EEL6938: Artificial Intelligence for Autonomous Systems

Trajectory Prediction Using DNNS

By Alexander Green

Trajectory Prediction Using DNNs

## Introduction

This project implements and evaluates a modified version of the trajectory prediction model from "TUTR: Trajectory Unified Transformer for Pedestrian Trajectory Prediction." The modification focuses on implementing a Minimum Average Displacement Error (MinADE) loss function to improve prediction accuracy. Using the NuScenes dataset, the model predicts vehicle trajectories for 4 seconds (8 time indices) based on 2 seconds (5 time indices) of historical data.

The implementation evaluates two distinct approaches: a single-trajectory prediction model (Part 1) and a multi-trajectory prediction model with MinADE loss (Part 2) that selects the best trajectory from multiple candidates. This report documents the implementation, code adaptations for Windows compatibility, and comparative results between the two approaches.

## Windows Conversion

When starting this assignment, I decided to make the code compatible with Windows operating systems rather than setting up a compatible Linux environment on my computer. The primary change was adding the ***if \_\_name\_\_ == "\_\_main\_\_":*** conditional statement at the beginning of the main execution block and indenting all subsequent code. This modification prevents code execution during import while maintaining functionality when run as a script, addressing Python's different module handling behavior on Windows platforms.

*A screen shot of a computer program

AI-generated content may be incorrect.*

***Code 1:*** *Main Function Conversion for Windows*

## How to Use

The code can be run on any platform after installing the required dependencies. First, download and unzip the project folder. Open a terminal in the project directory and install the requirements with ***pip install -r requirements.txt***.

For the baseline approach, run ***python train\_eval.py*** with default parameters. To implement the MinADE loss function, set the --minADEloss parameter to True using ***python train\_eval.py --minADEloss True***.

Additional parameters can be modified, such as the number of clusters (***--n\_clusters***), epochs (***--epoch***), or learning rate (***--lr***). For example: ***python train\_eval.py --minADEloss True --n\_clusters 75 --epoch 100*** would run the model with MinADE loss, 75 clusters, and 100 training epochs.

## Part 1 : Single Trajectory Prediction Results

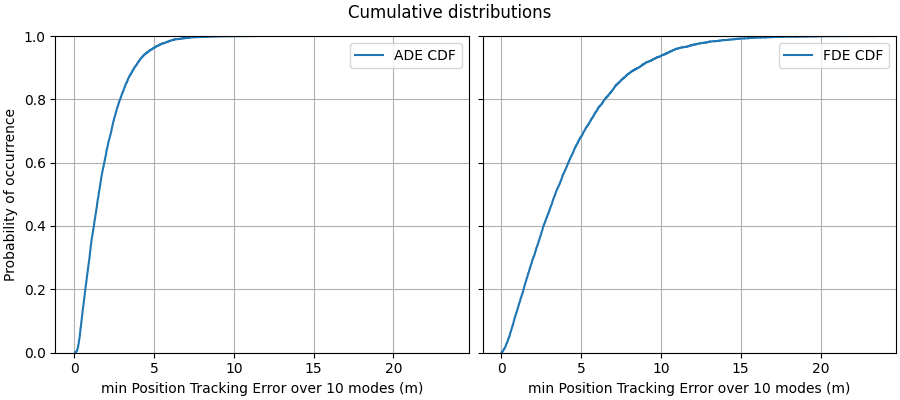
In the baseline implementation, the model is trained to predict a single trajectory with the highest probability. The evaluation metrics used are Average Displacement Error (ADE) and Final Displacement Error (FDE). ADE measures the average L2 distance between predicted and ground truth positions across all time steps, while FDE only considers the distance at the final prediction time step. This approach penalizes deviations from the ground truth trajectory without considering alternative paths or multimodal predictions. However, this method doesn't account for the inherent uncertainty in trajectory prediction where multiple valid futures might exist, potentially leading to higher error rates when the actual trajectory deviates from the most probable path.

