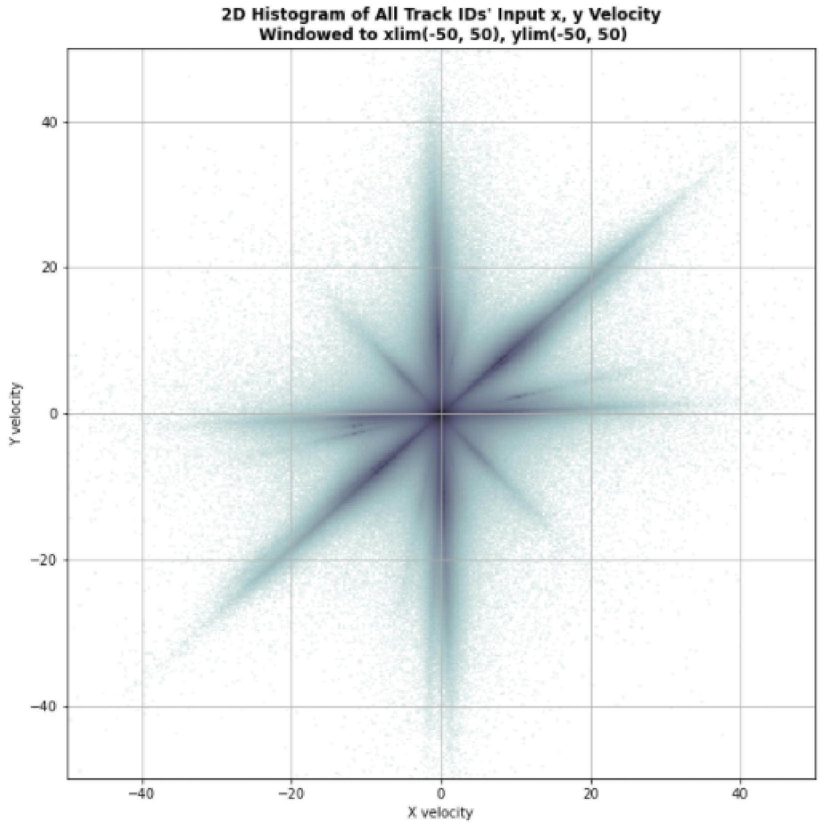

```
{'city': 'MIA',
 'lane': array([[ 559.5046 , 4004.9236 ,    0.     ],
       [ 550.3973 , 4004.625  ,    0.     ],
       [ 541.29004, 4004.3262 ,    0.     ],
       [ 532.1871 , 4003.9185 ,    0.     ],
       [ 523.08466, 4003.4966 ,    0.     ],
       [ 546.20245, 3982.5588 ,    0.     ]........, dtype=float32),
 'lane_norm': array([[-9.10728359e+00, -2.98760027e-01,  0.00000000e+00],
       [-9.10728359e+00, -2.98760027e-01,  0.00000000e+00],
       [-9.10728359e+00, -2.98760027e-01,  0.00000000e+00],
       [-9.10296726e+00, -4.07719672e-01,  0.00000000e+00],
       [-9.10240650e+00, -4.21878159e-01,  0.00000000e+00],
       [-9.10233021e+00, -4.23503131e-01,  0.00000000e+00],
       [-7.13277912e+00,  3.98584557e+00,  0.00000000e+00]......, dtype=float32),
 'scene_idx': 44393,
 'agent_id': '00000000-0000-0000-0000-000000032566',
 'car_mask': array([[1.],
       [1.],
       [1.]......, dtype=float32),
 'p_in': array([[[ 501.16738892, 4017.24853516],
        [ 501.16748047, 4017.24853516],
        [ 501.16751099, 4017.2487793 ],
        ...),
 'v_in': array([[[ 3.13575329e-05,  1.31237772e-04],
        [ 7.79712456e-04, -4.57685994e-04],
        [ 3.82767001e-04,  1.66631851e-03],
        ...),
 'p_out': array([[[ 501.16738892, 4017.24902344],
        [ 501.16744995, 4017.24902344],
        [ 501.16760254, 4017.24926758],
        ...),
 'v_out': array([[[-2.96797277e-03,  1.41289853e-03],
        [ 5.01174422e-04,  8.26028525e-04],
        [ 1.55860023e-03,  3.85732530e-03],
        ...)
 'track_id': array([[['00000000-0000-0000-0000-000000000000'],
        ['00000000-0000-0000-0000-000000000000'],
        ['00000000-0000-0000-0000-000000000000'],
        ...,, dtype=object)}
```

2.2 statistical analysis to understand the properties of the data.

