CSC490 Module 2

Chi Zhang, Artur Kuramshin

April 2022

1 Handout Introduction

2 Part A: Object Tracking

2.1 Two-Frame Tracking

Part 2

For example:

in cost matrix C = $\begin{bmatrix} 0 & 0 & 0.2 \\ 0.7 & 0.8 & 0 \\ 0 & 0.9 & 0 \end{bmatrix}$

1. In greedy matching, algorithm will find minimum element C_{11}, C_{23}, C_{32} in each iteration in order, and the total cost at last is $0 + 0 + 0.9 = 0.9$
2. In hungarian matching, algorithm will find minimum element C_{12}, C_{23}, C_{31} in cost matrix, and the total cost at last is $0 + 0 + 0 = 0$

In such scenario, greedy matching may fail: In the cost matrix, there are more than one minimum element.

When there are more than one minimum element, greedy matching tends to pick the first occurance or a random minimum. When deleting the row and column that the picked minimum are in, greedy matching is likely to delete actual optimal elements which should be picked in next iterations. Since greedy matching only focuses on current iteration, it is hard to decide which minimum element to pick in current iteration when there are multiple minimums.

2.2 Multi-Frame Tracking

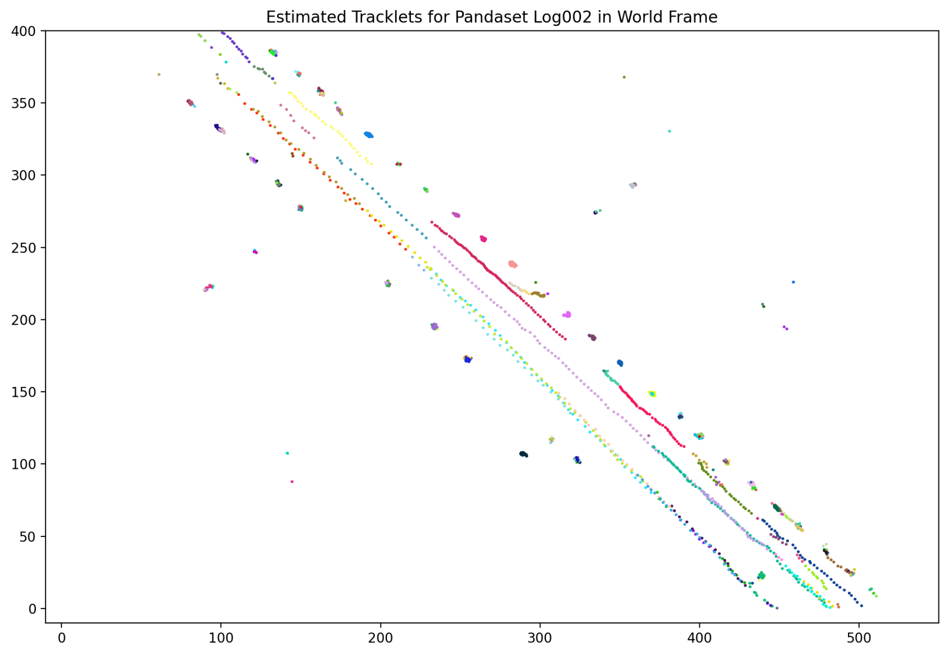
2.3 Evaluation

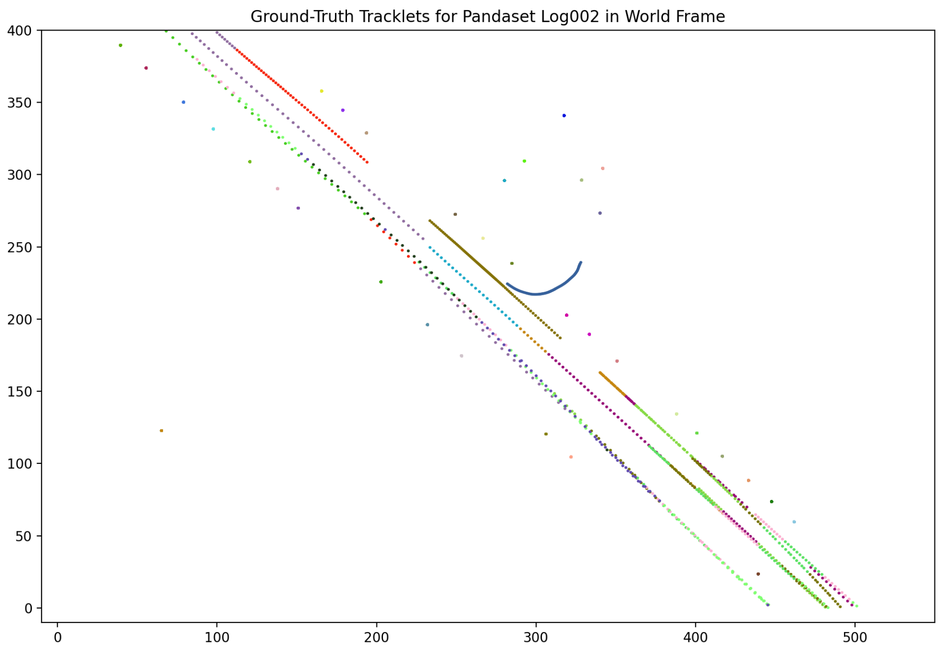
Part 4

Greedy matching algorithm and hungarian matching algorithm have the exactly same evaluation output, as shown in the table below:

vehicle	mota	motp	mostly_tracked	mostly_lost	partially_tracked
1	0.47575	0.57200	0.67611	0.30364	0.02024
2	0.35204	0.56917	0.75610	0.24390	0.0
3	0.25753	0.48482	0.34109	0.61240	0.04651
4	0.15725	0.62828	0.34270	0.64045	0.01686
5	0.52841	0.65332	0.67164	0.31716	0.01119
6	0.375	0.61630	0.52155	0.47414	0.00431
7	0.24525	0.61501	0.76582	0.20253	0.0316
8	0.40312	0.69233	0.619718	0.36620	0.01408
9	0.23974	0.66312	0.64623	0.32547	0.02830
10	0.0.26315	0.63089	0.64623	0.32547	0.0283
11	0.32564	0.67942	0.57862	0.38365	0.03774
12	0.26817	0.60932	0.55140	0.43925	0.00935
mean	0.32564	0.67942	0.58650	0.38365	0.02089
median	0.29690	0.62229	0.59917	0.37492	0.01855

This is one pair of visualization result of one sample:




Ground-Truth Tracklets for Pandaset Log002 in World Frame
400
350
300
250
200
150
100
50
0
0
100
200
