Autonomous Vehicle Trajectory Prediction

Sirui Tao*
Halıcıoğlu Data Science Institute,
Department of Mathematics
University of California, San Diego
San Diego, CA 92126
s1tao@ucsd.edu

Winston Yu*
Department of Mathematics,
Halıcıoğlu Data Science Institute
University of California, San Diego
San Diego, CA 92126
wiyu@ucsd.edu

ChungEn Pan*
Halıcıoğlu Data Science Institute
University of California, San Diego
San Diego, CA 92126
cepan@ucsd.edu

Akshay Murali*
Halıcıoğlu Data Science Institute
University of California, San Diego
San Diego, CA 92126
a3murali@ucsd.edu

Link to GitHub Repository: https://github.com/cepan/Autonomous_Vehicle_Trajectory_Prediction

1 Task Description and Background

1.1 Task Description & Social Impact

In 2010, car accidents caused more than 36K American deaths and approximately 3M injuries, and total economic loss was estimated to be $871B. [5] Autonomous vehicles (AVs) have the potential to significantly decrease this damage by preventing accidents from occurring in the first place. In order to avoid accidents, an AV must be able to accurately predict the trajectories of other agents, such as pedestrians or other AVs, in the AV's path.

A trajectory is a continuous object, so the data is necessarily sampled at a lower resolution than the underlying phenomenon; for this dataset, the trajectory is sampled at 10 samples per second. As the trajectory is two-dimensional, each point in the training set consists of a sequence of (x, y) coordinates of length 50, representing the observed 5-second trajectory, and a sequence of (x, y) coordinates of length 60, representing the trajectory up to 6 seconds after the first sequence ends. Thus, in more detail, the deep learning task is to predict 60 evenly spaced positions on the trajectory of an autonomous vehicle in the x-y plane 6 seconds into the future (i.e., 10 positions per second) given 50 evenly spaced positions on the trajectory on which the vehicle traveled for the past 5 seconds, such that when fed new data drawn from the same distribution as the training set, minimizes the mean squared error (i.e., when its outputs are compared to the true trajectories via MSE, the model has lower MSE than all other functions of a sufficiently rich yet constrained function space that excludes pathological functions). The data from which the model solving this task learned was collected from 6 separate cities, so there may be patterns that the model should learn and that are specific to each city.

1.2 Related Work

As mentioned, to avoid preventable accident from AV, the notion of accurately predicting the trajectories of agents has been extensively addressed. The baseline approach to this problem is investigated by Boucaud. [1] The problem of using past trajectories to using predict future trajectories can be solved by using a naive LSTM, MLP, or Seq2Seq LSTM, the last of which recursively predicts the position one time-step forward using the previously predicted position. Chandra et al[2].introduced an approach utilizing a combination of spectral graph analysis and deep learning

Preprint. Under review.

which captures the road-agent behavior and the trajectories from the training set and then are fed into a two- stream graph LSTM using embedded dynamic geometric graph. Chandra et al. [6] also proposed another method to predict future trajectories in dense traffic. Contrary to our dataset, which only captures the coordinates of each agent at a given time, Chandra et al. inputs were obtained from RGB cameras, which inherently captures the static and dynamic agent at a given time interval. This allows them to segment noises such as buildings and road trees from the road agents such as buses, cars, scooters, bicycles, or pedestrians. According to Chandra et al., the data was first fed into a combination of a nonlinear motion model and a deep learning based segmentation algorithm to obtain the trajectory. At this stage, the predicted trajectories becomes noisy due to a mixture of road-agents. Then the predicted trajectories were then fed into a LSTM-CNN neural network that differentiated road-agent traffic. Xiao et al. [7] proposed an approach Utilizing UB-LSTM (Unidirectional and Bidirectional LSTM) trajectory model with behavior recognition. First, the surrounding vehicle information is fed into the LSTM to identify vehicles behavior through indicators such as position, velocity, and acceleration. Then UB-LSTM makes trajectory predictions. Using NGSIM data sets, Xiao et al achieved an MSE of 0.000497 when considering vehicle behavior, which is 95.68% lower than the performance without vehicle behavior. However, the dataset that we must use for this task only records each position at a given time in 2D. So another method of solving this kind of problem is introduced by neural ordinary differential equations. [3] The introduced technique by Chen et al. could help accelerated the training of the model and learn more advanced features of agent behavior, such as turning and stopping.

1.3 Task Definition & Possible Application

Formally, the task of the model is to learn a function mapping $\mathbb{R}^{n\times 2}$ to $\mathbb{R}^{m\times 2}$ - in this particular case, $n = 50$ and $m = 60$ - such that when fed new data drawn from the same distribution as the training set, minimizes the mean squared error. That is, when its outputs are compared to the true trajectories via MSE, the model has lower MSE than all other functions of a sufficiently rich yet constrained function space that excludes pathological functions.

Written concisely, we want to find f^* such that

$$f^* = argmin_{f\in\mathcal{H}}\mathbb{E}_{(x,y)\sim T}[|f(x) - y| \cdot t^{-1}]$$

where T is the distribution of trajectories from which the data was drawn, $t = 2(n+m)$, $f : \mathbb{R}^{n\times 2} \rightarrow \mathbb{R}^{m\times 2}$, $\mathcal{H}$ is a set of "nice" functions, $x \in \mathbb{R}^{n\times 2}$, and $y \in \mathbb{R}^{m\times 2}$.