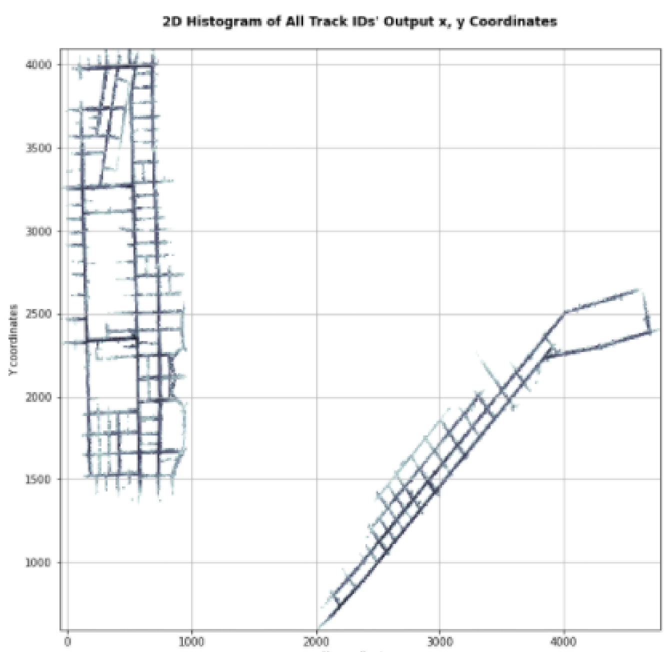
2.2.1 what is the distribution of input positions/velocity (magnitude) for all agents?



