Google Colab Code 3 Notes

Authors Notes:

Because I am writing this as I create all my different variations of models and stuff, in this note we will focus on some of the major points/models/results instead of putting every single model and every single one of its results created because this would just create a very long PDF file. I hope all this will make sense.

Table of Contents

Chapter 1:

chapter 1 is less important chapter it mostly just gives context to the code and gives reason why we forgo the x dot equation as part of our losses moving forward. chapter 1 code is given in the github repo under chapter 1 code, some tinkering may need to be done but try to follow up until after the training loop portion of the code.

Chapter 2:

goes over how we avoid the expensiveness of using the x dot equation in favor for the analytical solution of x as part of the losses, and shows results from training on a dataset with 500 simulation runs

Chapter 3:

We will go over the model we obtained that got us our best results

Chapter 4:

We go over making modifications to the model to be ammendable to both parameter time varying/time invarying cases

---

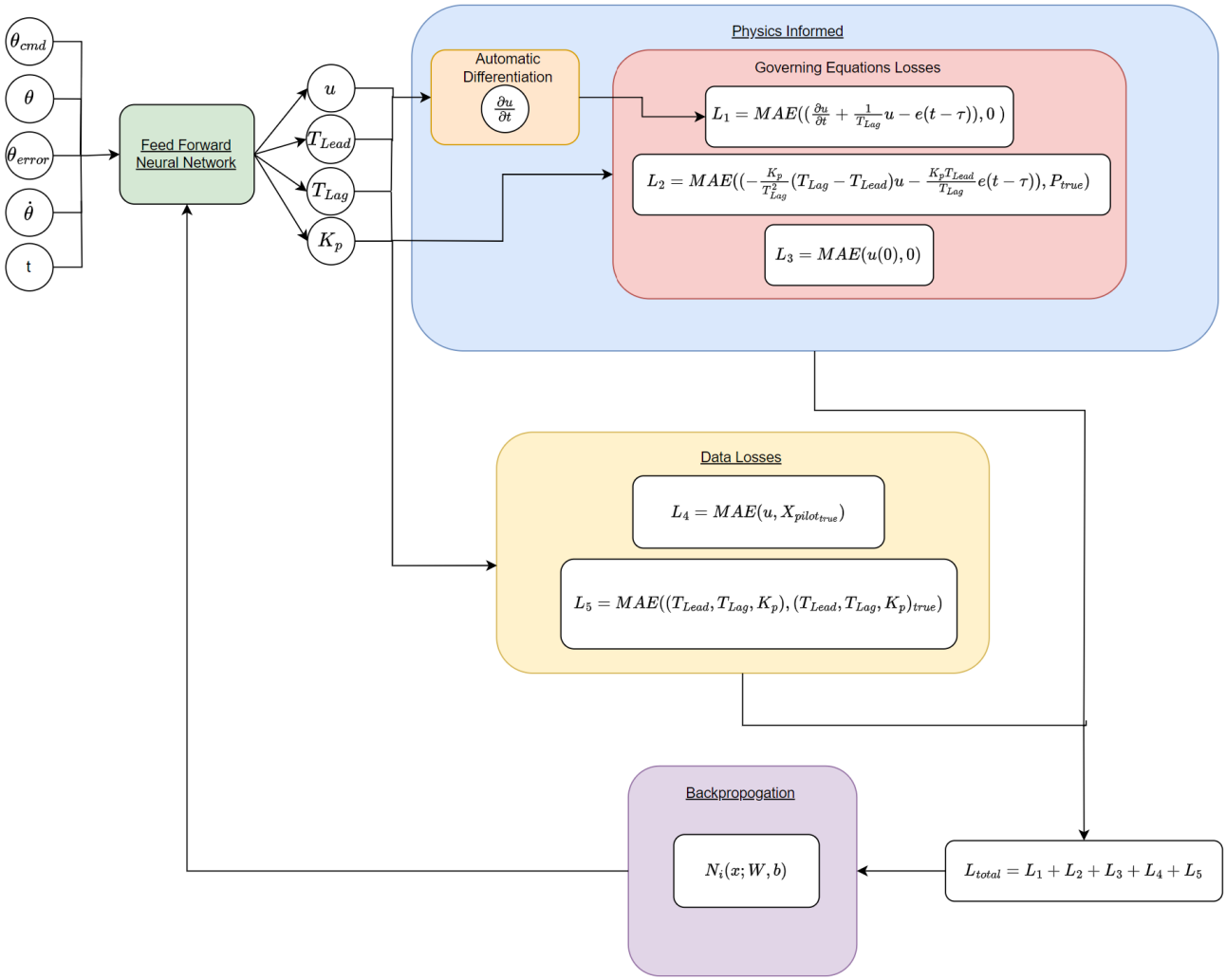
Chapter 1:

Intro

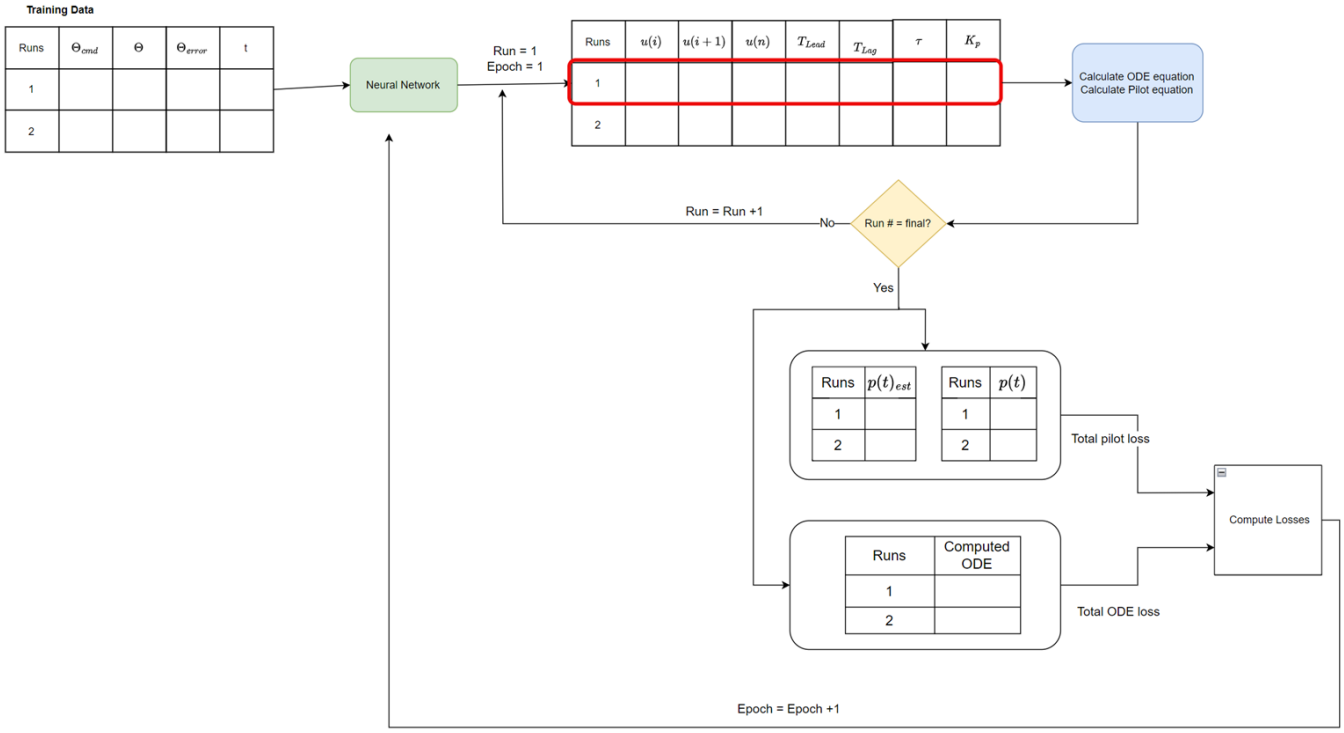
Most PINNS in the literature and through own experimentation are setup in such a way to solve their problem for a specific set of parameters/conditions. This approach can be seen in the previous notes/codes. They apply PINNS to problems such as the heat equation, navier stokes equations, Schrodinger equations, etc, and they apply PINNS for a specific set of conditions for these problems.
However for our problem we would like to not just setup our architecture in such a way that only 1 case can be solved for, we would like to create a model that uses the physics equations as part of the loss function and has the capabilities of being able to generalize to the input data to and properly estimate the x(t) and parameters for any run of data. For this we make some modifications to be able to extend PINNS to multiple simulation run datasets.