The input is a matrix of real values of 50 rows and 2 columns, and the output is a matrix of real values of 60 rows and 2 columns; in both matrices, the number of rows represents the number of time steps, and the number of columns represents the number of dimensions. The rows should be ordered in forward temporal order. The prediction task was to learn a function from the training set that would minimize the mean squared error on inputs drawn from the same distribution that generated the training set.

From this mathematical abstraction, we see that the benefits of solving the task of trajectory prediction need not be restricted to autonomous cars. For example, a missile that can more accurately predict the trajectory of its target will be more effective, a missile interceptor that can more accurately predict the trajectory of a missile will be more effective (especially as hypersonic glide vehicles are expected to be able to execute changes in direction even more sudden than cruise missiles can), and systems that reduce the error of asteroid trajectory predictions will either grant humanity more peace of mind or more time to prepare for impact. In general, the mathematical formulation suggests applications beyond mere physical objects: the data is in the form of a time series, so solving the problem of trajectory prediction may contribute to predicting n-dimensional time series, especially when the values of the coordinates of each data point may be interrelated. For example, progress in trajectory prediction may contribute to predicting economic indicators such as gas prices and business sentiment and consumer sentiment, or predicting climate indicators, such as rainfall and cloud cover.

2 Exploratory Data Analysis

2.1 Dataset Description

These are the sizes of the data collected from each city, as well as the dimensionality of each data point.

- Austin
  - Train: 43041 examples (50 timestamps, (x, y) coordinates)
  - Test: 6325 examples (60 timestamps, (x, y) coordinates)
- Miami
  - Train: 55029 examples (50 timestamps, (x, y) coordinates)
  - Test: 7971 examples (60 timestamps, (x, y) coordinates)
- Pittsburgh
  - Train: 43544 examples (50 timestamps, (x, y) coordinates)
  - Test: 6361 examples (60 timestamps, (x, y) coordinates)
- Dearborn
  - Train: 24465 examples (50 timestamps, (x, y) coordinates)
  - Test: 3829 examples (60 timestamps, (x, y) coordinates)
- Washington DC
  - Train: 25744 examples (50 timestamps, (x, y) coordinates)
  - Test: 3829 examples (60 timestamps, (x, y) coordinates)
- Palo-Alto
  - Train: 11993 examples (50 timestamps, (x, y) coordinates)
  - Test: 1686 examples (60 timestamps, (x, y) coordinates)

Below are some trajectories sampled from each city (Figure 1).

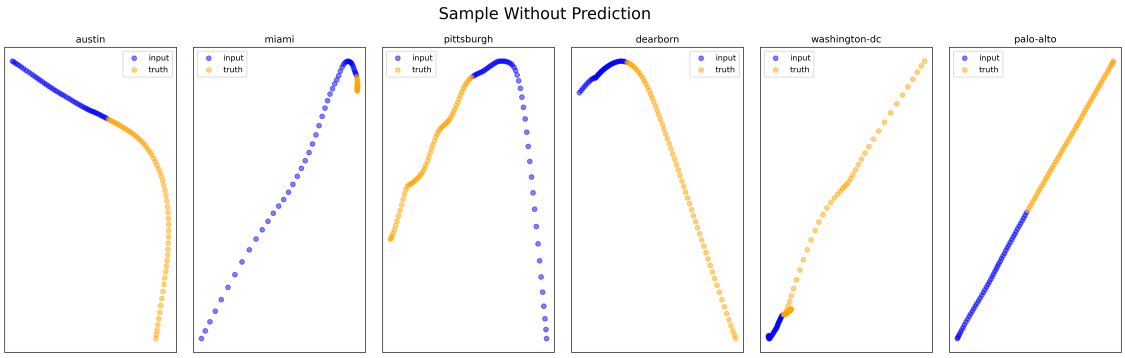

Figure 1: Sample Model Prediction by single-layer seq2seq LSTM with attention

2.2 Statistical Analysis

We started by plotting x-y value distribution from all 6 cities to exam the difference in distributions (Figure 2). As we can see, data from some cities seem to be concentrated in the city center, while data from other cities seem to be more evenly distributed.

Then, we used the heatmap to illustrate the distribution of input position in each city and combined (Figure 3).

Following the exact same procedures, we plotted the figures for the output data (Figure 4).