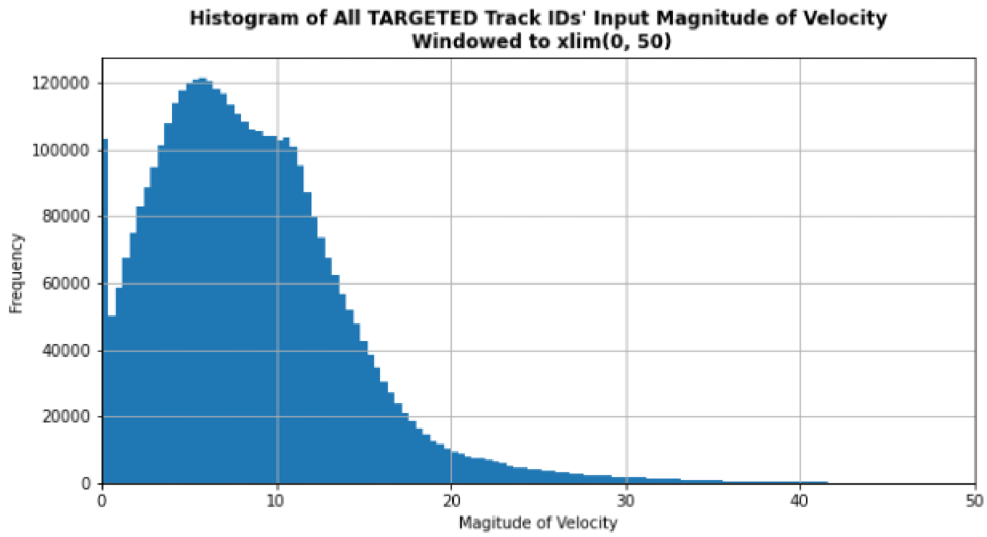


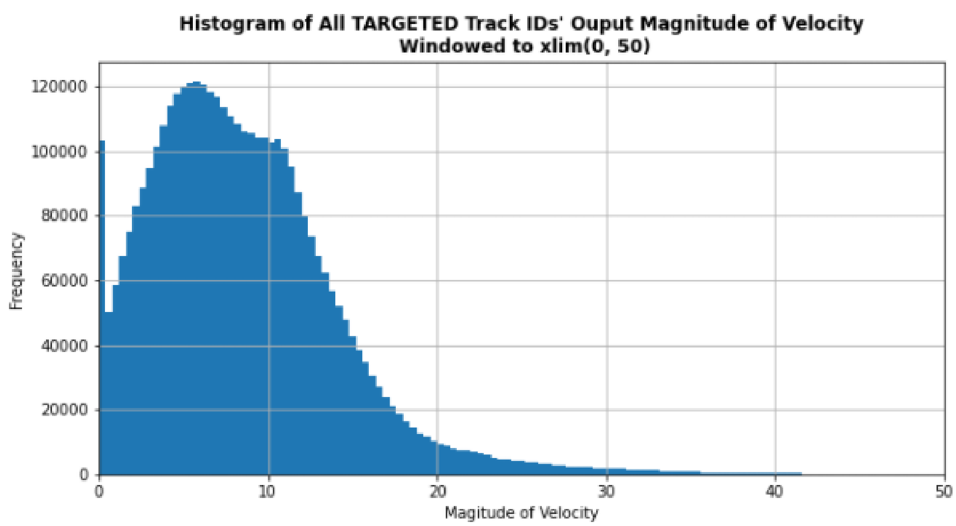


2.2.2 what is the distribution of output positions/velocity (magnitude) for all agents?



2.2.3 what is the distribution of positions/velocity (magnitude) for the target agent?





2.3 Process the data to prepare for the prediction task. Describe the steps that you have taken to process the data. Your description should at least answer the following questions

2.3.1 Did you use any feature engineering? If yes, how did you design your features? Explain your rationale

For our baseline model, in order to increase the model efficiency, we used linear regression and dropped all the car where car-mask =0, This lead to 181217 instances and we are able to split into small batches using dataloader class in pyTorch. According to physics, and our validation, the x, y coordinates along should cover the weight of the velocity, which should be able to estimated by using distance difference over time, so we did not use the velocity variables; Other variables, such as the lane information, were not included in the model (the dimensions for every lane is different) The only variables we used are the x and y locations. However, instead of the normalized location value or the raw location values, we find the difference between the current location and the previous location, resulting in (19 - 1) difference in x values, and (19 - 1) difference in y values; We did not drop car_mask = 0 for the later RNN model because it provides easier dimension manipulation.

2.3.2 How did you normalize your data? Why did you choose this normalization scheme?

We tried normalization of the x and y coordinates by using the mutual mean between the both cities, but after thinking about the overlapping of the features on the map, we decided not to use the mutual mean. Instead, we normalized the coordinates based on which cities the scene is in. However, for the linear model that yields the best result, we took the difference between the current location and the previous location, resulting in $(19 - 1)$ entries in x locations, and $(19 - 1)$ entries in y locations; the difference between the coordinates can also serve as normalization to some degree. We also tried normalizing the data through subtracting the first time point of every vehicle from the entire trajectory. Therefore, every vehicle will have a similar startpoint of $[0, 0]$ when feed into the model.

2.3.3 Did you use the lane information provided in the dataset. If yes, how did you exploit this information.

No, lane information was not included in our model.

3 Deep Learning Model

3.1 Describe the deep learning pipeline for your prediction task and answer the following questions.

3.1.1 What are the input/output that you end up using for prediction after pre-processing?

Similar to our baseline model, we are choosing to only use positional data when training the deep learning model. Our output would also only contain positional information. We merge the car and positional information into one feature vector with the length $60 * 2 = 120$ as input. The input vector would also have temporal information which is 19 for input and 30 for output.

3.1.2 What is your loss function? If you have multiple alternatives, discuss your ideas and observations.

The loss function we utilize is the MSE loss because the problem predicts continuous outcome (not a classification problem) and since trained models have no outlier, MSE can put a larger weight on the difference between the current location and the previous location.

3.1.3 How did you decide on the deep learning model? Explain the rationale given the input/output.

For the baseline model, we put all the features into a huge array, which is learned by a linear model. The simple baseline model that we designed is a linear regression model. We are inspired by the idea of auto regression and teacher-forcing: since we are predicting multiple time points in the future, each prediction will have a relationship with a previous prediction. We decide to create n * 2 Linear Regression models, n = number of time points that need to be predicted. Since we are predicting the next 3 seconds (30 time points), and we have both x and y coordinates to predict, we would create 60 linear regression models.