### Part 1: Results

The following results were obtained by running the original code with “***python train\_eval.py***“ as described in the [**How to Use**](#_How_to_Use) section.

|  |  |
| --- | --- |
| Best Average minADE | 1.8851 |
| Best Average minFDE | 4.1531 |
| Best median minADE | 1.5393 |
| Best median minFDE | 3.3809 |
| Best 10th percentile minADE | 0.4601 |
| Best 10th percentile minFDE | 0.7929 |
| Best 90th percentile minADE | 3.7669 |
| Best 90th percentile minFDE | 8.5885 |

***Table 1:*** *Single Trajectory Prediction Results*



***Figure 1:*** *Single Trajectory Cumulative Distribution*

In the baseline implementation without MinADE loss, the model is evaluated using traditional ADE and FDE metrics, which compute the error between the single predicted trajectory and the ground truth.

These values reflect the inherent limitation of using a single-mode prediction in environments with multimodal trajectory possibilities. The model cannot adapt to the uncertainty in pedestrian or vehicle movement, often resulting in high endpoint errors when the predicted path diverges from the actual trajectory.

## Part 2: MinADE Loss Implementation Results

The MinADE loss implementation fundamentally changes how the model is trained by considering multiple potential trajectory predictions. I implemented the MinADE loss function to generate num\_k (10) trajectory predictions and select the one closest to the ground truth for loss calculation. The implemented code is:

A computer screen with colorful text

AI-generated content may be incorrect.

***Code 2:*** *Additional Loss Implementation*

This code implementation can be broken down mathematically as:

1. Individual position error at each timestep:

*The Euclidean distance between predicted and ground truth coordinates at time* ***t***

1. Average Displacement Error (ADE) for trajectory k:

*Average error across all timesteps for batch element* ***b*** *and trajectory* ***k***

1. Minimum ADE across all trajectories:

*Best trajectory error for batch element* ***b***

1. Final Loss:

*Average of minimum errors across the batch*

Where:

* ***B*** is the batch size
* ***K*** is the number of predicted trajectories (num\_modes)
* ***T*** is the sequence length
* & are the predicted coordinates at time ***t*** for batch element ***b*** and trajectory ***k***
* & are the ground truth coordinates at time ***t*** for batch element ***b***

Finally, this can be represented in one equation as:

This approach allows the model to handle multimodal predictions more effectively, as it only penalizes the best prediction rather than forcing all predictions to match a single ground truth.

### Part 2: Results

After the mention code changes were implemented, the results were obtained by running the original code with “***python train\_eval.py***“ as described in the [**How to Use**](#_How_to_Use) section. Note this is only possible because the default value for “***--minADEloss***“ was changed from False to True.

|  |  |
| --- | --- |
| Best Average minADE | 0.8554 |
| Best Average minFDE | 1.6460 |
| Best median minADE | 0.6538 |
| Best median minFDE | 1.0934 |
| Best 10th percentile minADE | 0.3076 |
| Best 10th percentile minFDE | 0.3467 |
| Best 90th percentile minADE | 1.6143 |
| Best 90th percentile minFDE | 3.4549 |

***Table 2:*** *Multiple Trajectory Prediction Results*

A graph of a number of individuals

AI-generated content may be incorrect.

***Figure 2:*** *Multiple Trajectory Cumulative Distribution*

With MinADE loss enabled, the model predicts multiple potential trajectories (num\_k = 10) and selects the one closest to the ground truth for error calculation. This significantly improves performance in multimodal prediction scenarios.

These results demonstrate the benefit of using the MinADE loss approach, which enables the model to accommodate multiple plausible future paths and minimizes the displacement error based on the best possible match.

## Conclusion

This project explored two approaches to trajectory prediction: a baseline single-trajectory model and a modified multi-trajectory model using MinADE loss. The results show that the MinADE approach provides significantly lower ADE and FDE values, especially in complex, multimodal environments. By penalizing only the most accurate trajectory prediction, the MinADE loss function helps the model learn a diverse set of plausible trajectories, resulting in more robust performance. This highlights the importance of incorporating multimodal prediction techniques in real-world autonomous systems where uncertainty is inherent.