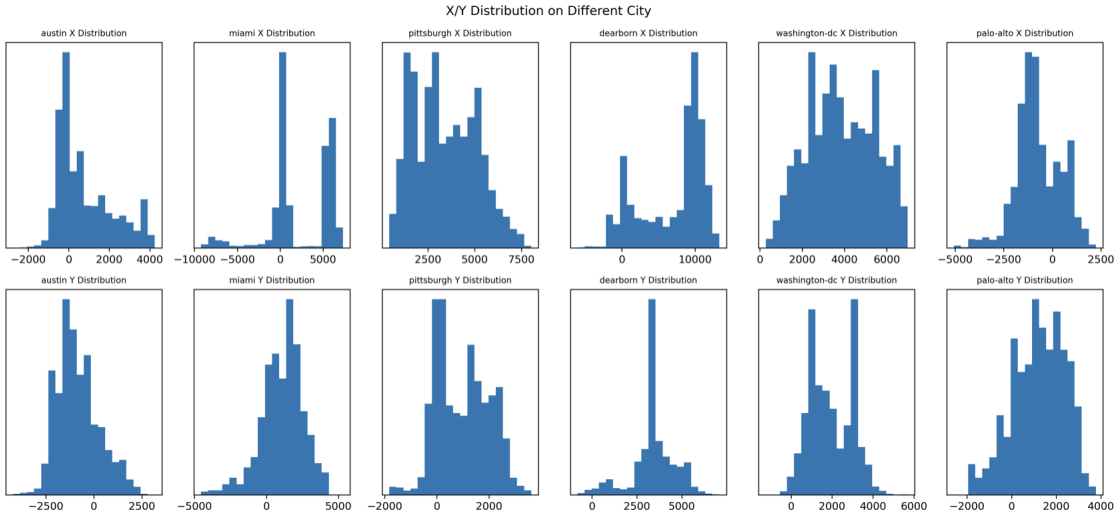

Figure 2: X/Y Distribution for 6 cities

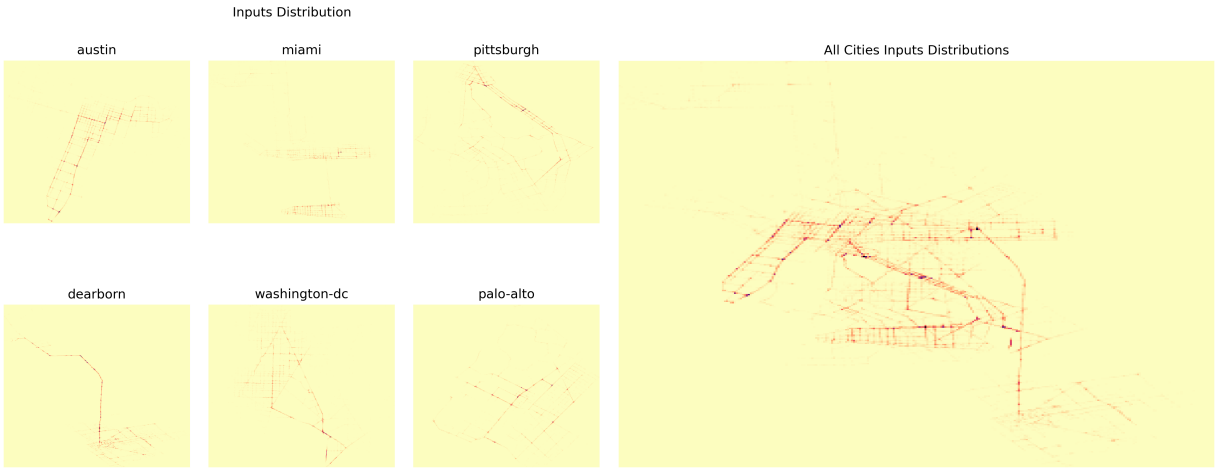

Figure 3: Input Distribution for 6 cities

We plot the distribution of input and output combined for the training set (Figure 5). It is clear that the pattern in the heatmap corresponds to a map of the city, which makes sense considering the prediction task we are given. Later, we learned that centering all trajectories to start from $(0, 0)$ can help reduce model complexity and help models perform better. So we also plotted the distribution of centered input trajectories in the following figure (Figure 5). The transformed data gives a easier representation to learn for the model. We can also observe that each city has distinct centered trajectories.

Finally, we did the same for velocity (Figure 6). This indicates that rotation might also help improve model accuracy, but we were not able to implement that into our deep learning pipeline due to a lack of time, which is quite a shame as other teams demonstrate that this is quite an useful feature.