Architecture:

note: $\dot{\theta}$ is not used as one of the inputs thus far



Details on L_1, L_2 calculations see below:



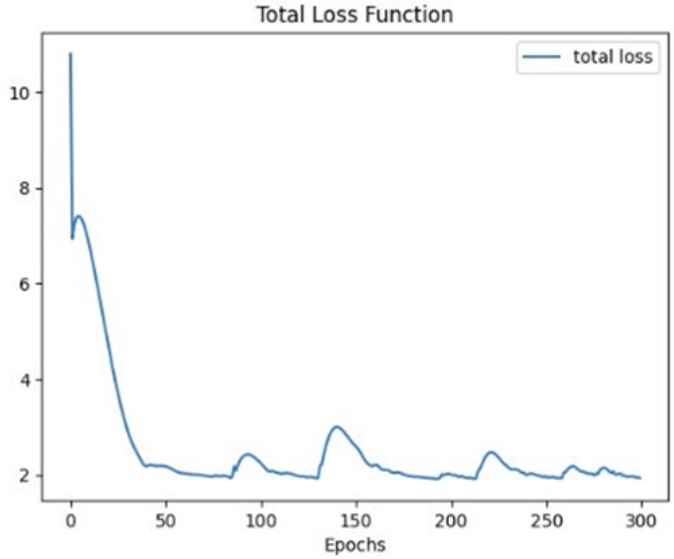
Analysis:

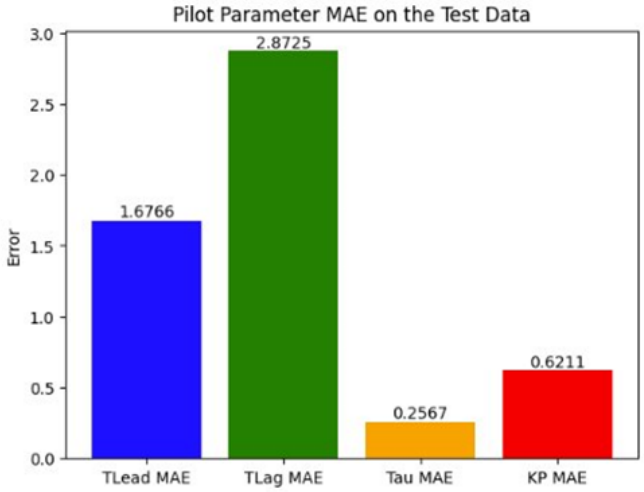
It was found that the ODE loss term is very computationally expensive to run, thus any attempts to do so with any substantial dataset size would result in a crash in the Google Colab. Because of this we experimented only with small datasets. The most we could really train for any not crash was about 25 simulation runs of data.

Results:

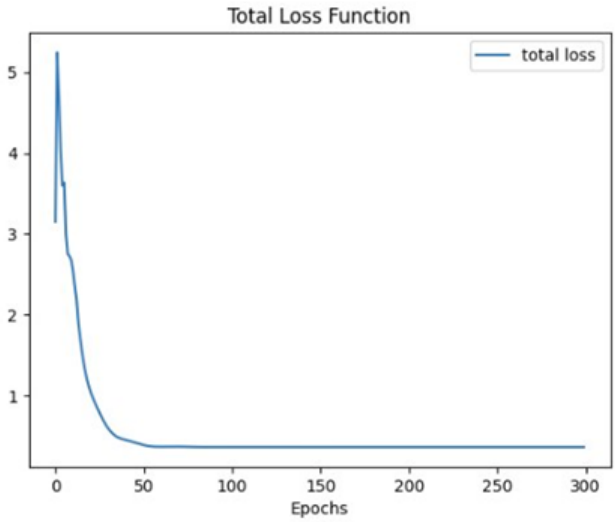
These results were so so, it is inconclusive if this approach would have worked just because our dataset was so small. The following results are from a dataset of 25 simulation runs, split into train/test set of 20/5, the loss function = $L_{params} + L_{xp} + L_{p(t)} + L_{ODE}$ so from the above architecture we just omitted the initial condition loss.

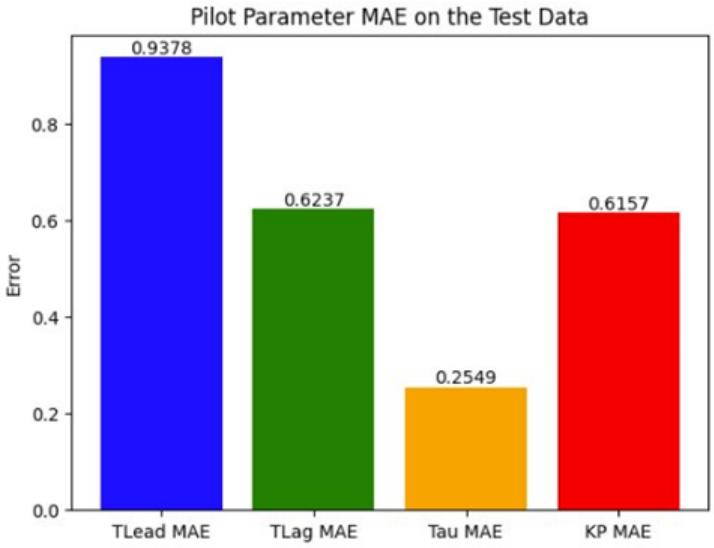
Trial 1:



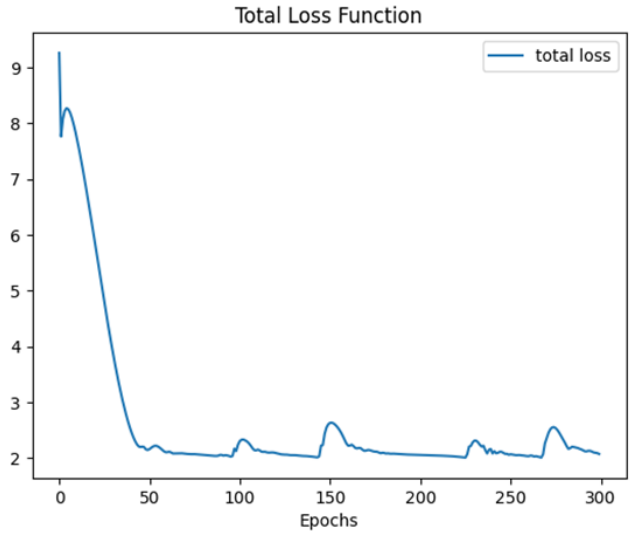


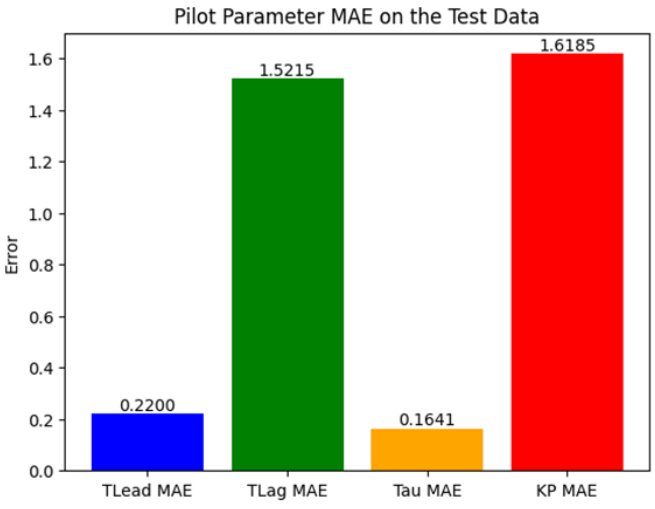
Trial 2:





Trial 3 (on 20 sets of data instead of 25):





Chapter 2:

Intro:

The point of PINNS/IPINNS is to use automatic differentiation to map the outputs of the standard neural network to the physics equations. However when working with a large dataset this can be computationally expensive to solve map u --> $\dot{u}$ for 100's of simulation runs. Since this requires the code to manually solve for the gradients. To avoid this and because of the less complexity of our governing equations we can instead opt to integrate our $\dot{x}(t)$equation with initial condition of $x(t=0)=0$ and instead utilize the analytical solution of x instead of the x dot equation as part of the loss function. This way our neural network will output the outputs we want and the loss function will be purely just arithmetic equations it has to compute which is much much cheaper than the previous configuration. See below for integration steps for analytical solution of x

$\dot{x}(t) = -\frac{1}{T_{Lag}}x(t) + e(t-\tau)$

$x(t) = Ce^{-t/T_{Lag}} + \frac{e(t-\tau)}{T_{Lag}}$

$x(t=0) = 0 = Ce^{-0/T_{Lag}} + \frac{e(t-\tau)}{T_{Lag}}$

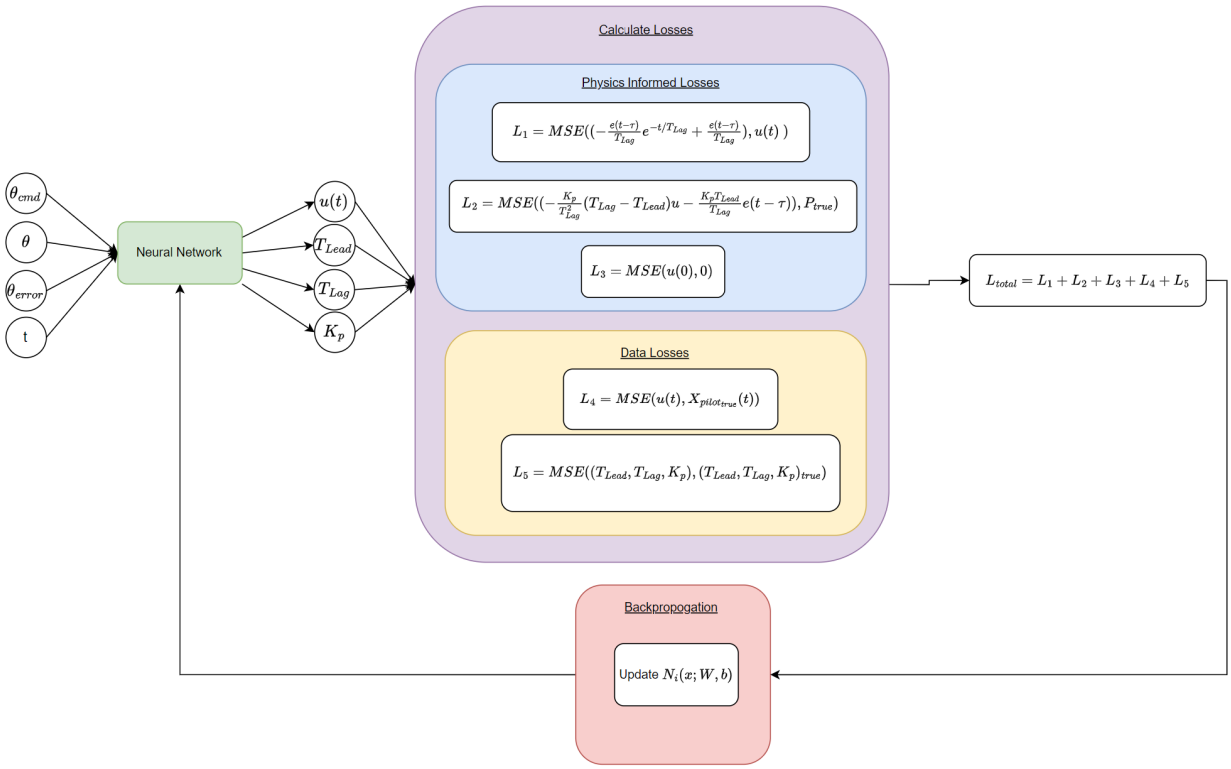
$C = \frac{-e(t-\tau)}{T_{Lag}}$

$x(t) = \frac{-e(t-\tau)}{T_{Lag}}e^{-t/T_{Lag}} + \frac{e(t-\tau)}{T_{Lag}}$

Architecture:

The updated L1 term from previous codes is now,
$L_1 = MSE((\frac{-e(t-\tau)}{T_{Lag}}e^{-t/T_{Lag}} + \frac{e(t-\tau)}{T_{Lag}}), u(t))$

