Argoverse'd :

Motion Forecasting with Deep Learning Trained on a New Dataset

Hardy Bright, Robyn Logan, Shane Parr, Ye Wang

Devpost link: https://devpost.com/software/vadimaniacs-heresy#updates
Github link: https://github.com/Robyn42/Vadimaniac-Code

Introduction

Motion forecasting is a multivariate time-series regression task in which the goal is to predict the future trajectories of moving objects. By analyzing the trajectory of objects in labeled data, algorithms can be trained to accurately predict the future trajectory of new objects based on their current positions and headings. This allows for autonomous vehicles to be proactive in their object avoidance instead of older reactive technologies used in driver-assist technologies.

We are training and assessing the robustness of deep learning models on the Argoverse 2 motion prediction dataset[1], which consists of vehicles driving on public roads in six US cities (Austin, Detroit, Miami, Pittsburgh, Palo Alto, and Washington D.C.) to provide a robust set of data from across the country. For the purpose of motion forecasting, the data was processed into 2-dimensional maps to simplify the training and implementation of motion forecasting algorithms. The data was also labeled to allow the algorithms to differentiate objects as various vehicles and beings to further improve forecasting accuracy.

Our team has developed multiple model architectures to date including stacked long short-term memory (LSTM), multi-module gated recurrent units (GRU) in an observe/predict configuration, single module GRU, and multi-layer perceptron (MLP). An MLP was selected to serve as a good baseline with parallel structure that might align with the timeseries data, the GRUs were selected because of their improved speed and efficiency when compared to LSTMs, and the stacked LSTM was selected as a counterpoint to the "stacked" observation/prediction model. The models trained in this paper have been shown to perform better than models on Argoverse 1's public leaderboards in average displacement error (ADE) metrics and comparably in others.

Methodology

The Dataset

Argoverse 2 is a multimodal, multivariate dataset produced by Argo AI[2]. It builds upon the foundation of Argoverse 1 by having four major components: sensor, lidar, motion forecasting and map change datasets were compiled to be the state–of-the-art for autonomous vehicle development. Our project makes use of the motion forecasting part of the dataset, which is portioned into train, validation and test datasets together totaling over 200 Gigabytes (GB). Over the course of development for ouAr various models, we used each of the three for cross-validation. Our final training and evaluation was done with the train and validation set respectfully.

```
observed: bool
track_id: string
object_type: string
object_category: int64
timestep: int64
position_x: double
position_y: double
heading: double
velocity_x: double
velocity_y: double
scenario_id: string
start_timestamp: double
end_timestamp: double
num_timestamps: int64
focal_track_id: string
city: string
----
observed: [[true,true,true,true,true,true,true,true,true,true,...,false,false,false,false,false,false,false,false,false,false]]
track_id: [["77543","77543","77543","77543","77543","77543","77543","77543","77543","77543",...,"AV","AV","AV","AV","AV","AV","AV","AV","AV","AV"]]
object_type: [["vehicle","vehicle","vehicle","vehicle","vehicle","vehicle","vehicle","vehicle","vehicle","vehicle",...,"vehicle","vehicle","vehicle",
object_category: [[1,1,1,1,1,1,1,1,1,1,...,1,1,1,1,1,1,1,1,1,1]]
timestep: [[0,1,2,3,4,5,6,7,8,9,...,100,101,102,103,104,105,106,107,108,109]]
position_x: [[-11.823673649007935,-12.39320578754739,-13.080372362384209,-13.880497176060121,-14.773626809536655,-15.74815042844231,-16.7757018228538
position_y: [[-567.4023974854001,-567.2253156611156,-567.0096938093519,-566.7544103059896,-566.4653864371573,-566.1455193078166,-565.8032455480408,-5
heading: [[2.8502121782024448,2.8500757486227837,2.849729679583497,2.8490452955231054,2.8479616883793915,2.846542156750936,2.844742334433931,2.842614
velocity_x: [[-10.135779773162483,-10.164790448705373,-10.185671800671363,-10.144088795057396,-10.107290461858751,-10.08233440424025,-10.087851720682
velocity_y: [[3.0359271296884756,3.0329899998597862,3.031218812995912,3.0140308580389377,2.997264820097428,2.9979003298467415,3.0169633604323733,3.03
...
```

Figure 1. Summary of observations and features within an Argoverse 2 scenario file.