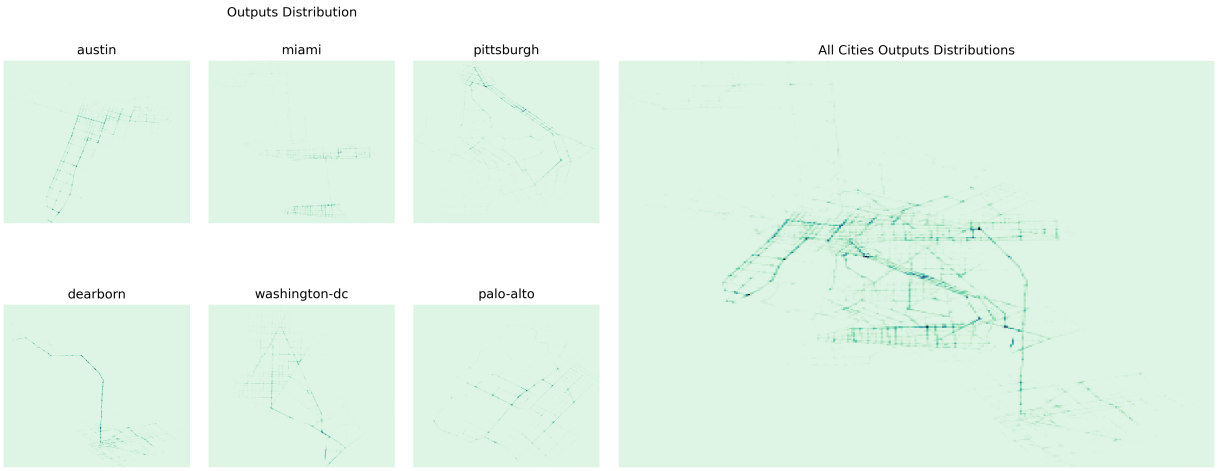

Figure 4: Output Distribution for 6 cities

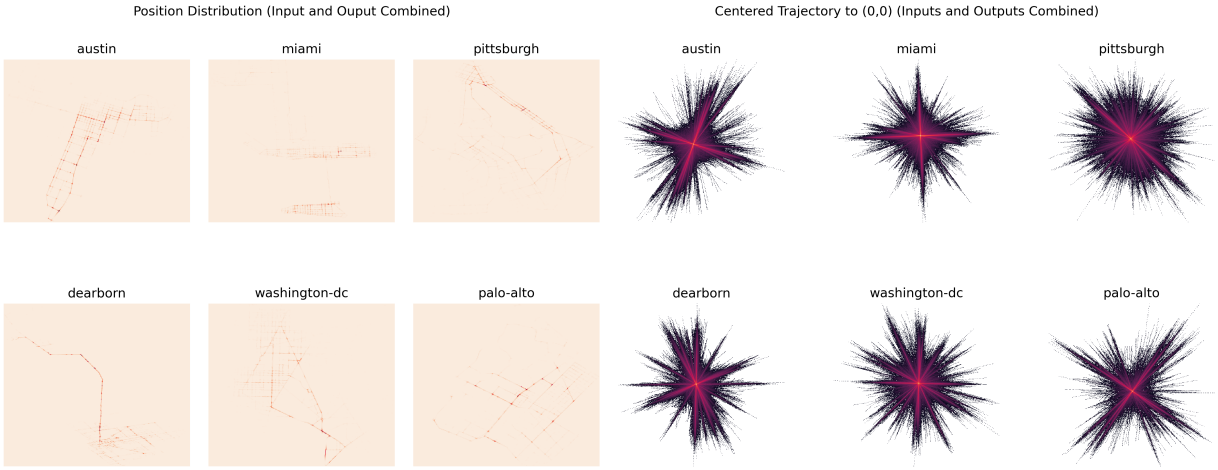

Figure 5: Position Distribution

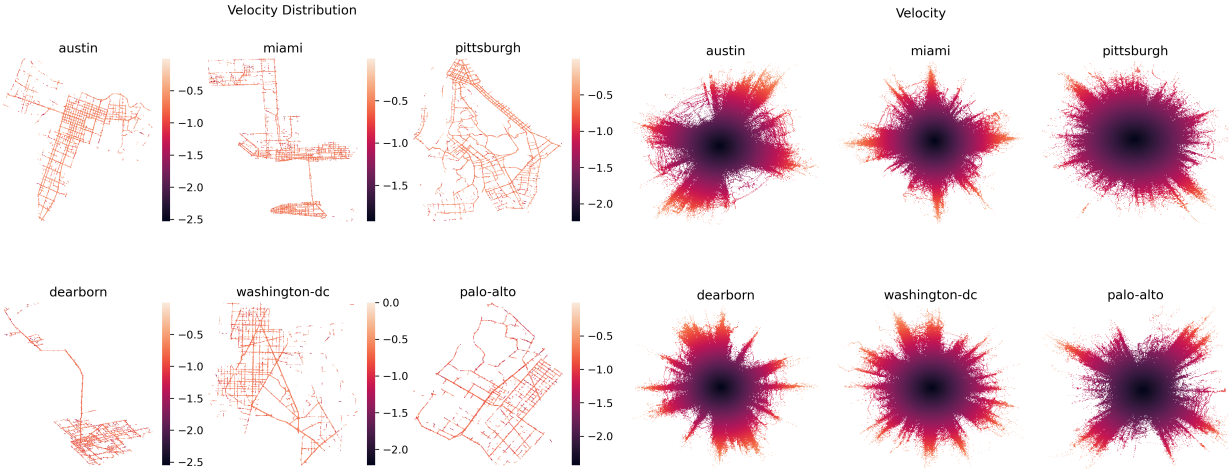

Figure 6: Velocity Distribution

2.3 Data Preprocessing

In the data pre-processing stage, we split the training dataset into an actual training set and a validation set. Before the split, we set a seed and shuffled the data. The size of train/validation dataset is just 80 and 20% of their respective size of their original data size listed in section 2.1.

Some feature engineering tricks we have used include:

We translate trajectories to start from $(0, 0)$ because we believe this will reduce the data complexity and better enable the model to efficiently learn the meaningful underlying structure of the data.