The updated architecture will look like,

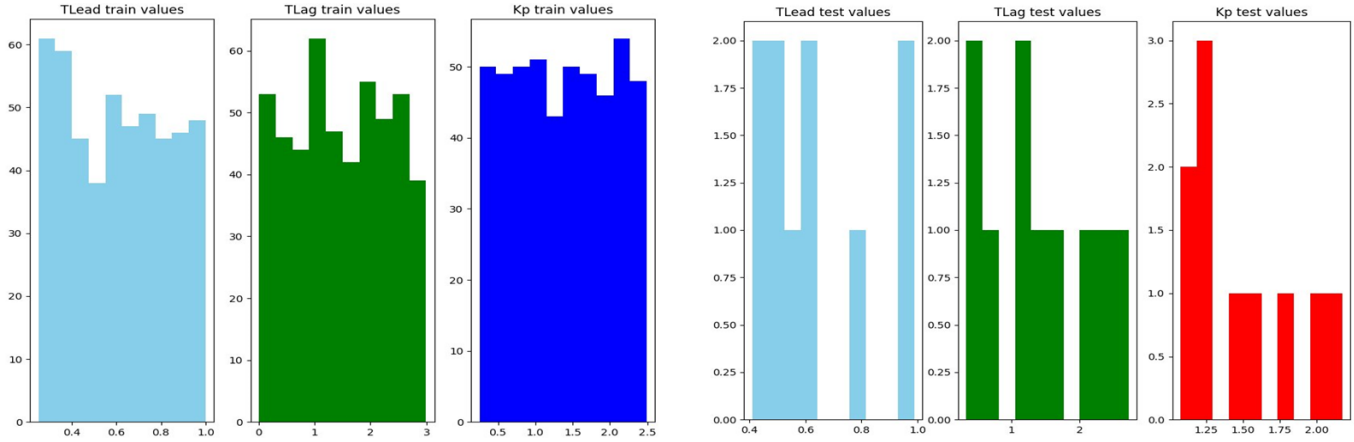


just to note because we solve for the analytical solution of x with initial condition equal to zero we can get rid of/omit L3 which is the initial condition loss term. With this updated architecture we will forgo the automatic differentiation because it is computationally expensive on large datasets and instead go for the analytical x solution.

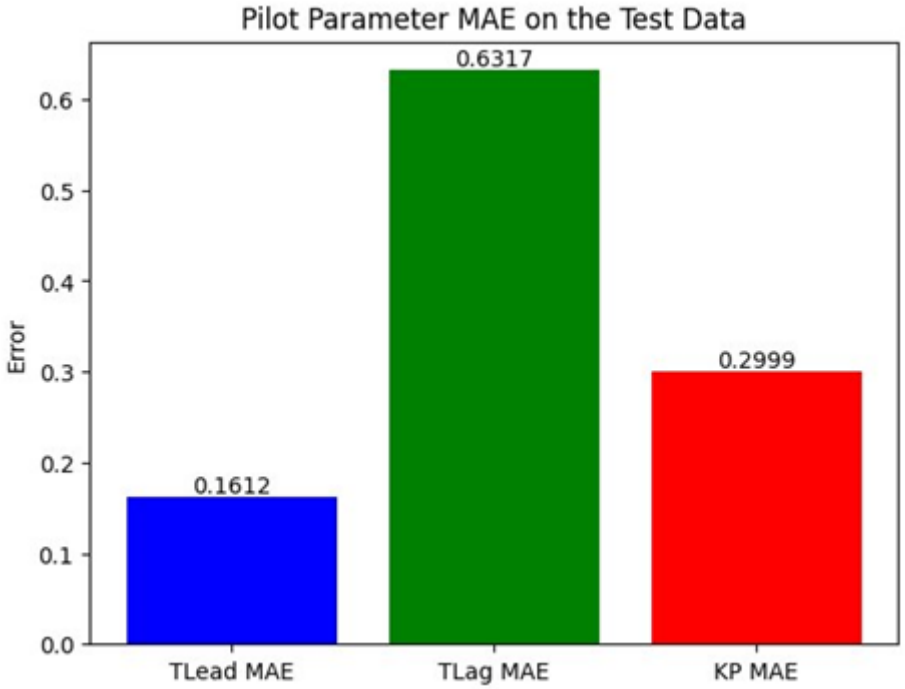
Training Setup:

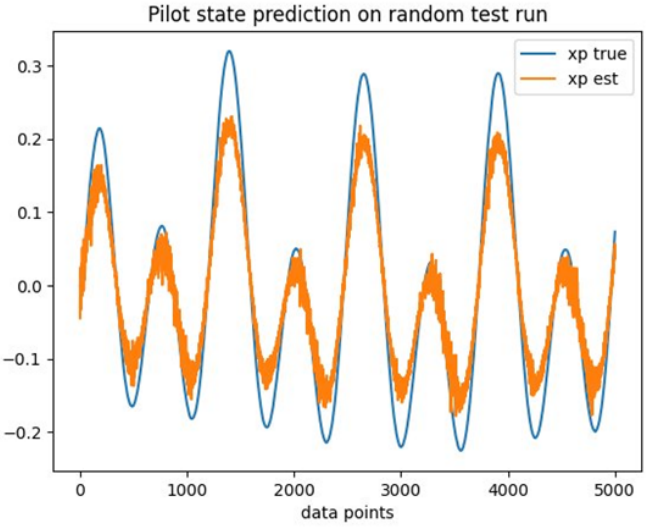
1- Load in 500 simulation data runs (490/10 train/test)
2- Pitch commanded: sum of sines
3- Dense layers model, 100 epochs
4- True pilot generated parameters in simulation: 0.1 < TLead <1 , 0.1 < Tlag < 3 , 0.1 < Kp < 2.5

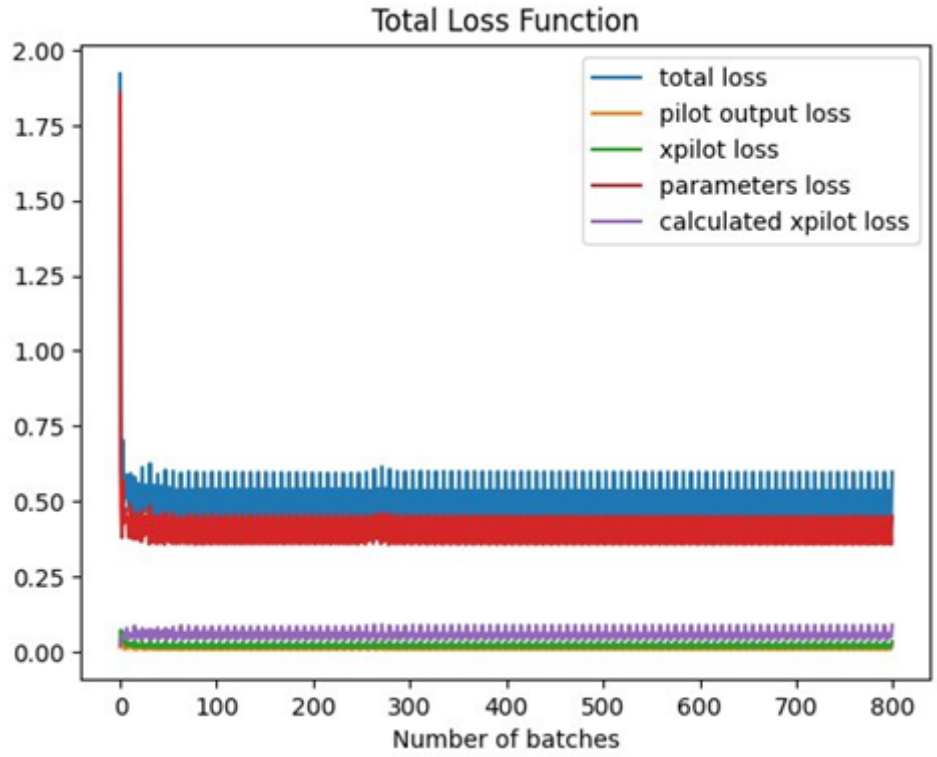
5- Loss = L1+L2+L4+L5



Results:







Analysis:

A common thread from all the other results not shown in this note is that the parameters loss term very much dominates the loss as seen above, the parameters loss is essentially right below the total loss. We will look to ways to avoid this problem of the problem of multi scale loss terms where our loss function has multiple terms each with their own units and their own different scales.

