GAN Enabled Drone Trajectory Prediction

Stanford CS236G Final Project

Akram Sbaih
Department of Computer Science
Stanford University
`akram@stanford.edu`

`Mentor` **Eric Zelikman**
Department of Computer Science
Stanford University
`ezelikman@cs.stanford.edu`

Abstract

Forecasting the possible trajecotries of other drones is important for a drone in a drone-race in order to avoid collision with them. In the other hand, there isn't only one possible trajectory for a drone given its history locations since it has the power to change its own direction. Therefore, I explore in this project training a Transformer GAN that generates a variaty of possible trajectories for the next second of a drone given its location history for the past 1.5 seconds. I propose methods for evaluating this task. Finally, I train and evaluate multiple novel approaches on the UZH FPV rapid drone racing dataset and discuss potential improvements for the model.

Code is available at https://github.com/aksbaih/drone-trajectory.

1 Introduction

Trajectory Prediction
This task is concerned with estimating the position of objects at a given point in the future. People have been interested in modeling this generic task since the beginning of science, but even more recently with the efforts to build autonomous agents that need to predict the trajectories of the environment in order to plan their own. An example of that is Pedastrian Trajectory Prediction needed when Autonomous Vehicles want to predict the trajectories of humans to avoid colliding with them.

Pedistrian Trajectory Prediction
Many datasets and benchmarks [1][2] have been introduced for the task of predicting the trajectory of pedistrians. The earlier models relied sololey on recurrent encoding on the position history of the individual pedastrians to predict their future trajectory as in [3] which also tackled the social aspect of such trajectories. After that, researchers started accounting for semantics of the space which heavily improved their predictions as in [4] and [5]. Generative adversarial networks have been also used for their ability to produce multiple possible trajectories, allowing for better recall (better safe than sorry) as in [5] and [6].

Drone Racing
Drones have become popular with ameture and professional racing enthusiasts in additino to its many practical uses in the modern economy [7]. Autonomous drones are yet more challenging than automobiles because their trajectories are 3-dimensional in the air. Yet it is important for an autonomous drone to predict the behavior of other surronding drones to avoid collision. This becomes harder and requires faster inference under racing conditions, where all drones (including the ego) are maneuvaring at high velocities.

2 Dataset

Source
I'm using the UZH FPV Dataset [7] for this project. The authors of this dataset wanted to challenge

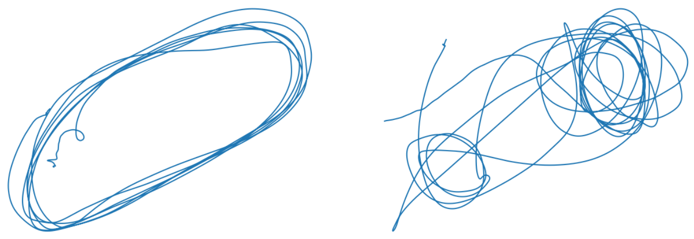

Figure 1: Example flight trajectories from the UZH FPV Dataset [7]

the state of the art state estimation models in UAV racing by introducing highspeed manuevers in real-world environments. This task aims to approximate the trajectory and momentum of the drone based on minimal sensory input (in this case, the grayscale view through a camera attached to the drone). The dataset consists of frames taken at 50Hz tied to a ground truth representing the momentum around each of the 6 degrees of freedom the drone has in 16 flights lasting around 100 seconds each.

Processing
I'm reusing the dataset for a different task here: I'm interested in predicting the immediate trajectory of a drone based on its location history as observed from a third-party (meaning we shouldn't assume access to the drone camera but we can observe its position history). This trajectory info is embedded in the ground truth of the original dataset. Therefore, I use the dataset toolbox and my own code in the linked repo to generate the xyz location of the drone at each frame of a constant FPS and store it locally as a txt file for each flight. Since I'm going to be using transformers, I reduce the FPS to 8 instead of 50 to span larger trajectory in less memory but this is subject to change based on later experiments. This totals 11486 frames. Figure 1 shows example trajectories.