Through our exploratory data analysis phase of the project, we discovered that the motion forecasting set is composed of over 250,000 scenarios. Argo AI utilized the Parquet file format to compile each scenario separately. Within each file is a recording of the movement and metadata characteristics of a main agent vehicle along with the corresponding information for the other vehicles present in the roadway. Being a time-series dataset, the vehicle data is recorded into timesteps where the main agent has information for 110 frames (or 11 seconds at 10 Hz) across the majority of the scenarios. The other vehicles' number of timesteps may vary depending on the time they were in the recorded scene with the main agent. All of the data is in "tidy data", tabular format where each observation is a timestep for a particular vehicle and the feature variables are organized into the columns of the table. There are 16 feature variables which are either continuous or categorical. We found that there were no missing or dropped features during our analysis. Argo AI's use of 10Hz recording provides us with short, 10 second snapshots of vehicle movements and driver behavior in major US city traffic conditions.

Data preprocessing

Beginning our regression analysis, we chose to focus our attention on the data's continuous variables. Specifically, the position_x, position_y, heading, velocity_x, and velocity_y information. As stated earlier, we did not detect any missing data so imputation of values was not necessary. Our early models made use of the unaltered data while later analysis included normalization of the feature values to values between 0 and 1. As noted earlier, each scenario contains data on multiple vehicles. Although the main agents and many of the surrounding ones are recorded for the full duration, the dataset contains a number of vehicle observations that are not a full 11 seconds long. During initial model exploration, we decided to focus on data that represented vehicle behavior for a full 110 timesteps. Any time series less than that target number was disregarded; the disregarded datapoints represent 1% of the dataset. The decision was motivated by considering how analogous this numerical time series data is to natural language prediction tasks. In essence, we considered each full 110 timestep series of observations to be a sentence. By equating the observations to words in a sentence, the features then become the letters composing the word. We thus considered employing model architectures that excel at sequence to sequence translation, such as LSTM and GRU. Models such as these generally take windows of data as input into the modules. Adhering to only full length, 110 timestep sequences allowed us to treat the dataset as containing "sentences" of equal length that are easily divisible by window sizes of 10, where each window is composed of 10 "words".

To augment the data discussed earlier, three *social features* were computed using a helper function. That function uses the available data to compute the minimum distance (MIN_DISTANCE_FRONT), minimum distance back (MIN_DISTANCE_BACK) and the number of neighbors (NUM_NEIGHBORS) for the main agent of each scenario. These new features are concatenated along with the previous 16 feature variables and all of the timesteps are specified for the main agent vehicle only. The calculation of these new features also afforded some file-size advantages. The process creates new dataset files for train, validation and test sets that are significantly smaller and in Comma Separated Value (CSV) format. For example the training dataset is roughly 80 GB before processing and 1.5 GB afterwards. Once the new files are created, the original dataset can be disregarded.

Prediction Task

The task was to predict the trajectory of the main agent vehicle given the observed x and y positional data combined with the computed "social features" outlined above.

Model architecture

Over the course of the project, our team approached the prediction task on this new dataset through the development of 4 different model architectures. Our Observe 80/predict 30[3], Stacked LSTM[3, 4], single module GRU, and multi-layer perceptron models[5] constitute different

conceptualizations of data flow to effectively address the multi-modal nature of our aggregated information.

All of the models we developed during the course of this project share some common hyperparameters and features:

Batch Size	128
Learning Rate	1e-3
Leaky ReLU Alpha	3e-1
Dropout Rate	3e-2
Hidden Layer Dimension	100
GRU or LSTM units	128

Figure 2. Shared baseline hyperparameter values.

Dropout was used during the training phase between each of the hidden dense layers. Each model utilizes Mean Squared Error (MSE) for its loss function. Weight optimization is achieved through the Adam optimizer.

Multi-layer perceptron model (MLP)

Our initial investigation began with this architecture where we have an input layer, one hidden layer and an output layer. The first two layers use Leaky ReLU as their activation function. During the training phase of the model, it uses Dropout between the layers.

It should be noted that this model was trained with a small portion of the dataset to address early storage capacity and computational concerns. The observations made here may have been influenced by that but what we experienced served as a foundation for the project's later work.

A significant portion of this stage of development involved cultivating an understanding of the multivariate nature of the data and how it should be prepared for prediction accuracy. The n-gram concept gave us an approach that worked well. Again, relating time series data to words and sentences, we created "5-grams". Each observation is a timestep and therefore a "word". Four "words" would go through the model to predict the fifth. We'd like to reiterate here that each "word" is composed of the feature variables (position_x, position_y, heading, velocity_x, velocity_y, MIN_DISTANCE_FRONT, MIN_DISTANCE_BACK, and NUM_NEIGHBORS). The model was tasked with predicting all eight of the variables given as input.