300
400
500

2.4 Improved Tracker

2.4.1 Introduction

In the basic tracker, we only leverage 1 - IoU as a measure of the cost between bounding boxes $\mathcal{B}_1(i)$ and $\mathcal{B}_2(j)$. We propose to compute loss based on *geometry distance* between targets and predictions.

2.4.2 Methods

We combine the geometry distance of centroids, sizes and directions of vehicles. The cost function is formulated as below:

$$\mathcal{L} = -\Sigma_{n=1}^{N}\Sigma_{t=1}^{T}\{\sqrt{(x-\hat{x})^2+(y-\hat{y})^2}+\sqrt{(l-\hat{l})^2+(w-\hat{w})^2}+|\theta-\hat{\theta}|\}$$

2.4.3 Experiments

Setup We set the hyperparameter *match_th* as 5.0.

1. run with hungarian matching algorithm

vehicle	mota	motp	mostly_tracked	mostly_lost	partially_tracked
1	0.23920	0.49071	0.50000	0.47159	0.02841
2	0.14494	0.62855	0.41333	0.58667	0.00000
3	0.50209	0.65584	0.74937	0.24810	0.00253
4	0.28764	0.61639	0.70638	0.29362	0.00000
5	0.16001	0.62721	0.85041	0.14472	0.00488
6	0.33608	0.69315	0.80676	0.19082	0.00242
7	0.22125	0.66312	0.65801	0.32900	0.01299
8	0.24446	0.63365	0.70927	0.27796	0.01278
9	0.30079	0.67942	0.71910	0.26217	0.01873
10	0.23183	0.60982	0.65940	0.33787	0.00272
mean	0.28337	0.62014	0.70558	0.28665	0.00777
median	0.26605	0.62788	0.71418	0.27006	0.00380

2. run with greedy matching algorithm:

vehicle	mota	motp	mostly_tracked	mostly_lost	partially_tracked
1	0.47708	0.57437	0.66800	0.31200	0.02000
2	0.35147	0.56940	0.76562	0.23438	0.00000
3	0.25753	0.49072	0.34884	0.60465	0.04651
4	0.15725	0.62828	0.35000	0.63889	0.01111
5	0.52472	0.65333	0.68531	0.30769	0.00699
6	0.37221	0.61630	0.53441	0.46559	-5.55111
7	0.23965	0.62721	0.79188	0.17766	0.03046
8	0.40199	0.69315	0.65161	0.34193	0.00645
9	0.23672	0.66312	0.58959	0.38150	0.02890
10	0.26245	0.63089	0.64679	0.33027	0.02294
11	0.32488	0.67942	0.57831	0.38554	0.03614
12	0.26570	0.60939	0.57013	0.42081	0.00905
mean	0.32264	0.61963	0.59838	0.38341	0.01821
median	0.29529	0.62774	0.61819	0.36172	0.01555

2.4.4 Discussions

Greedy matching is likely not to lead to optimal results and count more vehicles. Hungarian matching is more close to the true value. From the tables above, we could see mota and motp are both less than baseline.

