AI for State Estimation

Project

Zanolin Lorenzo[1]

[1]Control of Networked Systems
Universität Klagenfurt

February 2024

UNIVERSITÄT KLAGENFURT

Table of Contents

Introduction

Project description

The aim of the project is to use AI for state estimation; in this case we will implement some NN architectures to predict changes in heading angle and displacement using as data IMU data recorded using ROS gazebo simulator.

We will use TurtleBot3 as mobile robot to get the measurements inside the Gazebo environment; for convenience, we created the python script *turtlebot3_waypoints.py* to record the bagfiles from the simulations.

In this script we will save the output as bagfile, then it will be indexed to reduce the its size; finally we will pass as input to the NN 10 runs as training data and then the network will be evaluated on a new run to predict the waypoints.

Steps I

To summarize, we will do the following steps:

1. Launch TurtleBot3 inside Gazebo.

```
cd ~/catkin_ws
catkin_make
source devel/setup.bash
export TURTLEBOT3_MODEL=burger
roslaunch turtlebot3_gazebo turtlebot3_world.launch
```

2. Run the Navigation node to do Initial Pose estimation and to collect some of the surrounding environment information.

```
export TURTLEBOT3_MODEL=burger
roslaunch turtlebot3_navigation turtlebot3_navigation.launch
    map_file:=$HOME/map.yaml
```

Steps II

3. Launch the script *turtlebot3_waypoints.py* to record the bagfile from the simulation.

    ```
    roslaunch turtlebot3_waypoints turtlebot3_waypoints.launch
      bagfile:=/home/lorenzo/runs/run.bag
    ```

    In our case we will use 11 different waypoints arrays, thus we will end with 10 runs (which will be used as training data) and a test run.

4. Index all the obtained bagfiles to reduce the size of them.

    ```
    rosbag reindex run.bag
    ```

5. Launch the NN with the obtained run and record the results.

Now, we will briefly report the used NN architectures and the hyperparameters; we have used LSTM and Transformer as general structure.

Architectures

Some details

The used architectures are:

- LSTM [2]
- Transformer [1]

As for the parameters:

- Adam as optimizer (also with weight decay $wd = 0.01$)
- learning rate $lr \in \{0.0002, 0.0001, 0.0005, 0.001\}$.
- step size $step\_size \in \{50, 100, 200, 300\}$.
- gamma $gamma \in \{0.01, 0.1, 0.2\}$.
- number of epochs $epochs \in \{10, 15, 20, 25, 50, 100\}$.
- batch size $batch \in \{16, 32, 64, 128\}$.
- hidden size $hidden \in \{6, 64, 128, 256\}$.

Measurements

Experiment I

We will briefly report a short table demonstrating the best results for each architecture; the complete results can be found in

model	epochs	batch size	train loss	test loss
LSTM	20	32	0.1668	0.3618
Transformer, dropout = 0.2	15	16	0.389	0.4464

As we can see, we obtained slightly better results using LSTMs; the runs were done mainly on CPU, thus the required time per epoch was always above 1-2 mins. We will now show the obtained graphs of the errors w.r.t. the number of epochs.

Note: All LSTM measurements were done using Colab standard CPU (Intel Xeon CPU with 2 vCPUs and 13GB of RAM), while Transformer ones were done on a M1 Pro 10 core.

Experiment II

Results with LSTM:

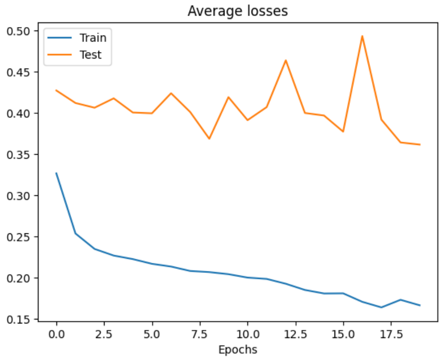

Figure 1: Errors

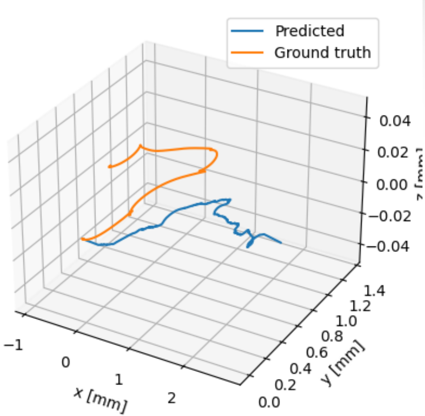

Figure 2: Estimated trajectory.

Experiment III

Results with Transformer:

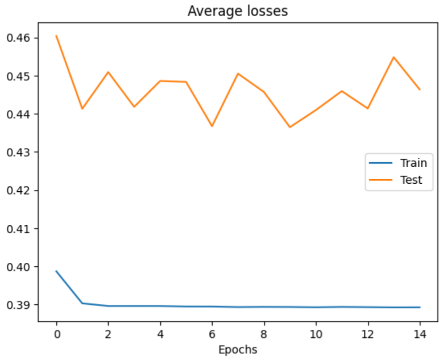

Figure 3: Errors

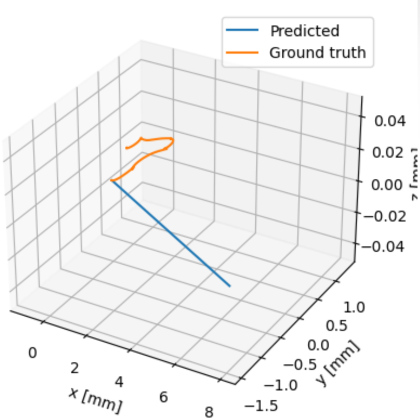

Figure 4: Estimated trajectory.

Conclusions

To conclude

In conclusion, while the experiments testing state estimation using LSTM and Transformers resulted in suboptimal results, the process proved invaluable in gaining a deeper understanding of their functionality.

We did various adjustments of hyperparameters and architectures (switching between the two), thus we have enhanced our comprehension of their capabilities and structure.

References I

[1] Ashish Vaswani et al. “Attention is all you need”. In: *Advances in neural information processing systems* 30 (2017).

[2] Yong Yu et al. “A review of recurrent neural networks: LSTM cells and network architectures”. In: *Neural computation* 31.7 (2019), pp. 1235–1270.