This model was taken a step further when we altered the makeup of the true values. Here we calculated the difference between the values of the fourth and fifth timesteps. Using these difference measures improved both training and testing loss significantly.

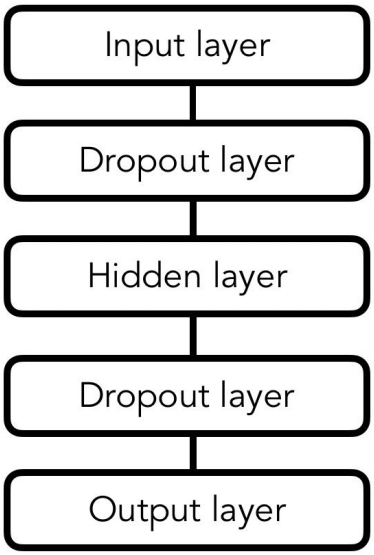

Figure 3. multi-layer perceptron model architectural diagram.

Single GRU module model

This model has a single GRU unit which processes all of its input and outputs it into a series of 2 dense layers. As with the previous model discussed, it utilizes LeakyReLU for layer activations along with Dropout during the training phase.

It was developed under the same limited data context as the multi-layer perceptron therefore the observations we gained from it may not be accurate of its full potential. It is included here because of its notable advantages and disadvantages to the multi-layer perceptron. We observed that the multi-layer perceptron reached a low level of loss with less epochs than this model. Conversely, the GRU's prediction accuracy appeared to be better than the MLP when given more epochs to train.

An explanation of this may reside in the way data is structured within n-grams. Because of its construction, the n-gram produces more data per scenario file in comparison to the windows of the GRU data preparation. That leads to more data being available during each epoch of training for the MLP. The GRU in effect needs to "catch up" with more epochs due to less available data per epoch.

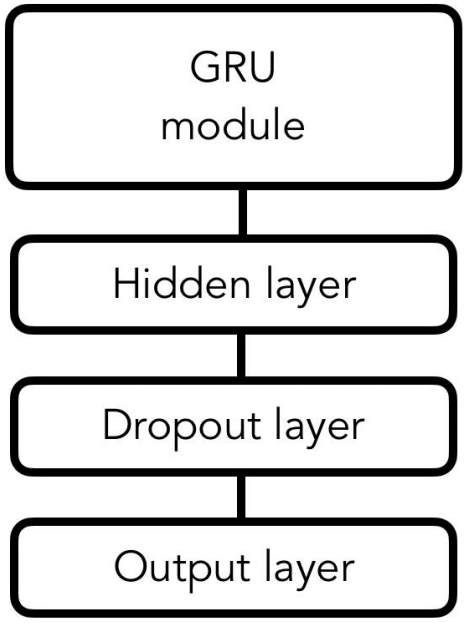

Figure 4. Single GRU model architectural diagram.

Stacked LSTM modules model

Our team developed this LSTM-based model in parallel with the observe 80/predict 30 model. This one is composed of an LSTM module whose output and final state are fed into a second LSTM module. The output of this second LSTM is input into a series of three dense layers. During the training phase, the model uses a dropout layer between the dense layers. During training, data was passed to the model in batches of 128 full length (110 frame) sequences.

The conceptualization here was to mimic the Observe 80/predict 30 model's ability to predict forecasting data over longer intervals. The previous models to this one predicted one new timestep for each batch of information sent as input. After training the model on the full computed dataset, it was validated using the full version of the validation set but with each agent time series providing 80 observations to the input (as opposed to 110 frames). The methodology here was born out of the 80/30 nature of observe/predict architecture. It should be noted that experiments were performed as well where unredacted observation series. The difference in testing metrics appeared to be negligible.