2.4.5 Next steps

We should add weights to geometry distances of centroids, sizes and directions. And wo could also try other cost functions between predictions and targets, like huber loss. Huber loss could handle outlier datapoint in time series data by tuning the hyperparameter inside loss function, δ.

3 Part B: Motion Forecasting

3.1 Baseline Motion Forecaster

Part 1

Adding the current and past yaw information as input helps the model predict future waypoints due to it now having more information to reason about dynamics. For example, when a car is performing a U-turn, if we only have its past position as information, it is reasonable to predict the next state as a linear extrapolation of the last couple states. While in reality, if have information about the changing yaw values of the car, we might be able to predict the change in yaw and see that the car will continue to turn before proceeding forward.

Part 2

One alternative possible output parameterization is a heatmap occupancy grid. With this parameterization, in each grid cell, the model will output a probability of the vehicle being in that location at time step t. The pros of this output representation is that it is probabilistic and not restricted so a single mode. The cons would be the difficulty of representing this distribution. Since it is not easily parameterized like a Gaussian, it is harder to efficiently differentiate and optimize.

Part 5

The two hyper-parameters that we decided to investigate were the amount of input frames used by the model and the encoder architecture. For both hyper-parameters, we looked at two possible values. For the encoder we looked at varying the amount of hidden units in the layer before the output layer and for the input frames we looked at the history length used as input.

- **Amount of input frames** (# of IF): 10 or 20
- **Encoder architecture** (# of HU): 256 hidden units or 64 hidden units

We trained the model with the various combinations of these hyper-parameters and evaluated them with the ADE and FDE metrics. The results are presented in the following table:

# of HU	# of IF	ADE	FDE
64	10	0.566351	1.268647
256	10	0.571151	1.260739
64	20	0.582686	1.307818
256	20	0.572781	1.266808

Intuitively we initially though increasing model size and increasing history horizon will both be beneficial. After evaluating the models, we saw that increasing the model size was beneficial but increasing the horizon from 10 to 20 had a negative effect.

3.2 Improved Motion Forecaster

3.2.1 Introduction

In order for the self-driving vehicle to plan a safe route that avoids collisions and maintains a buffer distance with other actors, the motion planner has to rely heavily on the motion forecasting system. The motion forecasting system has to accurately predict where other actors will go in the future. It is important to acknowledge that the future is uncertain and it is hard to predict the exact singular future. Instead of trying to predict a singular future trajectory, if we predict a distribution of possible trajectories, then our SDV can plan safer motions that take into account the stochasticity of the environment. For our approach, instead of predicting a waypoint $s_t = (x, y)$ at each future time step t, we predict a Gaussian distribution $\mathcal{N}(s_t|\mu_t, \Sigma_t)$. Gaussian distributions are very well explored and are computationally very convenient.

3.2.2 Background

Given the past trajectories of an agent in a scene $\mathbf{x}$, we want to predict a parametric distribution over future trajectories $\mathbf{s}$: $p(\mathbf{s}|\mathbf{x})$.

A trajectory $\mathbf{s}$ is a sequence of waypoints from time $t = 1$ to a fixed time horizon T: $\mathbf{s} = [s_1, s_2, \ldots, s_T]$. We are going to model trajectory waypoint uncertainty as a Gaussian distribution parameterized by μ_t and Σ_t which are predicted by our model as a function of $\mathbf{x}$ at each future time step t:

$$p(s_t|\mathbf{x}) = \mathcal{N}(s_t|\mu_t(\mathbf{x}), \Sigma_t(\mathbf{x}))$$

We will train this model, parameterized by weights θ, by setting the goal of our model to predict parameters for our distribution p such that the likelihood of the ground-truth trajectories $\mathbf{y}$ is maximized. First, lets define our data as a list of size N (number of actors), containing (history, future) pairs: $[(\mathbf{x}^1, \mathbf{y}^1), \ldots, (\mathbf{x}^N, \mathbf{y}^N)]$. We will update the weights θ according to the negative log-likelihood loss:

$$\mathcal{L}_{\text{NLL}} = -\Sigma_{n=1}^{N} \Sigma_{t=1}^{T} \log \mathcal{N}(y_t^n | \mu_t(\mathbf{x}^n; \theta), \Sigma_t(\mathbf{x}^n; \theta))$$

3.2.3 Method

To implement this algorithmically, we had to modify the decoder and create a new loss function.