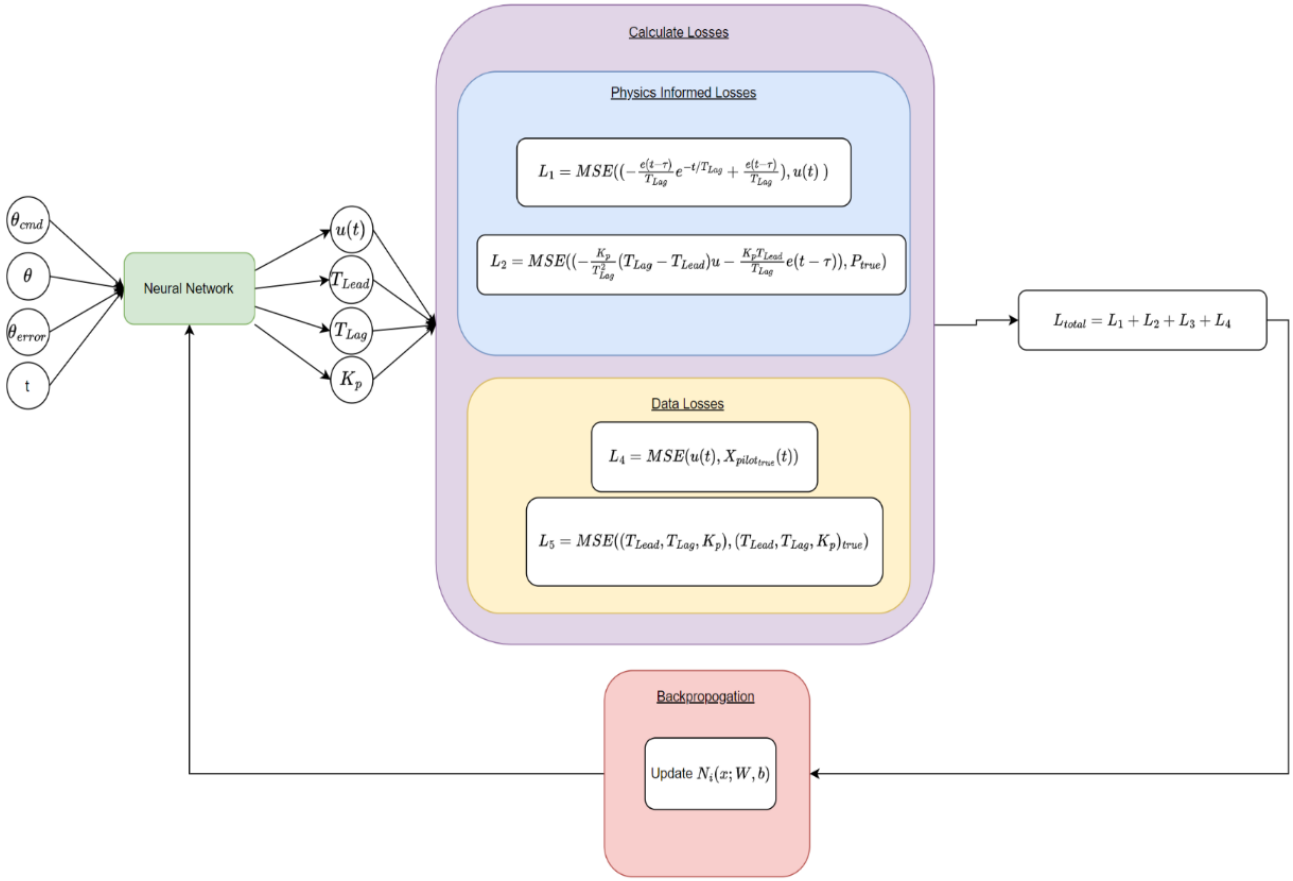
Another common thread is the lack of convergence of the pilot model parameters to their true values

Possible solutions:
-We will look to manually tune by adding a scale factor $\lambda = 10^{-3}, 10^{-2}$ or around there to the parameters loss term and see the impact
-Look at weight scheduling where depending on the epoch the scale factor to the parameters loss term may be minimal or may be 1, and vice versa with the other terms
-Optimiser scheduling as well

Chapter 3:

Whether it was a code error that was fixed or what this chapter shows our best results so far.

Architecture:



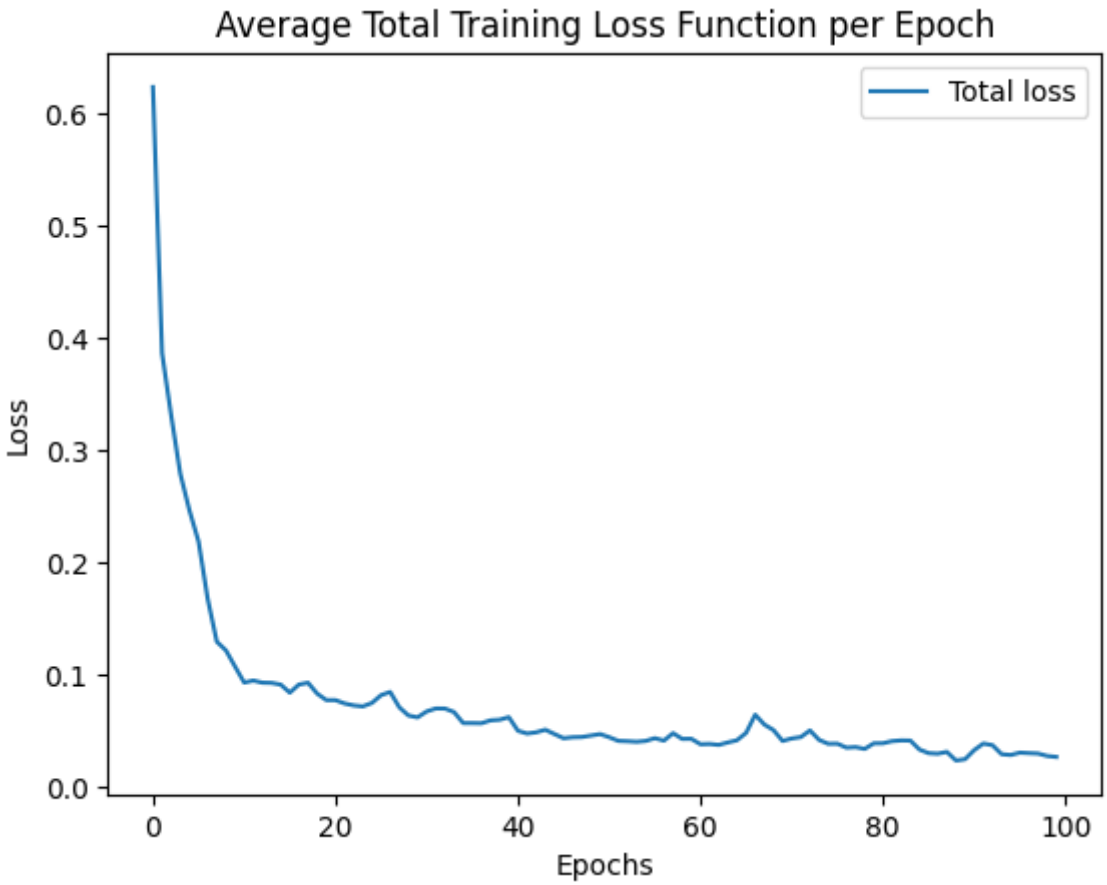
Neural Network

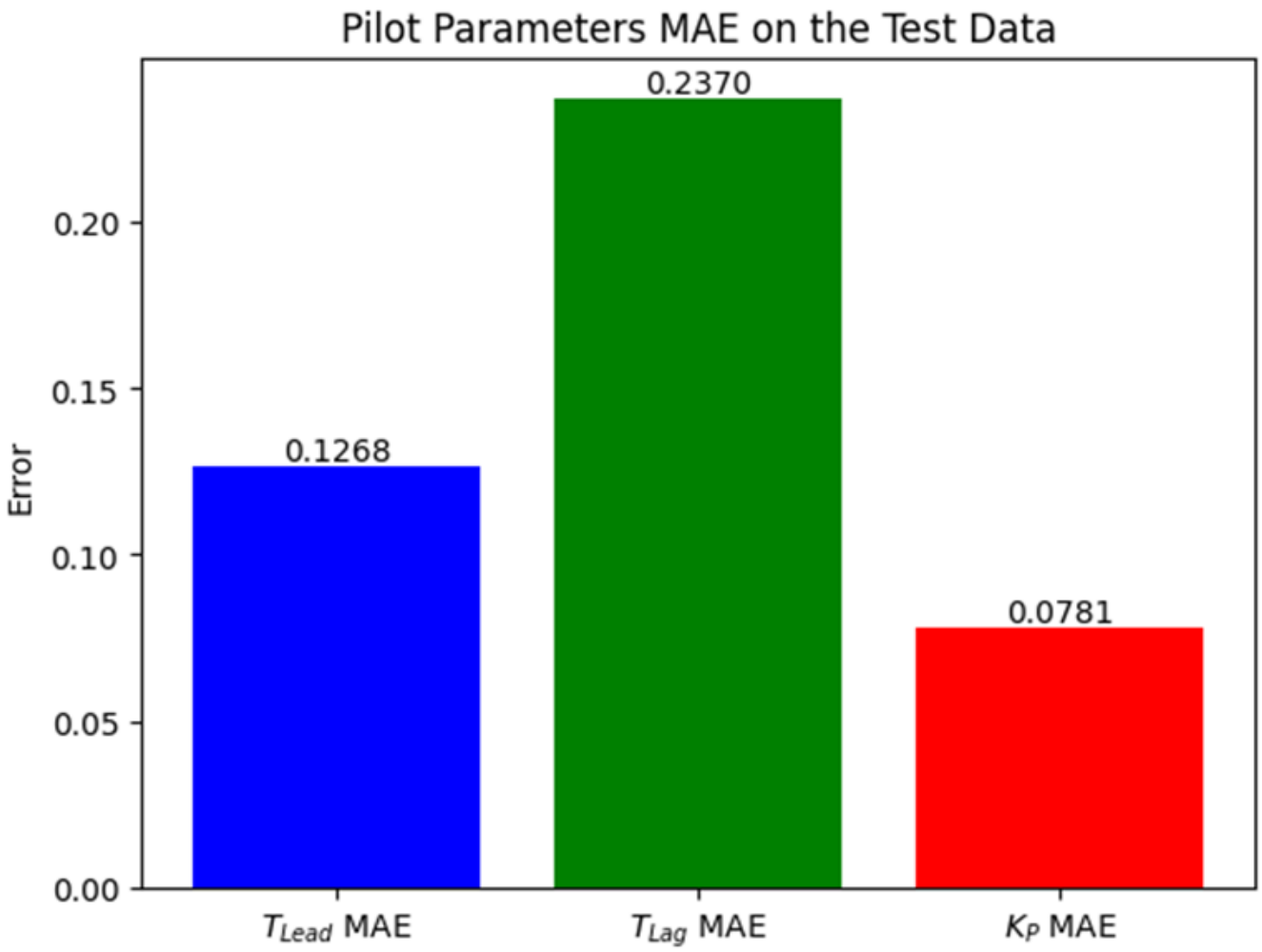
```
1-Input layer

2-hidden layer, 256 neurons, ReLU activaztion

3-hidden layer, 256 neurons, ReLU activation

4-hidden layer, 512 neurons, ReLU activation

5-Output Layer
```