The model is recursively called 30 times to build the one observation output into a series of 30 forecasts. We experimented with predicting 2 (position_x, position_y) and 5 (position_x, position_y, MIN_DISTANCE_FRONT, MIN_DISTANCE_BACK, and NUM_NEIGHBORS) features to address challenges in the development. These discussion of these challenges will be expanded upon in later sections

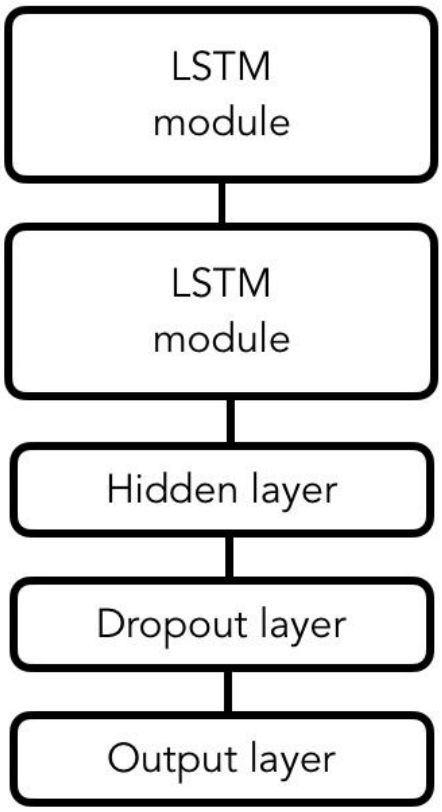

Figure 5. Stacked LSTM model architectural diagram.

Observe 80/predict 30 model

The observe/predict model has three components, a GRU, a GRU cell, and a set of dense layers. The observer processes the hidden layers, and returns a memory state over the entire observation sequence. The GRU cell takes the last observed values and the hidden state, as input state and input memory state. Over the entire prediction process, the cell uses the previous output states as input. At each prediction of the 30 prediction steps, the model calls the dense layers from the GRU cell's output state to produce a set of two values.

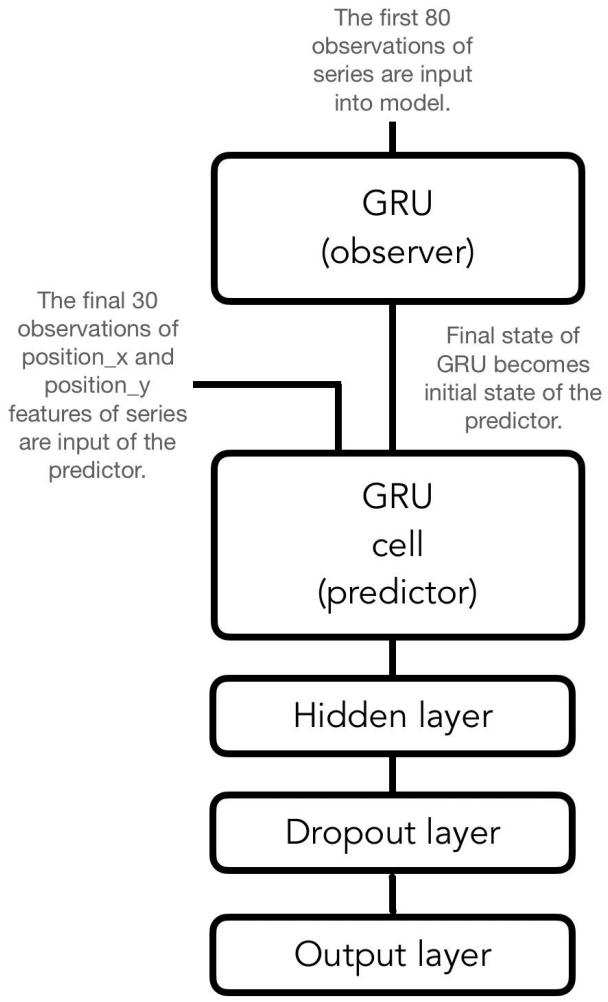

Figure 6. Observe 80 / predict 30 model architectural diagram.

Evaluation Metrics

Average displacement error, or ADE, is the average root mean squared error along a trajectory. It measures the difference between predicted and actual trajectories without exaggerating the effects of outliers.

MSE is used during our training process. It exaggerates the impact of large outliers during training so that the model can learn to minimize outlining inputs with reasonable speed.

Results

The observe 80/predict 30 model produces largely accurate predictions. We're observing a small degree of variability from between the ground truth and the training result as we move further from the last point of observation.

The Stacked LSTM performed comparably on metrics[6] when attempting to predict only x and y positions. When the number of predicted features increased to 5 (which was necessary to attempt to implement the recursive forecasting strategy) its performance suffered in comparison.