Our decoder (**Fig.1**) consists of an initial fully connected + ReLU layer which splits off into two fully connected heads. The first head outputs the predicted distribution means $\mu_t \in \mathbb{R}^2$ and the second head outputs the predicted lower triangular matrix $\mathbf{L}_t \in \mathbb{R}^{2x2}$ such that $\mathbf{L}_t \mathbf{L}_t^T = \Sigma_t$. To make sure that the diagonal elements of $\mathbf{L}_t$ are strictly positive, we pass the diagonal elements of the outputed matrix through the ELU activation and add 1.

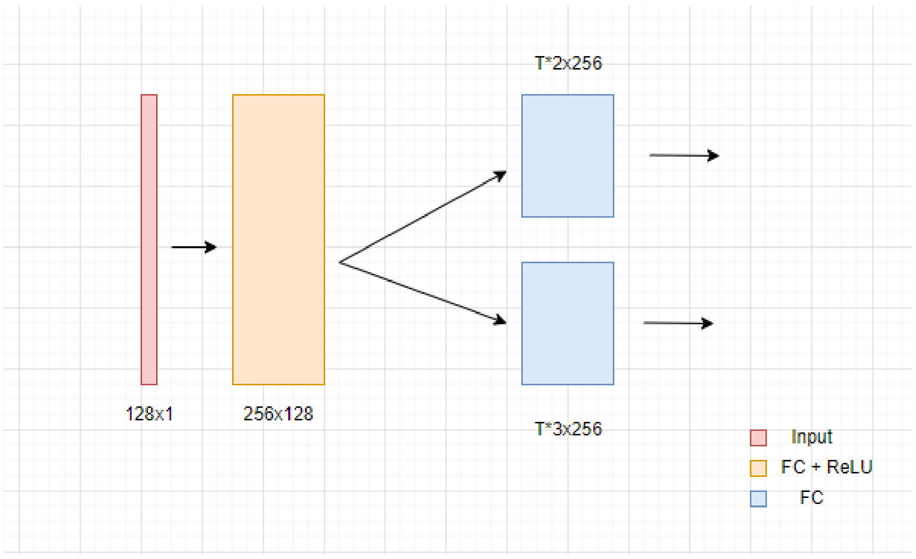


Figure 1: The modified decoder architecture.

To implement the loss function, we took advantage of PyTorch's `MultivariateNormal` distribution. We first filter out both prediction batches according to where the ground truth values are `NaN`. We then define the predicted distribution `MultivariateNormal` with `loc` set to our predicted means and `scale_tril` set to our predicted matrix $\mathbf{L}$. We then get the loss by getting the negative log probability of the ground truth targets in the distribution using `-1*log_prob(targets)`.

For the encoder we used the best performing encoder from our previous tests of 256 hidden units and 10 input history frames.

3.2.4 Evaluation

One of the hyperparameters we looked at was the activation for the diagonal elements of our predicted matrix $\mathbf{L}$. Since the diagonal of $\mathbf{L}$ has to be strictly positive and $\mathbf{L}\mathbf{L}^T = \Sigma$, the way we restrict the diagonal elements of our model is important. In our model we use the following formulation: $\mathrm{diag}(\mathbf{L}) = 1 + \mathrm{ELU}(\mathbf{x})$ where $\mathbf{x}$ is the diagonal elements part of our decoder output. ELU is the Exponential Linear Unit and is defined as follows:

$$\mathrm{ELU}(x) = \begin{cases} x & x > 0 \\ \alpha * (e^x - 1) & x \leq 0 \end{cases}$$

By varying the hyper-parameter α, we can vary the range of our diagonal values of $\mathbf{L}$ and as a result Σ. We evaluated the performance of our model by also adding an LL metric which is the log-likelihood of the target trajectories in the distribution predicted by our model. The results of different α values is presented in the following table:

α	ADE	FDE	LL
0	0.781559	1.715324	-2.673692
0.1	0.731492	1.523649	-2.561807
0.2	0.700884	1.505558	-2.375911
0.5	0.943869	1.88784	-2.207467
0.9	21.942719	10.22762	-4.045691

We found a value of $\alpha = 0.2$ to be the best as it performed the best on average in all the metrics. Alpha values of 0, 0.1, 0.9 performed worse in all metrics, and 0.5 performed better only in the LL metric.