Hyperparameters

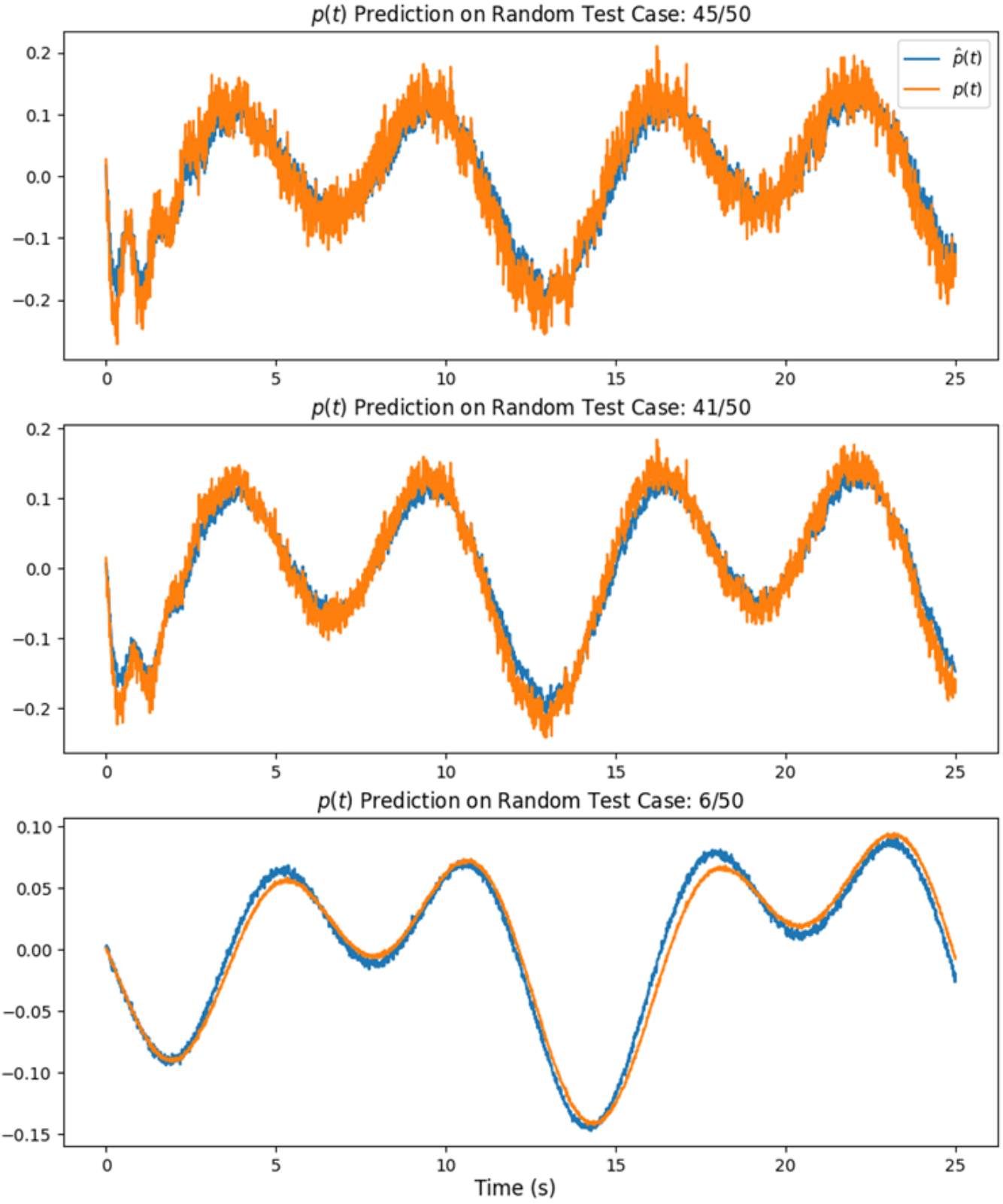
```
1-batch size=32

2-Adam optimiser LR=1e-3

3-trained for 100 epochs
```

Results

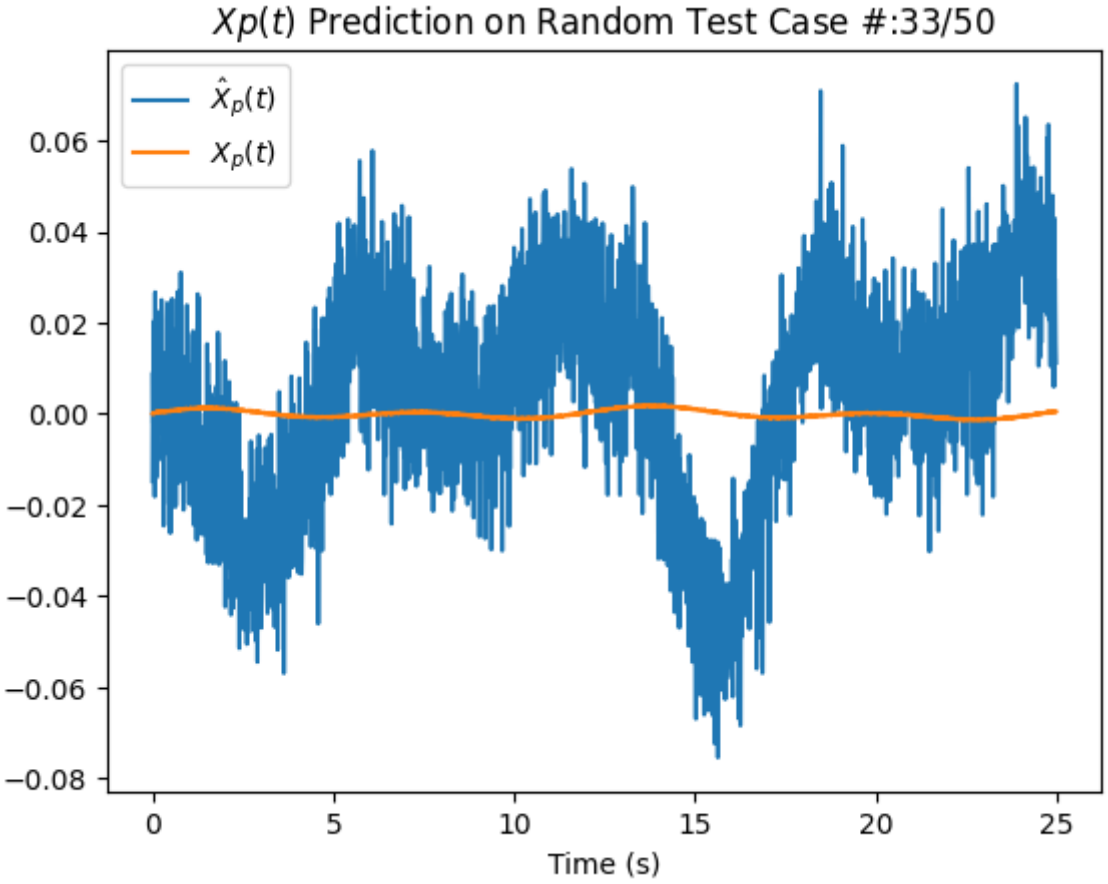

Average Total Training Loss Function per Epoch
Total loss
Loss
Epochs
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0
20
40
60
80
100