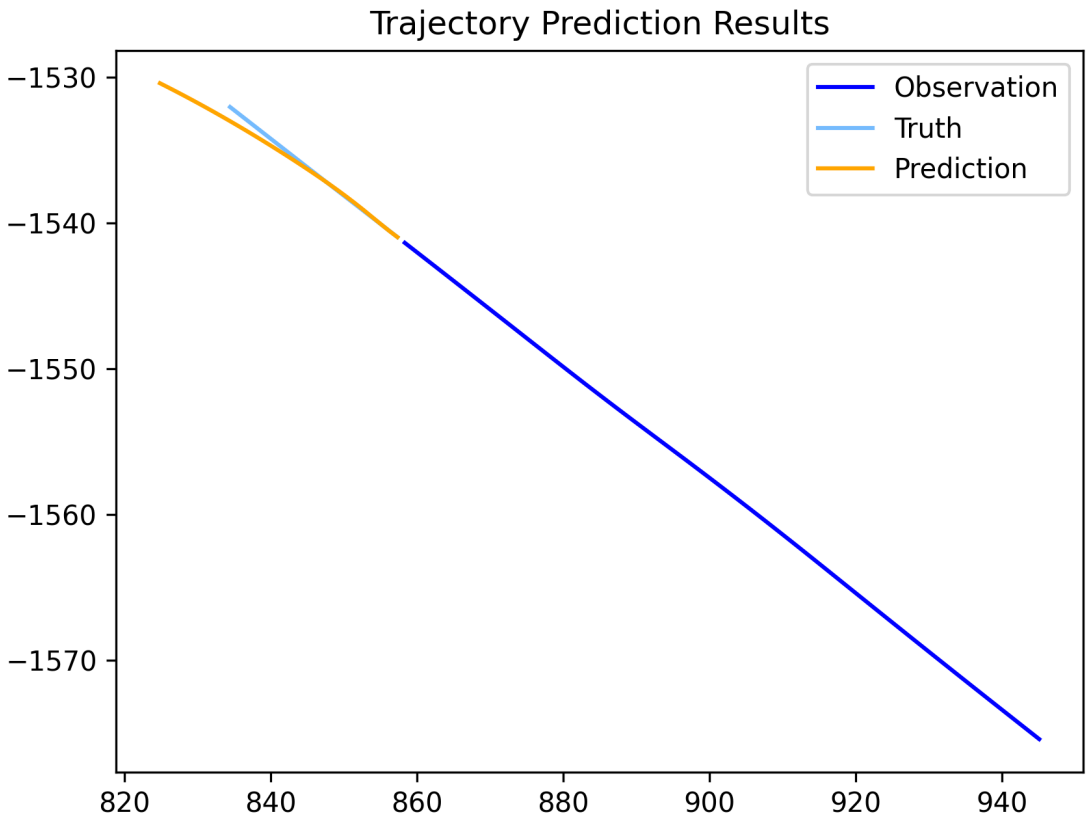

Figure 7. Trajectory prediction results plot for Observe 80/predict 30 model.

The following figure summarizes our validation metrics:

Model	MSE	ADE
Observe/Prediction	1.23	1.06
Stacked LSTM predicting 2 features (80 validation observations)	1.22	1.23
Stacked LSTM predicting 5 features (80 validation	255.05	22.33

observations)		

Figure 8. Metrics summary.

Discussion

In the observation/training model, since the next step is completely based on the results from the output state from the previous step, we worry that the error between the truth and the predicted positions will gradually build up. We did observe such errors in prediction results with particularly small unit size on particularly error-prone datapoints:

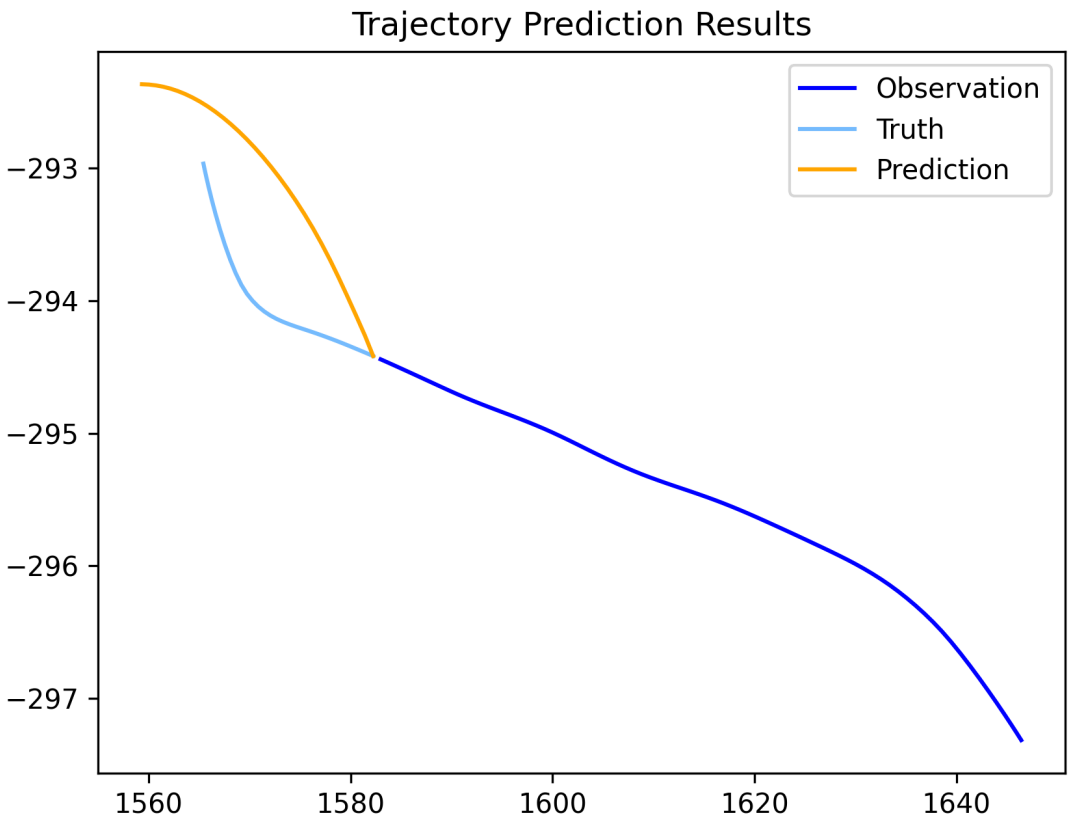

Figure 9.

We achieved higher ADE metrics than any models reported in the Argoverse 1 leaderboard (DCMS model at 1.4768), by a large amount. The observe and predict cadence creates a separation of concerns. The predictor gets the benefit of learning from the context of the observer while also only receiving 2 values for positional input into the cell while outputting 2

predictive values. The dense layers are also repeatedly called for each prediction step, together with the predictor GRU cell. This directly addresses the issues of recursive forecasting that the other models were hampered by the recursive prediction task.

We thought the prediction/observation model was the most reasonable, built it, and discovered its performance improvements. However, we are not sure which component or some nature of the dataset caused such performance improvement. To explain our performance, we performed the following experiments.

In the first experiment, we perform an ablation test on the recursive dense layers. We remove it from the recursive step. Instead, we use our GRU cells to recursively encode and output hidden information at a dimension of 128, and have the dense layers make final predictions at the end. We can see that the experimental setup performs better in ADE than our main model, and worse in MSE, indicating slightly worse performance for edge cases.

Experimental Setup	MSE	ADE
Recursive dense layers	1.23	1.06
Flat dense layer	1.28	1.00

If we significantly decrease our observation length, we can see that our accuracies slightly increase. However, the accuracy is still very high.

Experimental Setup	MSE	ADE
Observe 80 Predict 30	1.23	1.06
Observe 20 Predict 30	1.82	1.20

When we use only x,y positions to make forecasts, we understand that we could be training a neural network to perform regression without other contexts. But we noticed something very surprising:

Experimental Setup	MSE	ADE
With social features	1.23	1.06
Without social features	0.56	0.57

It turns out that the neural network trained without social features can perform significantly better in cases with nonlinear trajectories:

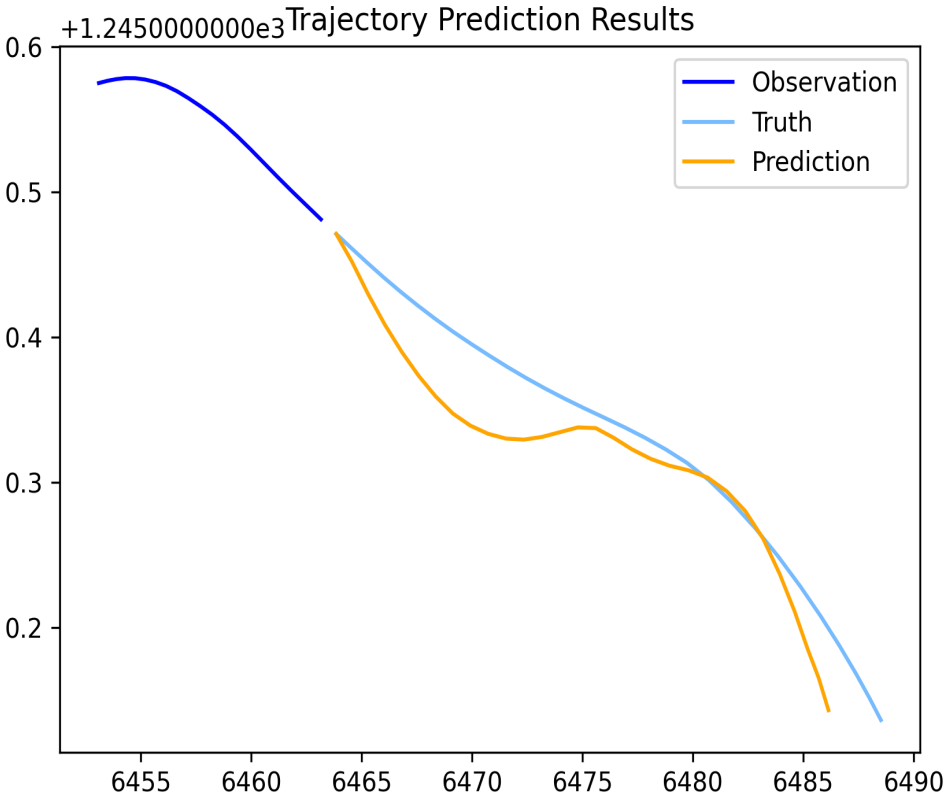
Trajectory Prediction Results
+1.2450000000e3
0.6
0.5
0.4
0.3
0.2
6455
6460
6465
6470
6475
6480
6485
6490
Observation
Truth
Prediction

Figure 10.

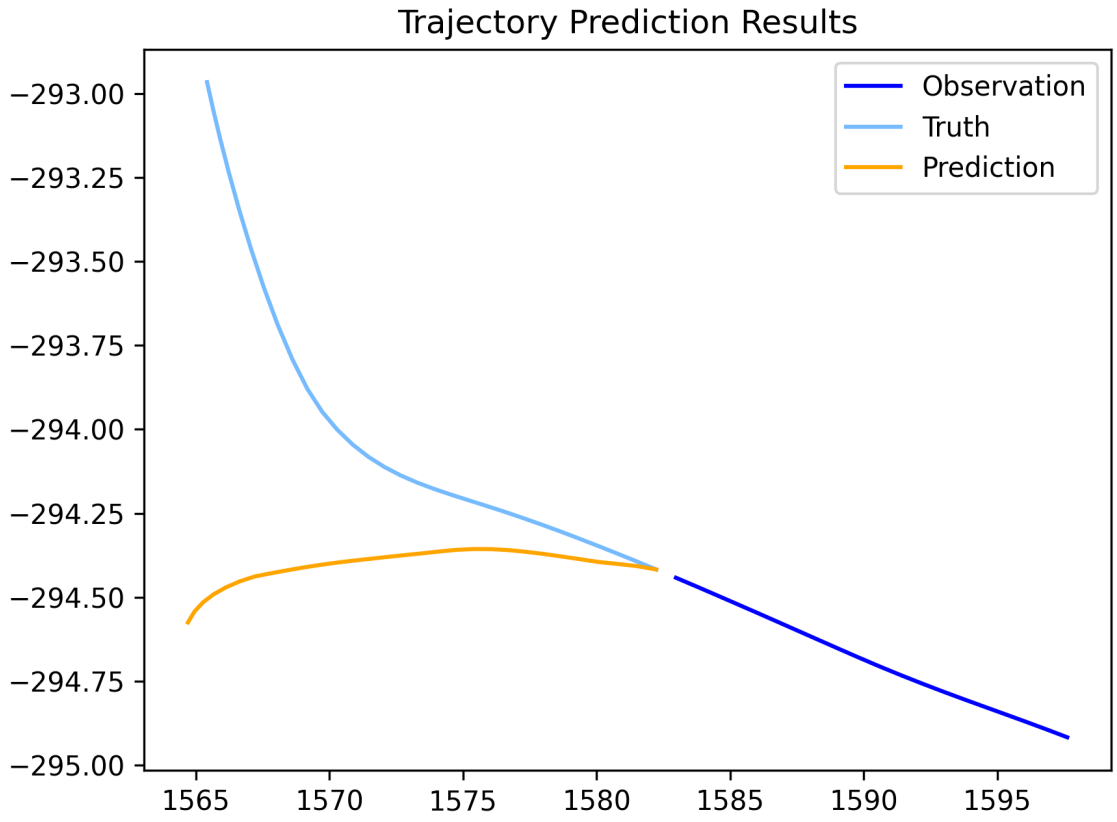

Figure 11.