Separate the future 60 trajectories points into 1 initial position and 59 following displacement to enforce the model to learn certain features of the dataset by training one model to just predict initial position of our future prediction and one for predicting future displacements in each time step.

We normalize our data by first centering them to $(0, 0)$. we believe this normalization is quite effective as it helps prevent our model from learning to overfit to some non-meaningful features of the data such as where the trajectory starts. Later, we also want to use the shift information we used to move the trajectory onto $(0, 0)$ as a feature in a fully-connect output layer, but we are not able to implement this due to time constraint.

Lastly, we mainly used the city information to train one model for each cities so the unique city information can be learnt by the model. Our rational is, since each cities' data from the heatmaps look so different, we should try to use them by training individual model for each city.

3 Models and Experiment Design

3.1 Input & Output Features for MLP

We picked MLP as our simple machine learning model. For this model, the input is 50 (x, y) positions of the input trajectory flattened into a vector in $\mathbb{R}^{100}$, and the 60 (x, y) positions of the output trajectory were flattened into a vector of $\mathbb{R}^{120}$. We used the MSE loss when training this model.

3.2 Deep Learning Pipeline

The input features that we used were the 50 positions of the input trajectory as well as the 49 differences between consecutive positions, i.e. the average velocity between consecutive positions over a timespan of 0.1 seconds.

We roughly based our model architecture on a Seq2Seq LSTM. The model is comprised of two Seq2Seq neural networks with LSTM cells. The first model maps the input positions to the first output position – i.e., it maps $\mathbb{R}^{50\times 2}$ to $\mathbb{R}^{1\times 2}$. The second model maps the input velocities to the output velocities – i.e., it maps $\mathbb{R}^{49\times 2}$ to $\mathbb{R}^{59\times 2}$. The prediction is generated by stacking the output of the first model on top of the output of the second model, which forms a matrix in $\mathbb{R}^{60\times 2}$ and then taking the cumulative sum across columns. The loss function used was simply 'torch.nn.MSELoss', as instructed.

Observations:

1. We conjectured that a sequential model based on an RNN or a LSTM would outperform a MLP because the data is composed of time series. We confirmed this empirically by feeding untransformed data to the MLP and the RNN models and seeing that the RNN model outperformed the MLP.
2. It seems that there is an optimal level of complexity for multi-layer perceptrons and for Seq2Seq LSTMs, because we observe empirically that the mean squared loss for the Seq2Seq LSTM (without attention layers) is lower when the LSTM portion of the model is composed of two layers, as opposed to one or three layers, and that the mean squared loss for the multi-layer perceptron is minimized when there are five layers, as opposed to there being three or seven layers.
3. We also observe that without an attention layer, incorporating bidirectionality in the LSTM improves performance (we wanted to use a bidirectional LSTM with attention, but we

encountered dimensionality problems). We speculate that bidirectionality improves performance because the data is also bidirectional in some sense - more specifically, a car is approximately as likely to travel in one direction on a road as it is to travel in the other direction, with some obvious exceptions such as one-way roads.

4. We observed that translating the trajectories to begin at the origin improves performance, which we attribute to the data having locality properties; that is, a car does not take into account its starting point when deciding where to proceed after the input trajectory has been traversed. Although we do not believe this assumption always holds (since, for example, there are some places where cars are more likely to turn than at other places, even if their trajectories up to the turning point are identical), we still believe that the assumption is usually true. In addition, centering the trajectories should also prevent overfitting because there may be very few trajectories that start at a given point, and so centering trajectories prevents the model from merely memorizing the initial position and outputting the output trajectory of that training example.

5. The reason that we added an attention layer to our decoder is that we believed that each LSTM cell may need assistance in learning the importance of each previous position to the position that the LSTM is supposed to predict.

6. After the competition had ended, we also trained a simpler RNN model in order to see if reducing model complexity (from LSTM to RNN) could improve test results. We observed a small decrease in MSE and speculate that the improvement is due to a) the role of the attention layer being able to be performed by the parameters of the RNN b) the RNN reducing overfitting due to it having fewer parameters than an LSTM with attention layers.

3.3 Model Summary

3.3.1 MLP

The 3-layer small MLP has an encoder that uses linear transformations to map $\mathbb{R}^{100}$ to $\mathbb{R}^{64}$ to $\mathbb{R}^{32}$ to $\mathbb{R}^{32}$ and a decoder mapping $\mathbb{R}^{32}$ to $\mathbb{R}^{64}$ to $\mathbb{R}^{120}$ to $\mathbb{R}^{120}$.

The 3-layer wide MLP has an encoder that uses linear transformations to map $\mathbb{R}^{100}$ to $\mathbb{R}^{256}$ to $\mathbb{R}^{64}$ to $\mathbb{R}^{64}$ and a decoder mapping $\mathbb{R}^{64}$ to $\mathbb{R}^{64}$ to $\mathbb{R}^{256}$ to $\mathbb{R}^{120}$.