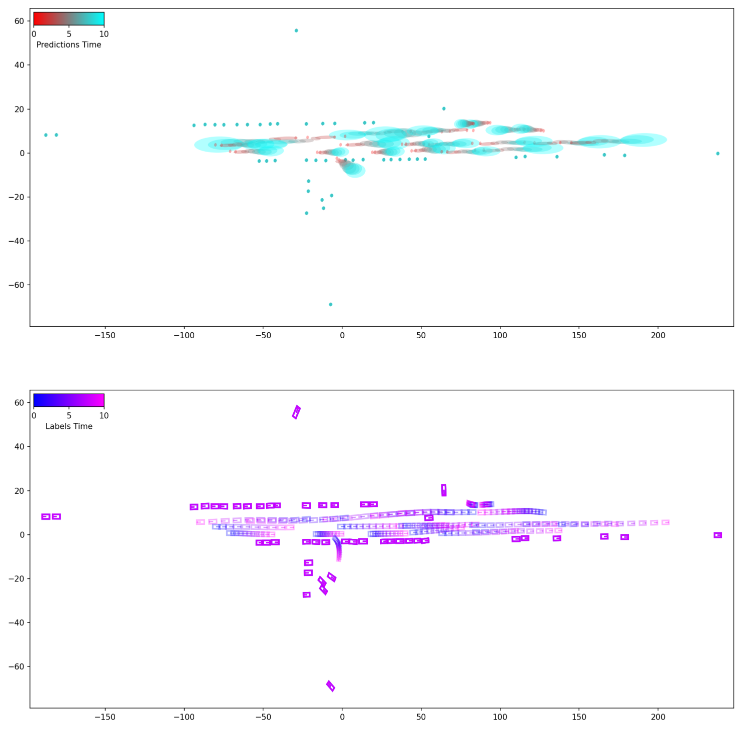


Figure 2: Probabilistic motion forecaster $\alpha = 0.2$

3.2.5 Limitations

Although the decision to predict the lower triangular matrix $\mathbf{L}$ has the benefits of training and inference performance, it also limits how expressive our model is. If the model was directly predicting the covariance matrix Σ, then it would be predicting 4 values instead of 3. The additional output variable might have helped the model explore more of the loss landscape, although we still would need to impose a positive definite constraint on Σ.

Another limitation and area of future exploration would be the model input features. There are several examples of our model not performing well for trajectories that involve large turns or curving roads (**Fig.3**). The problem stems from the model being unaware of the constraints that we have when driving. Although driving does not involve fully constrained motion, it is generally structured well by roads and intersections. It will be interesting to explore the benefit of adding road features to the input of our model.

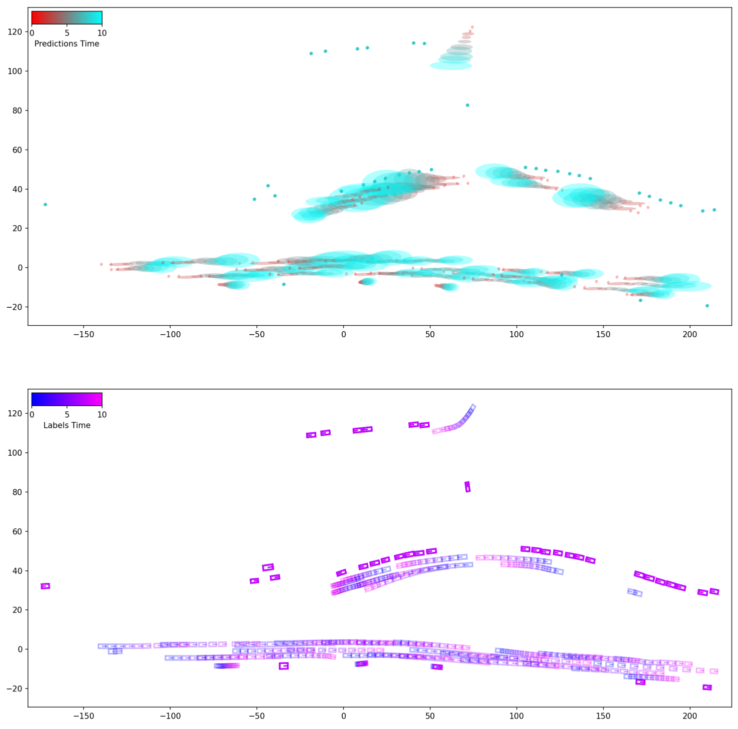


Figure 3: Failure mode (top vehicle turning)