In the meantime, it performs worse in other scenarios. The scenario above compares with Figure 9.

We believe that the network trained without social features performs better due to the large performance gain it obtains by performing tasks similar to linear regression in some cases but still failing in other cases.

Challenges

Argoverse 2 is a novel dataset and motion forecasting in self-driving is a novel topic for everyone in our team. Based on our prediction results (for instance, Figure 9) and how our models perform given the existing metrics, we believe that there is still a lot of future research work to develop better metrics, to systematically rule out large difference between observation and prediction.

The Stacked LSTM model was challenging to work into a true motion forecasting task using our method of recursive calls predicting the next step from the previous. For example, the architecture dictates that the input must include all 5 of the feature variables used to train the model. If the model is tasked to only predict 2 positional features then the process halts when attempting to feed that output back into the input recursively because the computed social features are not present. We experimented with concatenating values of 0 to represent the social features to the output before the next recursive step. Value imputation may also be an option here since in essence the social features are “missing” for the recursive input. Both options are potentially dangerous in this context.

Safety is paramount in applications of autonomous functionality. The two methods mentioned above cannot guarantee accuracy of prediction because the social feature values are either arbitrary or potentially inaccurate. The results show that it is possible to take the strategy of predicting all 5 features to satisfy the recursive step but as evidenced the metric accuracy suffers significantly.

Reflection & Future Work

Forecasting, by essence, is a very difficult task. For humans to forecast motion, we typically rely on the simple fact that an object will continue moving at the same speed as we see it. Based on our observations, recurrent networks perform well in learning this rule (for instance, figure 7). Therefore we believe that most of the challenge in this domain would be focused on cases such as Figure 9. However, it is worth debating whether there is any pattern based on nonlinear trajectories at all. In other words, we need to both explain how humans would perform such forecasting and in this context explain neural networks would be able to learn realistic patterns and make predictions.

A lesson we learned is to measure the amount of data and training time per batch early, so we could more accurately reason about compute time and resources. We also noticed that both GCP and Oscar are down near the end of the semester; it would be helpful to find and use a reliable and free alternative computational infrastructure at the very early stage of the project.

We also gained valuable experience in designing training pipelines. We found that it is extremely important to log and visualize metrics, and to save trained weights in an organized fashion. In addition, we also found that it is helpful to serve data as a generator, either with a Dataset class or a Python generator function.

Additionally, we were initially hoping to get to the point where we could adversarially attack[7] our models, but we ran out of time. Pivoting away from this helped us focus on getting reportable results from several of the models we had in development at the time, which was a good choice.

Despite the large improvement in ADE, we believe that there is still a lot of room for improvements, especially to improve accuracy for cases with non-linear trajectories such as shown in Figure 9.

It's also interesting to consider the application of observe and predict strategies to other forms of time series data such as stock prices, audio, and video.

Citation

1. Argoverse 2 Dataset: https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/file/4734ba6f3de83d861c3176a6273cac6d-Paper-round2.pdf
2. Argo AI, https://www.argoverse.org/
3. Multimodal Motion Prediction with Stacked Transformers Paper: https://arxiv.org/pdf/2103.11624.pdf
4. How to Develop LSTM Models for Time Series Forecasting https://machinelearningmastery.com/how-to-develop-lstm-models-for-time-series-forecasting/
5. How to Develop Multilayer Perceptron Models for Time Series Forecasting https://machinelearningmastery.com/how-to-develop-multilayer-perceptron-models-for-time-series-forecasting/
6. Argoverse 1 leaderboard https://eval.ai/challenge/454/leaderboard/1279
7. https://openaccess.thecvf.com/content/ICCV2021/papers/Li_Fooling_LiDAR_Perception_via_Adversarial_Trajectory_Perturbation_ICCV_2021_paper.pdf