How we train and test the models are the following: for the x output predictions, we used all the 30 inputs of x coordinates as features, the first output of x coordinates in the ground truth in the train data as the label, and we generate parameters W for the first 19 x inputs to predict the first output in the test dataset. In the training set, we can always access the ground truth output of both x and y, but in the testing set, we do not have the output, thus we will use the prediction of the previous linear regression models to append to the feature matrix to create a new feature matrix with more input.

For testing the performance of different design of linear regression models, we used 5942 scenes as validation out of our 205942 data, and we used the validation set to show that the accuracy dramatically increased when we use the idea of auto regression: if we just predict each output point from the 19 inputs points, the accuracy for the later time points decreased, and the result is much worse than the autoregression result.

While considering which recurrent model would perform best in this task, we first thought about using a traditional RNN model, possibly LSTM model. Since the data is temporal, the model would capture this feature with great flexibility unlike a CNN. However, aftering testing with a basic LSTM model, we believe that a basic LSTM model is not too good at predicting timesteps when the predicted timesteps are longer than the inputted timesteps. We seek out to read more papers about variation of RNN, and we discover that autoencoders/ seq2seq models might be very useful in our task. Specifically, when using the seq2seq model, there are two LSTM models that are linked together with a hidden state. More importantly, the decoders can be trained with the teacher forcing method where existing output labels can sometimes substitute model output, thus increasing the training accuracy of the model. This type of model fits our tasks well, and we eventually decide on using the seq2seq model.

3.2 Describe all the models you have tried to make predictions. You should always start with simple models (such as Linear Regression) and gradually increase the complexity of your model.

3.2.1 Use an itemized list to briefly summarize each of the models, their architecture, and parameters, and provide the correct reference if possible.

1. Close-form Linear Regression
   (a) architecture: Y = w * X + b
      i. Created 60 models, each with different dimensions for X, and w.
   (b) Total number of parameters: 3990
      i. First model have 36 input columns, with a biased variable, total parameter is 37
      ii. Total = (36 + 1) + (37 + 1) + . . . + (36 + 60 + 1)
2. Multi-layer linear model
   (a) Architecture: (input size: n) n = range(38, 98)
      i. Linear: (size: n, n//2)
      ii. ReLu
      iii. Linear (size: n//2, (n//2)//3)
      iv. Linear (size: (n//2)//3, 1)
      v. Sigmoid activation function
   (b) Parameters:
      i. First model, n = 36, num_p = 787
      ii. Last model, n = 60, num_p = 5248
      iii. 161945 in total for all our 60 models
3. LSTM model
   (a) Architecture:
      i. LSTM (4 layers, 520 hidden units, 0.2 dropout rate)
      ii. Dropout (0.2)
      iii. Linear (520 -> 120)
   (b) Parameters: 7899960
   (c) Reference: CSE 151B RNN tutorial
4. Seq2seq model
   (a) Architecture
      i. Encoder
         A. LSTM (3 layers, 520 hidden units, 0.2 dropout)
      ii. Decoder
         A. Adopt hidden state from the last state of encoder model
         B. LSTM (3 layers, 520 hidden units, 0.2 dropout)
         C. Linear (520 hidden units -> 120)

(b) Parameters
   i. First model have 36 input columns, with a biased variable, total parameter is 37
   ii. Total = (36 + 1) + (37 + 1) + . . . + (36 + 60 + 1)
(c) Reference: https://github.com/lkulowski/LSTM_encoder_decoder

3.2.2 If you end up designing your own model architecture, include a picture/sketch of your model architecture. Explain why you choose such a model.

We used linear regression, multiple-linear model, LSTM, encoder-decoder that are common in Deep Learning, therefore, we do not have unique architectures.

3.2.3 Describe different regularization techniques that you have used such as dropout and max-pooling in your model.

1. Dropout layers are utilized within LSTM and between LSTM and linear mapping layer
2. Learning rate decay of 1e-4 is used while training the model

4 Experiment Design. Describe how you set up the training and testing design for deep learning.

4.1 What computational platform/GPU did you use for training/testing?

Schools DSMLP server (datahub), and personal PC setup. We have 1 GPU, 8 CPU, and 16 G RAM. For our Local setup: 1 GPU, 32 Gb RAM

4.2 How did you split your training and validation set?

We perform a train/validation split in our train data size of 80% and 20% because the given validation set has no label. Thus the train set dimensions is 164754 * 60 * 19 * 2, the validation set dimension is 41188 * 60 * 19 * 2, and the test set dimension is 3200 * 60 * 19 * 2 if we account for all the agents 19 in the scenes.After we have a relatively good result, we use all train data as the training set and test data as the validation set.