The 5-layer wide MLP has an encoder that uses linear transformations to map $\mathbb{R}^{100}$ to $\mathbb{R}^{512}$ to $\mathbb{R}^{256}$ to $\mathbb{R}^{256}$ to $\mathbb{R}^{256}$ to $\mathbb{R}^{128}$ to $\mathbb{R}^{64}$ and a decoder mapping $\mathbb{R}^{64}$ to $\mathbb{R}^{128}$ to $\mathbb{R}^{256}$ to $\mathbb{R}^{256}$ to $\mathbb{R}^{512}$ to $\mathbb{R}^{120}$.

The 7-layer wide MLP has an encoder that uses linear transformations to map $\mathbb{R}^{100}$ to $\mathbb{R}^{512}$ to $\mathbb{R}^{256}$ to $\mathbb{R}^{256}$ to $\mathbb{R}^{128}$ to $\mathbb{R}^{64}$ and a decoder mapping $\mathbb{R}^{64}$ to $\mathbb{R}^{128}$ to $\mathbb{R}^{256}$ to $\mathbb{R}^{512}$ to $\mathbb{R}^{120}$.

Between every linear transformation of the MLPs, there is a ReLU layer.

3.3.2 LSTM (Encoder/Decoder)

From the data provided, we obtained the coordinates of each agent trajectory sampled at 10 Hz. Using the information above, we can derive the change in position for $1 \leq t \leq 59$, where $t \in \mathbb{N}$, between each pair of consecutive coordinates. More explicitly, we engineered the features $(x_t - x_{t-1}, y_t - y_{t-1})$. We fed these as inputs to the second neural network that was intended to predict the output velocities.

The code upon which we based our code was written by Post-Doctoral Fellow Laura Kulowski at Harvard Department of Earth and Planetary Sciences [4].

3.3.3 Regularization Methods

For each LSTM model except for the final model, we used dropout with rate 0.3. We also trained each model with early stopping by stopping training as soon as validation loss increased a certain amount.

4 Results

4.1 Training & Testing Design

In order to obtain these results, we initially used DataHub's GPU, an NVIDIA 1080Ti. The DataHub setup has 1 GPU, 8CPU, and 16GB of RAM. We then moved to utilizing SDSC's Expanse Supercomputer, which hosts a NVIDIA Tesla v100-sxm2 GPU, to reduce our models' training times.

The optimizer we used in all our MLP models and Seq2Seq LSTM models trained without attention was Adam due to its adaptability in optimizing non-convex loss functions, its popular usage to train neural networks, and because of the fact that it tracks loss on a global level, rather than simply looking at the loss corresponding to the previous few epochs. For training our Seq2Seq LSTM model with attention and our Seq2Seq RNN model, we used RMSProp as our optimizer simply for the reason that it is faster to train with (reduces the training time) due to its lack of implementation of momentum and the fact that it only looks at loss at a local scale (previous few epochs), compared to Adam, which looks at overall loss over time/through epochs.

We experimented with several initial learning rates and finally settled on using $2 * 10^{-4}$ based on the rate of slope change in the training/ validation loss graph in our MLP and initial Seq2Seq LSTM models (without attention). For our Seq2Seq model with attention and Seq2Seq RNN model, on the other hand, we implemented a learning rate scheduler that decreases linearly from $1 * 10^{-2}$ to $1 * 10^{-10}$ based on if the training loss plateaus (changes by less than 0.001) in the past five epochs.

For our MLP models, we were able to make multistep (60 step) prediction by flattening both input and output into sizes of 100 and 120 and using a encoder-decoder architecture to allow mapping from 100 to 120 via several intermediate layers easily.

The resulting trained models make multi-step predictions via a simple encoder-decoder architecture using multiple fully-connected layers. If a model was named n-layer multi-layer perceptron, then the encoder and decoder each have n linear transformations with conformable dimensions, and neighboring linear transformations are connected by a ReLU layer, except for the linear transformation connecting the encoder and decoder.

In our Seq2Seq models, we made multistep predictions by:

1) feeding the known input trajectory to the encoder, which outputted a hidden state 2) feeding that hidden state and the last position of the input trajectory to the decoder

3) repeating step 2), but the hidden state was generated by the previous decoder cell and the last position of the input trajectory is replaced by the prediction of the previous decoder cell.

In training our Seq2Seq models, since we used mixed teacher-forcing, we chose a probability of 0.6 to feed the correct/known output trajectory position to a decoder cell instead of the previous cell's prediction. We also used dynamic teacher-forcing, which means that every epoch, the probability of using the correct position instead of the prediction was lowered by 0.02.

For training all of our models, we used the city information to train separate models for each city; however, we did not incorporate city information into our feature embedding for encoding into our training process.

The batch size we used eventually for our models is 128 after testing a series of value, ranging from 4 - 512 and picked one that seems to decrease loss fastest based on our train/validation loss graph.

We used 50 epochs for smaller models as it seems the train and validation converge well in the end. For later more complex MLP models, we set a max training epoch of 120 and a early stop criteria of validation loss not decreasing for 25 epoch. We saved the model with best validation loss before early stop. For more complex Seq2Seq models, we trained position models for 1000 epochs and velocity models for 150 epochs with early stopping due to long training times and the fact that loss converged by the time the full number of epochs were completed.