Division
Since the number of flights is not very big and each flight has a different pattern of trajectories, I reused the method of [8] where the validation set is sampled as random windows of the training set when constructing the dataloader. I'm using 64 frames from each flight for validation, making around 9% (1k frames) of all frames, but this could increase if needed. For the testing set, I left aside two flights that were considered "medium" according to [7] consisting of 1.9k frames, or 17% of the original count. This leaves 74% (8.6k frames) for training.

3 Approach

The Transformer
Transformers have shown great success with sequential tasks in NLP and other domains [9]. They also offer parallelization making them faster than recurrent approaches like LSTMs. This makes them a good fit for my task given the sequential nature of trajectories and the need for speed of inference in a drone race. The purely transformer-based approach introduced by [8] for pedistrian trajectory prediction achieved state of the art results for that task. Therefore, I'm using their approach as the baseline and backbone of my GAN-based approach.

The Adversarial Loss
The baseline transformer uses MSE loss given the spacial nature of the data. However, adversarial loss allows for our desired diversity in the output. Figure 2 shows my proposed architecture which uses the transformer as a generator that takes in the positional history $x_{i-n}...x_{i-1}, x_i$, concatenated with random seeds, and the temporal embedding proposed in [8] as input. While the decoder generates a trajectory $y_0, y_1, ...y_m$.
The discriminator is an encoder that takes the concatenated sequence making the candidate trajectory and gives it a fakeness score as insipred by [10]. These two components make a complete GAN that can be trained with BCE loss to generate diverse predicted trajectories.

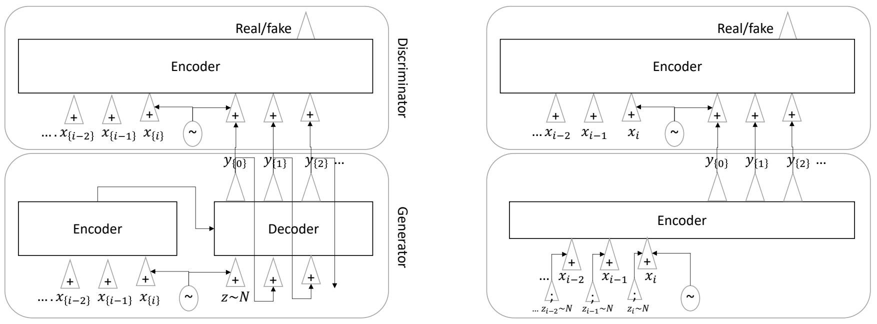
Real/fake
Encoder
Discriminator
Generator
Decoder
$z \sim N$
$y_{\{0\}}$ $y_{\{1\}}$ $y_{\{2\}}$ ...
.... $x_{\{i-2\}}$ $x_{\{i-1\}}$ $x_{\{i\}}$
... x_{i-2} x_{i-1} x_i
... $z_{i-2} \sim N$ $z_{i-1} \sim N$ $z_i \sim N$

Figure 2: The proposed GAN architecture. The circle represents the temporal embedding for each timestep as proposed in [8]. Left is the original proposal, Right is the new parallelized one.

Evaluation
I would like the predicted trajectory to be similar to the ground truth. The literature [8] uses the Mean Average Displacement MAD and the Final Average Displacement FAD. The latter of which averages the distance of the final state of the predicted trajectory from the final state of the ground truth, while the former does the same averaging over all the points of the trajectories.
Since we desire diversity in the GAN output, it is not fair to penalize it for the trajectories that don't match the ground truth. Therefore, I propose to consider only the minimum MAD and FAD over a batch of generations with diverse random vectors for the same history sequence. I could also experiement with the truncation trick to produce predictions with high fidelity and treat them the same way as the single-prediction baseline.
To measure the diversity of the GAN outputs, I propose to use a clustering algorithm (e.g. k-means) over its output on a batch of random vectors for the same history sequence and find the distance between these clusters. This metric is very loose because I'm not comparing the model diversity to the baseline, but it's a good indicator that there's at least some diversity in the generations.