4.3 What is your optimizer? How did you tune your learning rate, learning rate decay, momentum and other parameters?

We tried Adam, SDG, RMSprop optimizers and Adam works the best. We usually set our learning rate into 0.01 on linear models and 0.001 on RNN models. In order to tune our learning rate, we used a part of the training dataset as a validation set. We also set our learning rate decay as 0.0005. We wrap the ADAM optimizer in a scheduler torch.optim.lr scheduler.StepLR and set the parameter step size to be 3. Moreover, we used different activation functions to see which can obtain the lower loss. We also tried incorporating validation sets into our training, actively tuning the learning rate while training with the validation set.

4.4 How did you make multistep (30 step) prediction for each target agent?

We used an RNN approach. Taking advantage of LSTM, we are able to use the information of the previous layer to further predict the later frame information for 30 timesteps while updating the hidden states H and current state C. Then we add the predicted output to the end of the input strain and select the last input-sized timestamps for the next iteration of the model. This process iterates until a desirable number of outputs are generated (30 in this case) from the model.

Moreover, we also use linear regression/multilayer models through autoregression. The prediction workflows on the test set goes like this: we used the 19 input x, 19 input y, in total 38 features to predict the first x output, using the first linear regression classifier we have made from the training set. Then, we used 20 input x (19 input from the real input, and 1 output from the prediction), 19 input y, in total 39 features to predict the first y output. Then, we used 20 input x, and 20 input y, (40 features) to predict the second x output.

4.5 How many epoch did you use? What is your batch-size? How long does it take to train your model for one epoch (going through the entire training data set once)?

We usually train 20+ epochs if we just take in a small portion of data. For the Multiple linear model, our batch size is 512. As we go through the entire training data set, we train 5 epochs for around 80mins (16min/epochs) for linear models. We know that 5 epochs isn't ideal for obtaining the best accuracy, but due to the time limitations, and the fact that the linear model loss is quite small at the second epoch, given the massive amount of training data/batches, we used 5 (if number of epoch increased to 20, we would need many more hours of training). For RNN models, the basic LSTM model takes around 5 minutes per epoch and the seq2seq model takes around 15 minutes to train one epoch. For RNN models, batch size is around 256. Basic LSTM model takes around 40 epochs to convert while autoencoder takes around 10.

4.6 Explain why you made these design choices. Was it motivated by your past experience? Or was it due to the limitation from your computational platform? You are welcome to use screenshots or provide code snippets to explain your design.

It is because of the limitation from the computational platform. The datahub is unstable and always crashes. Therefore, we can only use the local CPU instead of cuda GPU to train and examine the model.Most of the design choices come from the compromise among time, computational power, and performance. If we increase batch size, even though it would decrease time and increase performance, oftentimes the GPU capacity does not allow. Therefore, we always try to check the maximum batch size we could run, and then run with a large batch size. Regarding epochs, we observe how the training loss decreases, and oftentimes we stop the training when the loss stops decreasing and starts fluctuating.

5 Experiment Results

5.1 Select a few representative models of yours and report the following results by comparing different models.

5.1.1 Use a table to compare the performances of different model designs. What conclusions can you draw from this table.

Model	Linear Regression (Closed form)	Multiple Linear Model	Basic LSTM	Autoencoder
Loss	2.53127	10.06080	880.10048	3.67532

Conclusion: our close form linear regression achieves the best result, this not only shows that a simple model can achieve the same accuracy as fancier, more complicated models, but also the fact that our deep learning model needs better design.

5.1.2 Provide an estimate of training time (in flops or minutes) for different models. What did you do to improve the training speed?

For the baseline model, we eliminated the zeros in the dataset to reduce the number of training objects in order to improve the training speed.

For Multi-layer linear, it took 90 mins to go through the entire training data set and train, and we change the number of layer and activiation function to improve the training speed.

For the RNN models, different models have different training times according to model complexity. Regarding the basic LSTM model, one epoch would take around 300 seconds, and the entire training takes around 40 epochs to reach optimum.

For the seq2seq autoencoder model, one w. Also, we increased the batch size and multithreading number in the model to increase the training speed.

