COMP7310 PERSONAL PROJECT

Kang, Zheming

The University of Hong Kong

ABSTRACT

This project report introduces the solution of my IMU indoor tracking task. First, I will introduce the problem formulation and the LSTM baseline model. Then, I'll describe my methodology and model design. After that, I'll introduce the experiments I made during the contest. In the end, I'll make a conclusion of this personal project.

Index Terms— Indoor Tracking, IMU sensor, LSTM

1. INTRODUCTION

For COMP7310 personal project, we are required to design a Machine Learning or Deep Learning model for an IMU-based Robot Indoor Tracking. We have 4 dataset which are Training Set, Validation Set, Test Seen Set and Test Unseen Set. The data for our model training is collected in the office while those for testing is collected in large rooms. From the description, we could see the data was collected with a smartphone on a two-wheel robot by its IMU sensor.

The data format is hdf5, which are collected by Google Tango Phone. Our data contains two parts: synced and pose. Synced data are collected by the phone, and pose data is the ground truth of the robot. In synced data, we have gyroscope, acceleration with and without gravity, magnetometer, rotation vectors, and timestamp data. In pose data, we have time stamp, position and orientation of the robot.

Our task could be described as a regression problem: we want to predict the robot's moving trajectory with the data from IMU. The data loading and converting was done by the base line workflow.

Since most of the work of data was finished by basic workflow, we could focus on our model and training. The baseline model contains 8 parts. The first part is config setup, which allows us to adjust the parameters and hyper parameters of our model. Next is model design. In model design, baseline model constructed a Bilinear LSTM Network with pytorch for us, which is a class inherited from torch.nn.Module. The model consists of a bilinear layer, a lstm layer, two linear layer and a hidden layer. During the forward period, we first make a mixed input which is mixed from a bilinear and the original input. Then, we modeling the mixed input by a lstm layer and get the output and hidden layer. Then we connect mixed input and output together and input it to linear1 and linear2 layers respectively. Finally we get an output from linear layer 2.

The third part is data loader. It reads the input file and takes synced and pose data. Initially it read gyro and acce data from synced data with game rotation vector. It also reads position and orientation data from pose data. Data loader then convert data to appropriate formats. The fourth part is criterion, which provides loss function calculations for the training. Baseline model use MSE loss in global position loss class, and calculate the cumsum value of both target and predict data. There is also an MSEAverage class which provides MSE average calculations. Utils is the fifth part and it provides useful tools for this project. Then there comes the main function in the sixth part. Training and testing functions are defined in part six. Finally we train, test and save the answer in part 7 and 8.

In training, we have loss function to see the difference between training answer and ground truth. In evaluation, we have absolute trajectory error(ATE), relative trajectory errer(RTE), position drift error(PDE), and absolute yaw error(AYE). We calculate them in every epoch of testing.

2. METHODOLOGY

Notice that the baseline calculates MSE loss in calculating global position loss, we could use L1 loss instead of MSE. This is because L1 loss is more robust than MSE loss and during the training, we could see there exist a outlier in our dataset. By changing this, we could have a little improvement.

Deeply dive into the GlobalPosLoss class, we could found the loss value was calculated by torch.cumsum in the second dimension of the data. Torch.cumsum provides the accumulating array of the input array, i.e., all the value will be accumulated in the second dimension of the array. Back to our task, we want to predict the trajectory of the smartphone and reduce the error. We have gyroscope velocities and three dimensional accelerations. This is a regression problem so we used a LSTM network. However, in the global position loss function, we calculated the cumsum value. Since we had velocities data, we actually computed the cumsum of the velocities according to the 6-Dof of IMU tutorial[1]. It could be inferred that if we use the

velocities instead of accumulated values, we could predict precisely since each time our speed could be taken with an error. Following this, I get rid of the torch.cumsum and use the raw input to compute loss.

After finishing those two steps, the result improved but not significantly. Back to the introduction, we could see the angle of the tango phone is different when collecting data. Knowing this, we could assume that the different angles bring different gravitational results when collecting accelerations. Google tango phone provides gravity-free linear acceleration which confirms this assumption. Since our task is to predict the trajectory, we shall only focus on the linear acceleration if the data are collected in office and large flat rooms. Following this idea, I loaded linear acceleration instead of acceleration.

Then, I changed hyper parameters to make training converge faster. After the last epoch, I compare the 4 evaluation values to see if the model is really converged.

3. EXPERIMENTS

At the very beginning, I tried to run the baseline model to see the result, the result was around 0.2. Then, I tried to adjust the hyper parameters. I increased the layer size and the number of epochs and did see an improvement. However the improvement was too weak to pass the line. I also tried to add one layer to see the improvement but things went worse.

Then, I read the details of the baseline and changed the loss function to L1 loss. This makes a great improvement but after that, it seemed I had nothing to do with the model. I tried increasing the epochs and layer size but the improvement is tiny. From figure 1 you could see the previous result I got. There must be other things to improve.