4 Experiments and Results

The Baseline
I trained the simple transformer baseline using the same implementation linked in [8] with the training dataset for 130 epochs in 30 minutes on a Tesla K80. The batch size was 64 and the number of validation frames was 64 per flight (as described in the dataset section of this report). I also requested that the model be evaluated on the test set on every epoch for more insight given the small size of the validation set and ended up with both evaluations yielding almost identical curves with better results on the test set. Figure 3 shows the results and a few visualized samples.

GAN-Based
I implemented GAN-based model I proposed shown in Figure 2 (left) and trained it on the same dataset using Wasserstein Loss [11] with multiple values of gradient penalty and critic repetitions. The training curves are shown in Figure 4. Training took longer than for the baseline because of the sequential unfolding of the generator's decoder (coming from the assumption that we don't have ground truth for the generation task). This sequential nature also led to vanishing gradient for the generator where it didn't learn past the random weights it was initialized on, leading to mode collapse in the generations. Some examples are shown in Figure 5.

Analysis Looking at the loss curves in Figure 4, we find two things:

- Higher Gradient Penality allows the generator to learn more by slowing down the critic as the Cyan and Magenta curves are flipped for the Critic and Generator.
- Smaller trajectory window (unfolds of the generator) lead to better training and smaller loss for the generator. This is probably due to the vanishing gradient happening during the unfolding process for long sequences.

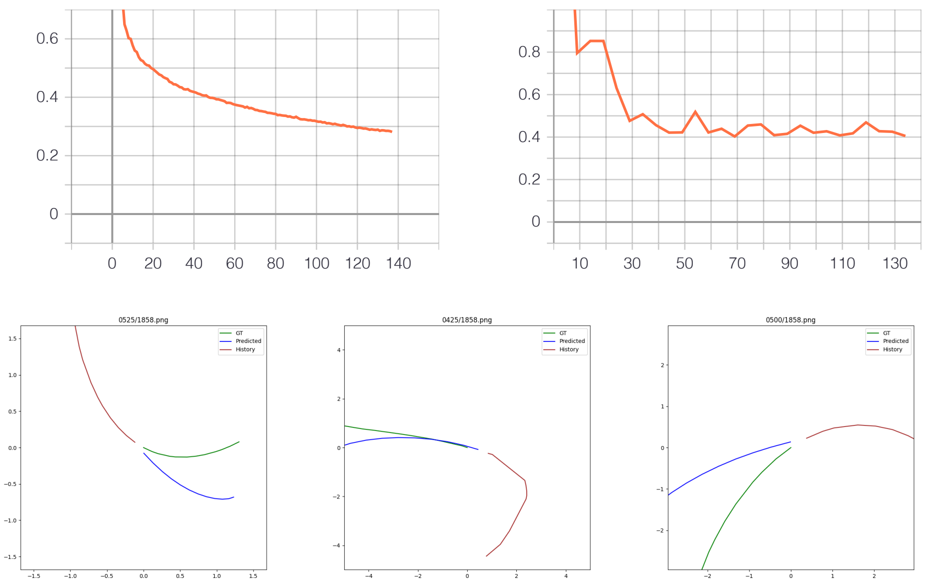

Figure 3: The x axis is the epoch. The left chart is the average loss per epoch. The right chart is the MAD on the test set at each epoch. On the second row are example outputs: brown is the input, green is the gound truth, and blue is the prediction.