Each epoch in our MLP models takes less than a second to run. The total training time for 6 roughly-50-epoch city model is around 5 minutes for 3 layer model, 10.5 minutes for 5 layer model, and 11.5 minutes for 7 layer model. For our Seq2Seq models, each epoch took a little more than one second to run, resulting in overall training times of several hours.

Because we wanted to start with a simple model that gave us a reasonable baseline with a relatively small training time, we started with this multi-layer encoder/decoder architecture and then progressed to more complicated models that began to capture the sequential nature of the data (motivation for implementing LSTM/RNN-based neural network architectures).

4.2 Model Results

Table 1: Models Statistics Summary

Model	Kaggle Score	Training Time (mins.)	# of Parameters
3 Layer Small MLP	∼750.2	∼5	34032
3 Layer Wide MLP	∼659.3	∼5	98104
5 Layer Wide MLP	∼587.7	∼9	590264
7 Layer Wide MLP	∼110962.8	∼11	631608
1-Layer Seq2Seq LSTM	∼67.1	∼177	2249988
2-Layer Seq2Seq LSTM	∼50.1	∼398	2587172
3-Layer Seq2Seq LSTM	∼62.1	∼603	2806404
1-Layer Seq2Seq Bidirectional LSTM	∼42.9	∼387	4499972
1-Layer Seq2Seq LSTM after translation	∼35.4	∼154	3672497
1-Layer Seq2Seq LSTM with Attention	∼34.2	∼455	3672497
1-Layer Seq2Seq RNN (After competition)	∼33.3	∼106	563460

From the table above, it is apparent that greater model complexity generally improved results, up to a certain point. As model complexity continued to increase past this point, overfitting became a strong factor that acted as a detriment to model performance, hindering its ability to generalize to the test set. The reason that we suspect overfitting to appear as as model complexity increased is that in the case of the 7-layer MLP model, loss on the training set was still relatively low and comparable to training loss on the 5 and 3-layer MLP models; however, upon submitting to Kaggle, we found that the test MSE loss was much larger, suggesting that the model overfit to the training data. A similar case of suspected overfitting was also present in comparing MSE loss on the test set for our 3-Layer Seq2Seq LSTM, compared to our 1-Layer and 2-Layer Seq2Seq LSTM models trained without attention, and our 1-Layer Seq2Seq LSTM models compared to our 1-Layer Seq2Seq RNN model.

In order to improve training speed of our models, especially our Seq2Seq models, we investigated how manually reducing the number of epochs for which we trained our model affected model performance. From our investigation, we found that there was little significant difference in model performance as we continually reduced the number of epochs to a certain point where the losses converged. As a result, we implemented early stopping to our model, allowing for training to end early based on tracking of validation loss over time. Another approach for decreasing our training time was upgrading the hardware on which we trained our model. Instead of using DataHub, on the last day we transferred our training workload to run on the Expanse Supercomputer from SDSC, using the NVIDIA Tesla v100-sxm2 GPU. Using this machine, we were able to cut our training times significantly, which is reflected in the table above.

4.3 Best Performing Model

We observe an exponential decay for all the models we trained. Here is a graph of the training and validation loss for our best performing model - **1-layer seq2seq LSTM with attention** (Figure 7).

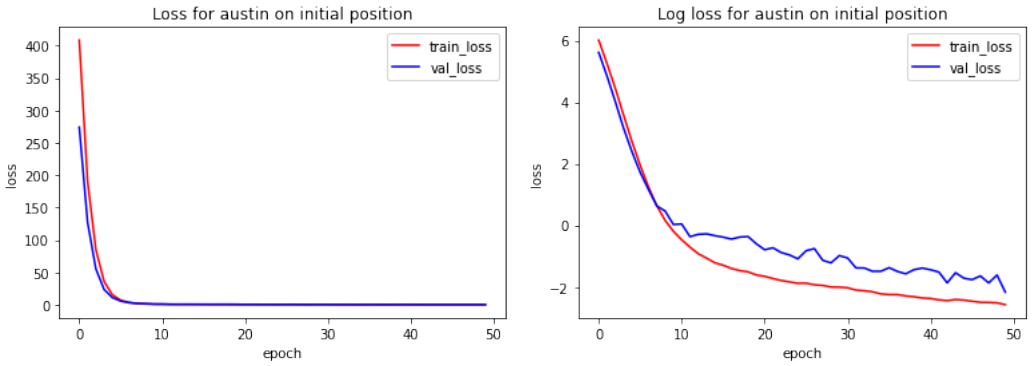

Figure 7: Training and validation loss of 1-layer seq2seq LSTM with attention for Austin

However, as you can see from the plot below (Figure 8), our single-layer seq2seq LSTM is not performing well when trying to predict the trajectory.