5.1.3 Count and report the number of parameters in different models.

- For the linear regression model, as well as the multi-layer linear models, since we want to be greedy and have as much useful training data at predicting each timestamp, we created 60 models to predict each timestamp (30 for x, 30 for y). Therefore, the total number of parameters we count here is the sum of all the number of parameters for all the models.

- Closed form linear regression parameter count: 3990 (sum of 60 models)

- Multi-layer linear model parameters count: 161945 (sum of 60 models)

  – Explanation: In the multilinear model in Pytorch, we have 3 layers. Let us take the first model for example, when we have 18 x diff points and 18 y diff points, 36 feature columns in total: for the first layer, we have 18 * 38 hidden layer parameters, along with 18 biased parameters, we have 648 + 18 parameters, for the second layer, we have 6 * 18 hidden layer parameters, along with 6 biased parameters, we have 108 + 6 parameters, for the final third layer, we have 1 * 6 hidden layer parameters with 1 biased parameters. In total, we have 648 + 18 + 108 + 6 + 6 + 1 = 787 parameters; This number varis because we used mathematical formula to determine the parameters of hidden layer based on the input features (if we have 55 features instead of 38 features, the number of hidden layer will also vary; the last model we generate, where we have 96 input points to predict the last x and y timestamp, we will have 5457 parameters.)

- LSTM total Parameters: 7899960

- Encoder-decoder total Parameters:

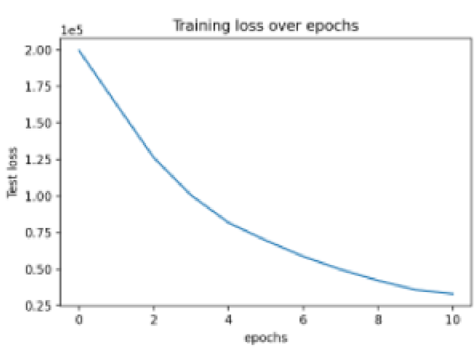
  – Encoder: 5500928

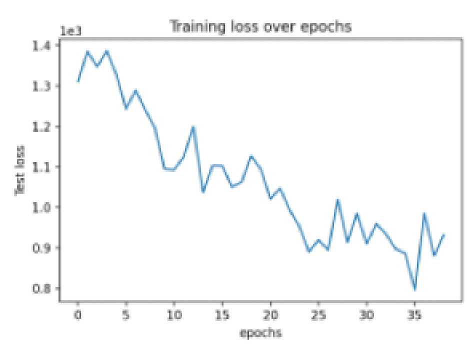
  – Decoder: 5562488

5.2 Play with different designs of your model and experiments and report the following for your best-performing design:

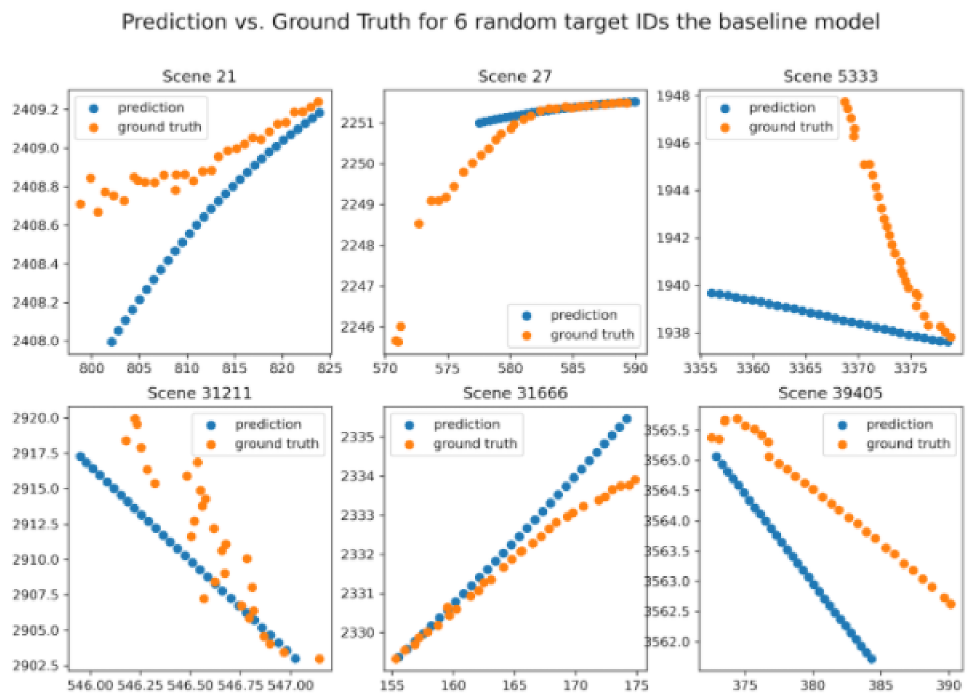
5.2.1 Visualize the training/validation loss (RMSE) value over training steps (You should expect to see an exponential decay).