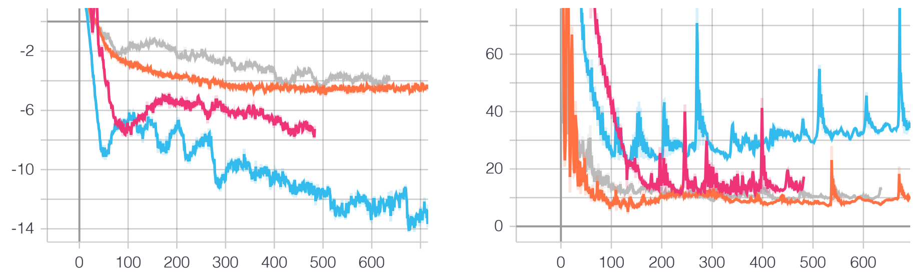

Figure 4: W-Loss curves for Critic on the left and Generator on the right with batch on the horizontal axis. **Cyan** is **G**radient **P**enality of 10 and **W**indow of 8; **Magenta** is **GP** of 100 and **W** of 8; **Grey** is **GP** of 10 and **W** of 2; **Orange** is **GP** of 10 and **W** of 1

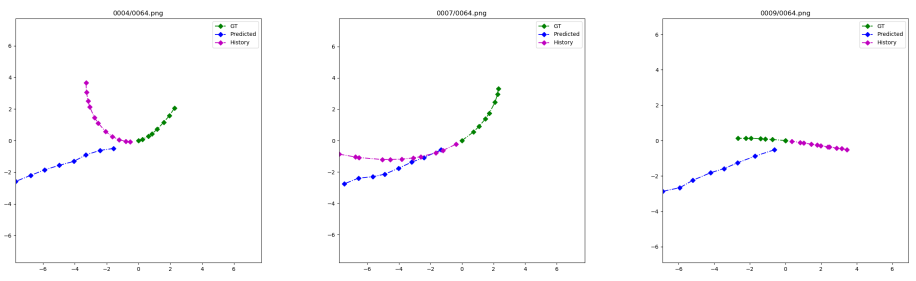

Figure 5: A few samples generated by the generator during training. Magenta is the condition input sequence, Green is the real trajectory, and Blue is the generated trajectory. Notice how similar all the blue trajectories are for various conditions.

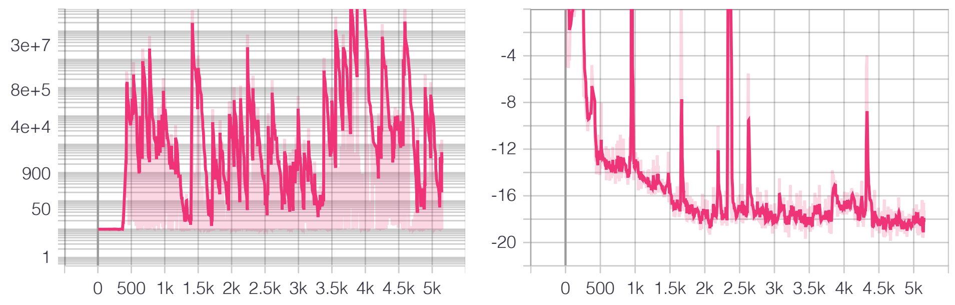

Figure 6: W-Loss curves for Critic on the left and Generator on the right with batch on the horizontal axis. Notice that learning still hasn't plateaued by the end.

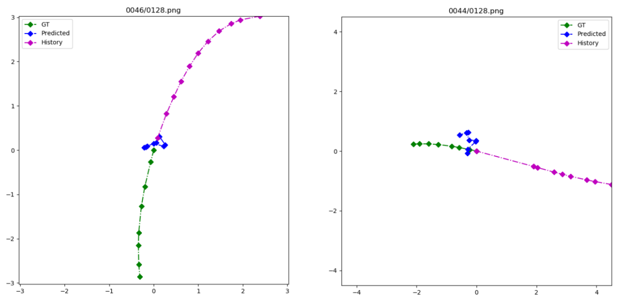

Figure 7: A few samples generated by the generator during training. Magenta is the condition input sequence, Green is the real trajectory, and Blue is the generated trajectory. Notice that blue is clustered around the end of the trajectory.