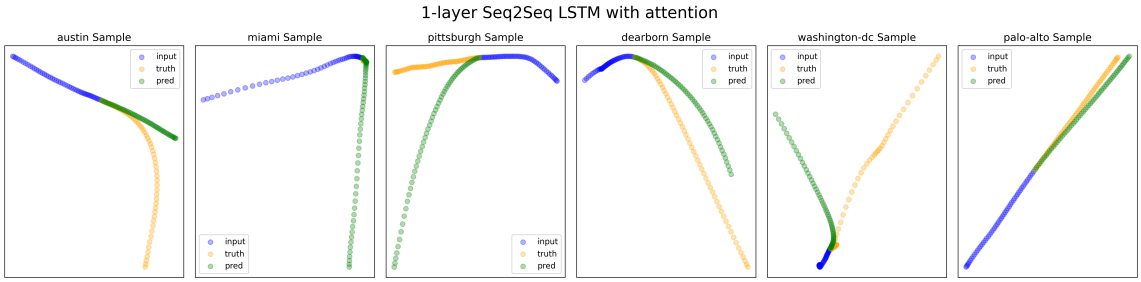

Figure 8: Sample Model Prediction by single-layer seq2seq LSTM with attention

Our final Kaggle private leaderboard ranking is 23rd with final test MSE of 34.04 (Figure 9).

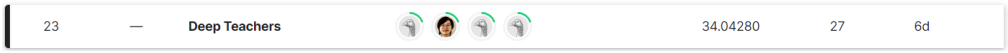

Figure 9: Final Kaggle Ranking

5 Future Work

5.1 Result analysis & Lessons learned

Comparing the models we have implemented so far, we made the following conclusions.

The most effective feature engineering strategy:

1. Translating trajectories to start from $(0, 0)$ significantly reduces the data's complexity and we can use a way smaller model to achieve similar performance. Using a model with the same parameter dimension, its performance increased significantly because it can more easily learn the meaningful structures in our dataset.

List of techniques most helpful in improving our ranking:

1. Sampled prediction visualization is extremely helpful as it clearly shows our model's deficiency
2. Shuffle the data before training with a seed to ensure both better efficiency and reproducibility
3. Method such as ADAM help model converge faster and in a more stable manner
4. Using validation to set both early stop and learn rate scheduler help
5. Using more advanced model such as seq2seq LSTM to capture sequential nature of the data help improve model performance
6. Using dropout and teacher forcing help improve model performance
7. Varying number of layers and set it to be bidirectional also help

List of major bottlenecks:

1. The training time for more complex models is quite long
2. Implementation speed is slower than expected such that we don't have enough time to test all features and models we planned
3. Domain expertise in deep learning constrained us from testing more advanced model

List of advice for deep learning beginners in terms of designing deep learning models for similar prediction tasks:

1. Start with very simple models
2. Monitor training loss as the model trains because it may save you significant amounts of time when you are training more complex models later
3. Plot your prediction against the ground truth to get intuition on how to tweak your model
4. Do random train/validation split (or, k-fold cross-validation) to prevent overfitting
5. Read literature to learn about baseline models and state-of-the-art techniques

List of ideas we would like to explore if we had more resources and time:

1. Verify if rotating the trajectories can decrease data complexity and improve accuracy
2. Verify if using city as a embedded feature and training a large model instead 6 individual ones help
3. Verify if adding explicit velocity and acceleration embeddings in the feature vector help
4. Verify if adding implicit embedding of previous points in the feature vector help
5. Verify if adding starting position we removed in the data prepossessing stage into feature space help
6. Verify if using data augmentation will help, both via addressing under-represented data and adding noise to the dataset
7. Test if bidirectional LSTM can help improve our model as suggested in the literature
8. Test if neural ODE model can learn turning behavior more precisely and efficiently

References

[1] Laurent Boucaud. *Elbuco1/attentionmechanismstrajectoryprediction: In this repository, one can find the code for my master's thesis project. the main goal of the project was to study and improve attention mechanisms for trajectory prediction of moving agents.* URL: https://github.com/elbuco1/AttentionMechanismsTrajectoryPrediction.

[2] Rohan Chandra et al. "Forecasting Trajectory and Behavior of Road-Agents Using Spectral Clustering in Graph-LSTMs". In: *IEEE Robotics and Automation Letters* 5.3 (2020), pp. 4882–4890. DOI: 10.1109/LRA.2020.3004794.

[3] Ricky T. Q. Chen et al. "Neural Ordinary Differential Equations". In: *Advances in Neural Information Processing Systems* (2018).

[4] Laura Kulowski. *Lkulowski/lstm_encoder_decoder: Build a LSTM encoder-decoder using pytorch to make sequence-to-sequence prediction for time series data*. URL: https://github.com/lkulowski/LSTM_encoder_decoder.

[5] Miller Kory Rowe LLP. *Motor Vehicle Crashes Cost the U.S. Nearly $1 Trillion/Year*. URL: https://www.mkrfirm.com/blog/motorcycle-accidents/motor-vehicle-crashes-cost-u-s-nearly-1-trillionyear/.

[6] Chandra Rohan et al. "RobustTP: End-to-End Trajectory Prediction for Heterogeneous Road-Agents in Dense Traffic with Noisy Sensor Inputs". In: *CSCS '19: ACM Computer Science in Cars Symposium* 2 (2019), pp. 1–9. DOI: https://doi.org/10.1145/3359999.3360495.

[7] Haipeng Xiao et al. "UB-LSTM: A trajectory prediction method combined with vehicle behavior recognition". In: *Journal of Advanced Transportation* 2020 (2020), pp. 1–12. DOI: 10.1155/2020/8859689.