The first image is linear regression. The second image is LSTM.





5.2.2 Randomly sample a few training samples after the training has finished. Visualize the ground truth and your predictions.



5.2.3 Your current ranking on the leaderboard and your final test RMSE.

Private leaderboard: 21 RMSE: 2.67167 Public leaderboard: 20 RMSE: 2.52455

6 Discussion and Future Work :Analyze the results and identify the lessons/issues that you have learned so far. Briefly answering the following questions

6.1 What do you think is the most effective feature engineering strategy?

We use the linear regression model to evaluate which variables/features are important for the prediction task. It only takes <15 mins to predict 60 columns , while the DL models take much longer than that. After knowing the importance of features, perform data preprocessing and build the deep learning model based on the features. In our model, We used this trick to confirm that difference in location work better than location, and itself is better than location and velocity combined.

6.2 What techniques (data visualization/model design/hyper-parameter tuning) did you find most helpful in improving your score?

We think all techniques(data visualization /modeldesign/ hyper-parameter tuning) are all important. Data visualization allows us to gain a better understanding of the dataset and the predicted result (Geographical visualization).Model design is also very important. We try different models such as muti-layer perceptron, Lstm and autoencoder.After you find a good model, tuning the parameter can increase your accuracy a bit. Overall, the most helpful one should be data visualization. You can not engineer a model and do data processing properly until you really understand your dataset and targeted question.

6.3 What was your biggest bottlenect in this project?

Time-consuming. It takes too much time to train and predict the model, and we will not know the accuracy of model till train though the entire data set. Moreover, familiarity with various machine learning models. All of the members are not too familiar with RNN, and a lot of time has been put into understanding the architecture and debugging dimensional problems.

6.4 How would you advise a deep learning beginner in terms of designing deep learning models for similar prediction tasks.

Data Visualization is Important. Getting to know the data is much more important than the training model. For our model and data, We used the difference between the current location and the previous location to train in our model instead of involving the velocity and lane information.

Keep Everything Simple. Sometimes complex models would make your prediction less interpretable. We try linear regression with careful data preprocessing, and that is still our best outcome so far.

Always Learn from your mistakes. Adjust your time through the mistakes such as high loss and long time processing. Your mistake can improve your model in a good manner

6.5 If you had more resources, what other ideas would you like to explore?

We want to explore the situation, which the car changes direction. Sometimes cars change their direction for the next 30 timestamps but our model failed to predict that situation using the first 19 timestamps.

Try attention based seq2seq model, which allow the decoder to focus on certain timestamp by inspecting the attention distribution.

Improve the Multi-layer linear model design by adding more validations and convergence check (since we are creating 60 models, using same number of epochs for them might not work well)

7 References:

1. Zhao, Ling, et al. "T-gcn: A temporal graph convolutional network for traffic prediction." IEEE Transactions on Intelligent Transportation Systems 21.9 (2019): 3848-3858.
2. Zhao, Zheng, et al. "LSTM network: a deep learning approach for short-term traffic forecast." IET Intelligent Transport Systems 11.2 (2017): 68-75
3. Yu, Bing, Haoteng Yin, and Zhanxing Zhu. "Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting." arXiv preprint arXiv:1709.04875 (2017).
4. Ma, Xiaolei, et al. "Learning traffic as images: a deep convolutional neural network for large-scale transportation network speed prediction." Sensors 17.4 (2017): 818.
5. Cui, Zhiyong, et al. "Deep bidirectional and unidirectional LSTM recurrent neural network for network-wide traffic speed prediction." arXiv preprint arXiv:1801.02143 (2018).

8 GitHub Link

https://github.com/jeffrey7377/151b-project-ml-noob