Improved GAN-Based
This leads to the conclusion that I should get rid of the sequential unfolding in the generator and to incorporate the injected noise better into the generator. Therefore I propose another approach shown in figure 2 (Right) where there's no sequential unfolding. Instead, the decoder is abandoned for a simple Encoder-only transformer (similar to BERT) where the noise is concatenated to the input states.
I also add an additional reconstruction loss term for the generator which is a scaled MSE loss. It's important to keep this scaled minimally so it doesn't reduce the variaty of the generations even though it boosts its fidelity. I ran training for the new method for 3 hours with the losses shown in figure 6. That is 78 epochs. The reconstruction loss was weighted by a factor of 2 which I chose after experimenting with 0.1 and 1. Notice that the model doesn't stop improving at that point but I had to stop the training because of the time constraint. Some examples are shown in Figure 7.

Analysis
Looking at the loss curves in Figure 7, we find two things:

- The new architecture solved the mode collapse problem and the predicted trajectories are not identical to each other anymore.
- The predicted trajectories have a problem with scale and the generations are too small spacially even with the reconstruction loss. There are multiple solutions to try for this. One is to increase the reconstruction loss weight. Another one is to normalize the input of the generator and rescale it to the original for the output. This is similar to instance normalization but with the additional step of scaling up/down to the original again at the output layer.

Risky Experiments

- Request a longer prediction horizons and use that to measure how over-fit the model could be. Because the past trajectory of an object doesn't necessarily contain enough information about the future, if the model gets good results, it means that it's leaking information from the training set.
- Set all the input states as random values and see how well the model performs on that. Good accuracy on that implies that the model doesn't depend heavily on the input but instead gives a general prediction regarless of the condition.
- Preprocess the coordinates of a drone during flight as seen from another drone during a concurrent flight (this doesn't exist in the dataset but we can assume that two flight recordings were happening at the same time and with the same coordinate reference/calibration). This will show us if the model actually generalizes to do the task when it's on a moving drone in a race rather than on a stationery sensor.

References

[1] A. Robicquet, A. Sadeghian, A. Alahi, and S. Savarese. Learning social etiquette: Human trajectory prediction in crowded scenes. In *European Conference on Computer Vision (ECCV)*.

[2] Stefan Becker, Ronny Hug, Wolfgang Hübner, and Michael Arens. An evaluation of trajectory prediction approaches and notes on the trajnet benchmark. In *ECCV*, 2018.

[3] A. Alahi, K. Goel, V. Ramanathan, A. Robicquet, L. Fei-Fei, and S. Savarese. Social lstm: Human trajectory prediction in crowded spaces. In *CVPR*, 2016.

[4] Karttikeya Mangalam, Yang An, Harshayu Girase, and Jitendra Malik. From goals, waypoints and paths to long term human trajectory forecasting, 2020.

[5] Amir Sadeghian, Vineet Kosaraju, Ali Sadeghian, Noriaki Hirose, S. Hamid Rezatofighi, and Silvio Savarese. Sophie: An attentive gan for predicting paths compliant to social and physical constraints, 2018.

[6] Agrim Gupta, Justin Johnson, Li Fei-Fei, Silvio Savarese, and Alexandre Alahi. Social gan: Socially acceptable trajectories with generative adversarial networks, 2018.

[7] Jeffrey Delmerico, Titus Cieslewski, Henri Rebecq, Matthias Faessler, and Davide Scaramuzza. Are we ready for autonomous drone racing? the UZH-FPV drone racing dataset. In *IEEE Int. Conf. Robot. Autom. (ICRA)*, 2019.

[8] Francesco Giuliari, Irtiza Hasan, Marco Cristani, and Fabio Galasso. Transformer networks for trajectory forecasting, 2020.

[9] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

[10] David Ifeoluwa Adelani, Haotian Mai, Fuming Fang, Huy H. Nguyen, Junichi Yamagishi, and Isao Echizen. Generating sentiment-preserving fake online reviews using neural language models and their human- and machine-based detection, 2019.

[11] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan, 2017.