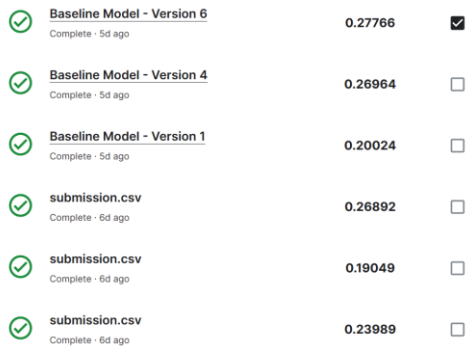

Figure 1. L1 loss with hyper parameters adjusting

Then, I went to read the paper of RoNIN[2] and found their project on github. From the paper I found they used a layer to reduce the error when a smart phone is carried by a human being. However, in our task, we have a Google Tango Phone on a two-wheel robot to get the data. Since our data seems not the same as those from RoNIN, we could save an integration layer. From the data we have, we could found acceleration and linear acceleration. Also, we have an attribute of gravity. From the description, we found linear acceleration is gravity free. That allows us have no need to calibrate the velocity, and we can use linear acceleration directly for our task.

From RoNIN's work and our basline, I found there were torch.cumsum functions in the forward function in global position loss class. This function calculates the sum of a specific dimension of the matrix. If we use original velocities, we could make an improvement in AYE calculation since it will calculate with v_i.

Put all the findings together, I made some different combinations and the result is in figure 2. Sometimes you need some luck to train a brilliant model, but for most of the time, a good combination of hyper parameters and loss functions will increase our score. The original submission record is in figure 3. Since my GPU usage was exceeded, and I changed new methods, the submission of figure3 is in the latest account. The submission of Figure1 was in team" NoLongerInUseAsGPUExceeded". The last line of figure was trained from gyroscope data and magnet data. This result is too bad. You can see high ATE, RTE, PDE and AYE so I didn't submit it.

Type of acce	Learning ra	epoch	layersize	loss func	ATE	RTE	PDE	AYE	Grade
cumsum acce	0.0015	200	200	L1	1.65943	1.273356	0.028357	60.18414	0.20024
cumsum acce	0.0015	200	150	L1	1.104873	0.974438	0.020491	46.30662	0.26964
cumsum acce	0.0015	300	150	L1	1.337955	1.078532	0.024194	46.61394	0.27766
acce	0.0015	150	100	MSE	N/A	N/A	N/A	N/A	0.25902
acce	0.003	150	100	MSE	3.89492	2.05817	0.064196	31.79003	0.25994
linacce	0.003	150	100	L1	1.164487	1.097063	0.025857	23.89906	0.31987
linacce	0.0015	220	100	L1	1.640054	1.202764	0.033969	23.16242	0.33313
linacce	0.0015	200	150	L1	1.155367	0.991983	0.020229	22.62314	0.34612
linacce	0.003	150	100	Smooth L1	1.235992	1.180402	0.026262	25.53589	0.35038
magnet	0.003	150	100	L1	9.960201	6.253578	0.184015	84.40849	N/A

Figure 2. Result of different combinations

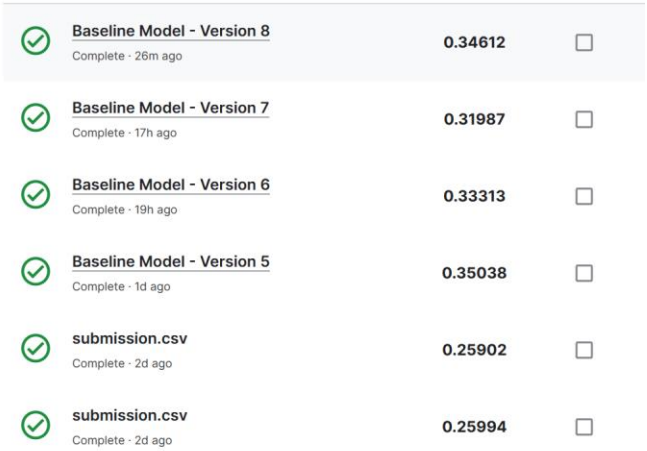

Figure 3. Original submission for replacing acce type

I also tried smooth L1 loss function, which is defined in torch package and more robust than L1 function. It is smooth near 0. The result shows the smooth L1 with linear acceleration is the best combination.

4. CONCLUSION

In this report, I designed a biLSTM model with a customized loss function. The result increased 0.15989, which is 0.0885 higher than the base line.

In the new model, I keep the biLSTM model but use smooth L1 to compute the loss value of gyroscope and linear acceleration data from Google Tango Phone. I also made a comparation with baseline model and my model. The results shows a bigger layer size, more epochs can increase the performance. The learning rate should be suitable for the epochs. If the learning rate is high, it will converge quickly but gave a bad result near a value. If the learning rate is small, it will converge slowly and need more epochs. The result also shows that the Smooth L1 is more powerful than L1, and L1 is more powerful than MSE.

5. REFERENCES

[1] Coordinate System Transformation for Inertial Sensors. From course materials.

[2] Herath S, Yan H, Furukawa Y. Ronin: Robust neural inertial navigation in the wild: Benchmark, evaluations, & new methods[C]//2020 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2020: 3146-3152.