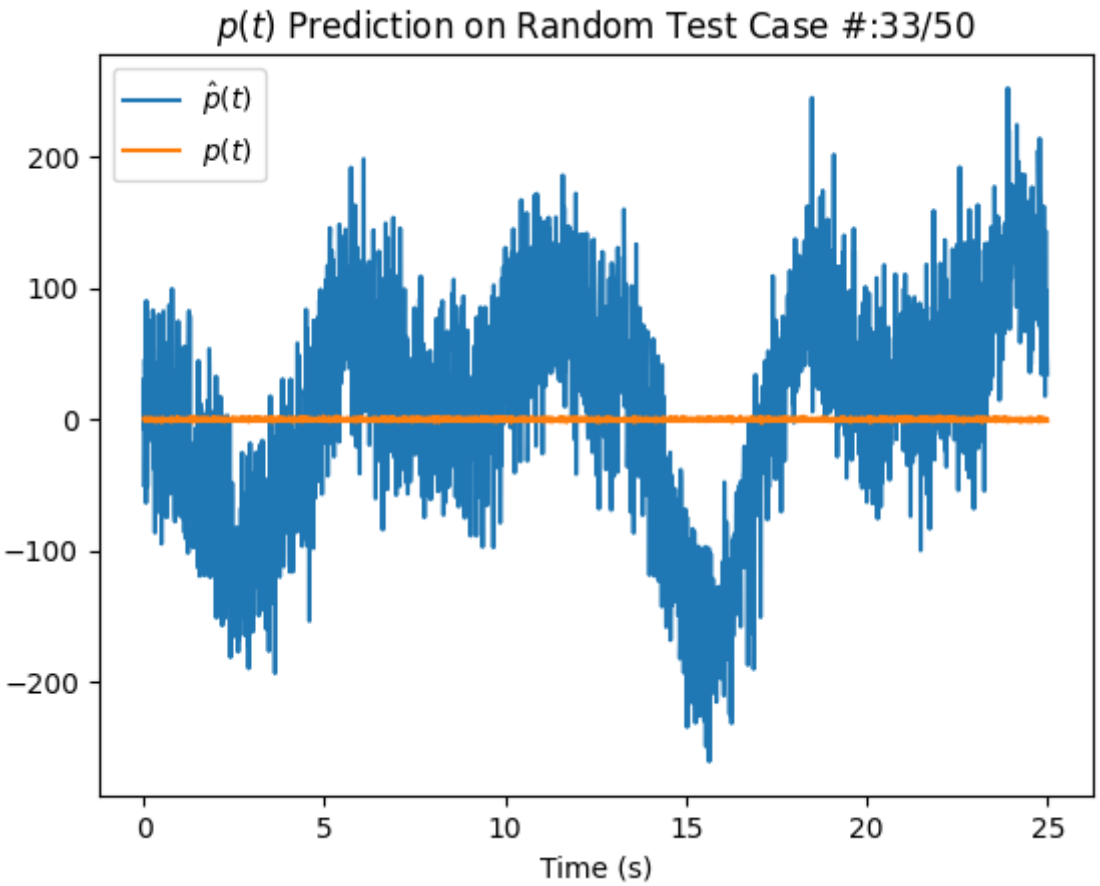

Pilot Parameters MAE on the Test Data
Error
0.00
0.05
0.10
0.15
0.20
0.1268
0.2370
0.0781
T_{Lead} MAE
T_{Lag} MAE
K_P MAE

Sample results:



-when looking at p(t) MAE on entire test set it was higher than anticipated and an outlier was found if we look at test case 33. so the MAE of x(t) and p(t) excluding run # 33/50 is. see below for outlier results


$Xp(t)$ Prediction on Random Test Case #:33/50
$\hat{X}_p(t)$
$X_p(t)$
0.06
0.04
0.02
0.00
−0.02
−0.04
−0.06
−0.08
0
5
10
15
20
25
Time (s)


$p(t)$ Prediction on Random Test Case #:33/50
$\hat{p}(t)$
$p(t)$
200
100
0
−100
−200
0
5
10
15
20
25
Time (s)

Excluding outlier
MAE(X(t)) = 0.0295
MAE(p(t)) = 0.0187

~~Including outlier~~
~~MAE(X(t)) = 0.017610705~~
~~MAE(p(t)) = 1.3878601~~

Analysis

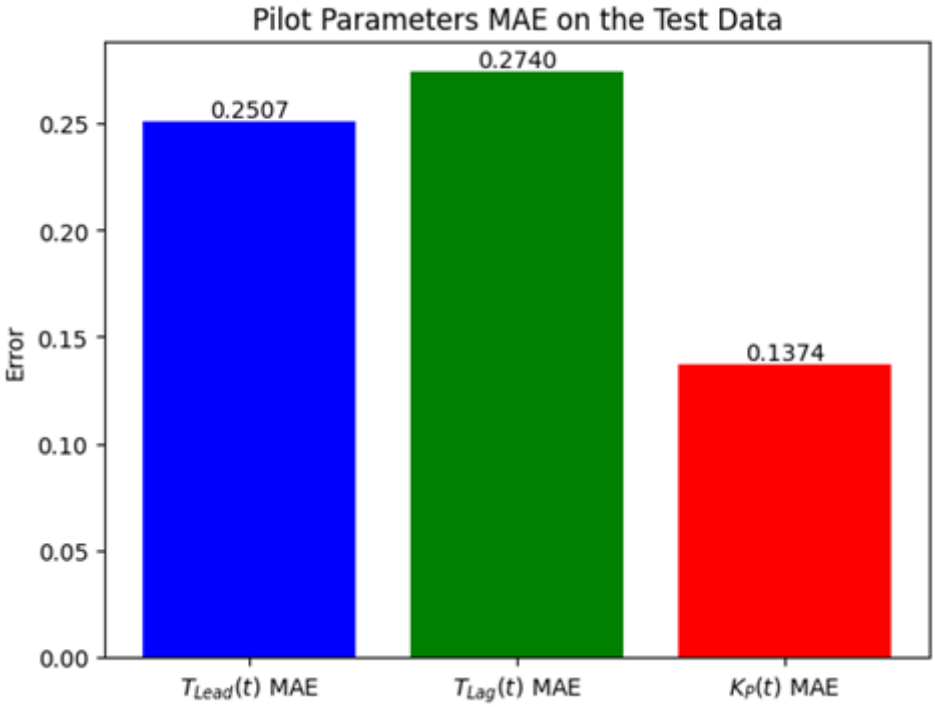
There was something with the x(t) calculated loss term that would equate to inf for the first few epochs. It was believed that this could be avoided if the loss term was not instantiated at epoch=0 but instead we have it so that loss term is only included after the 10th epoch after the network has been reasonably initialized to avoid this inf issue

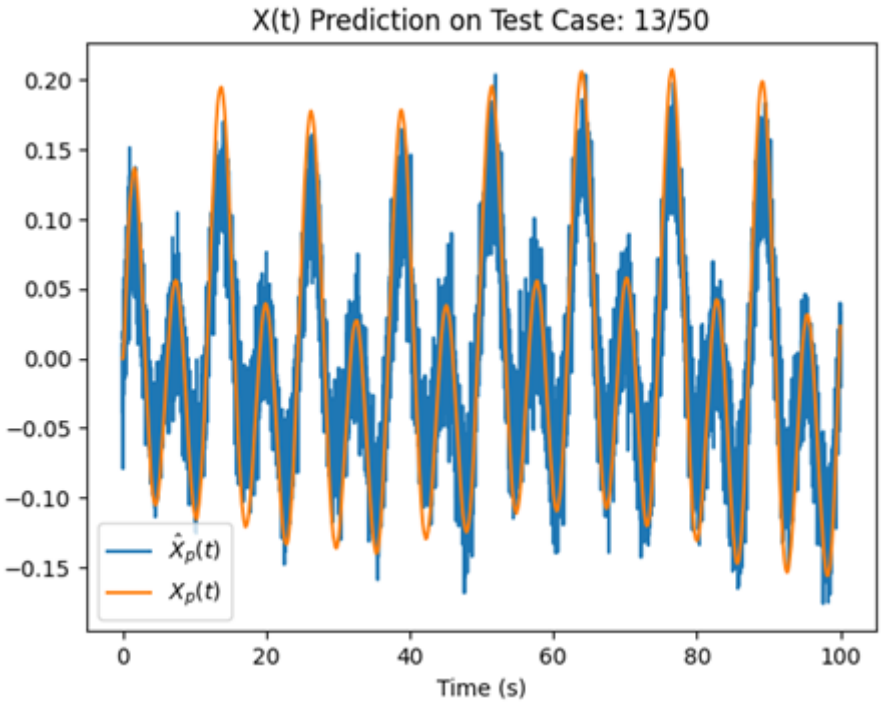
Chapter 4: Time varying + time invarying cases

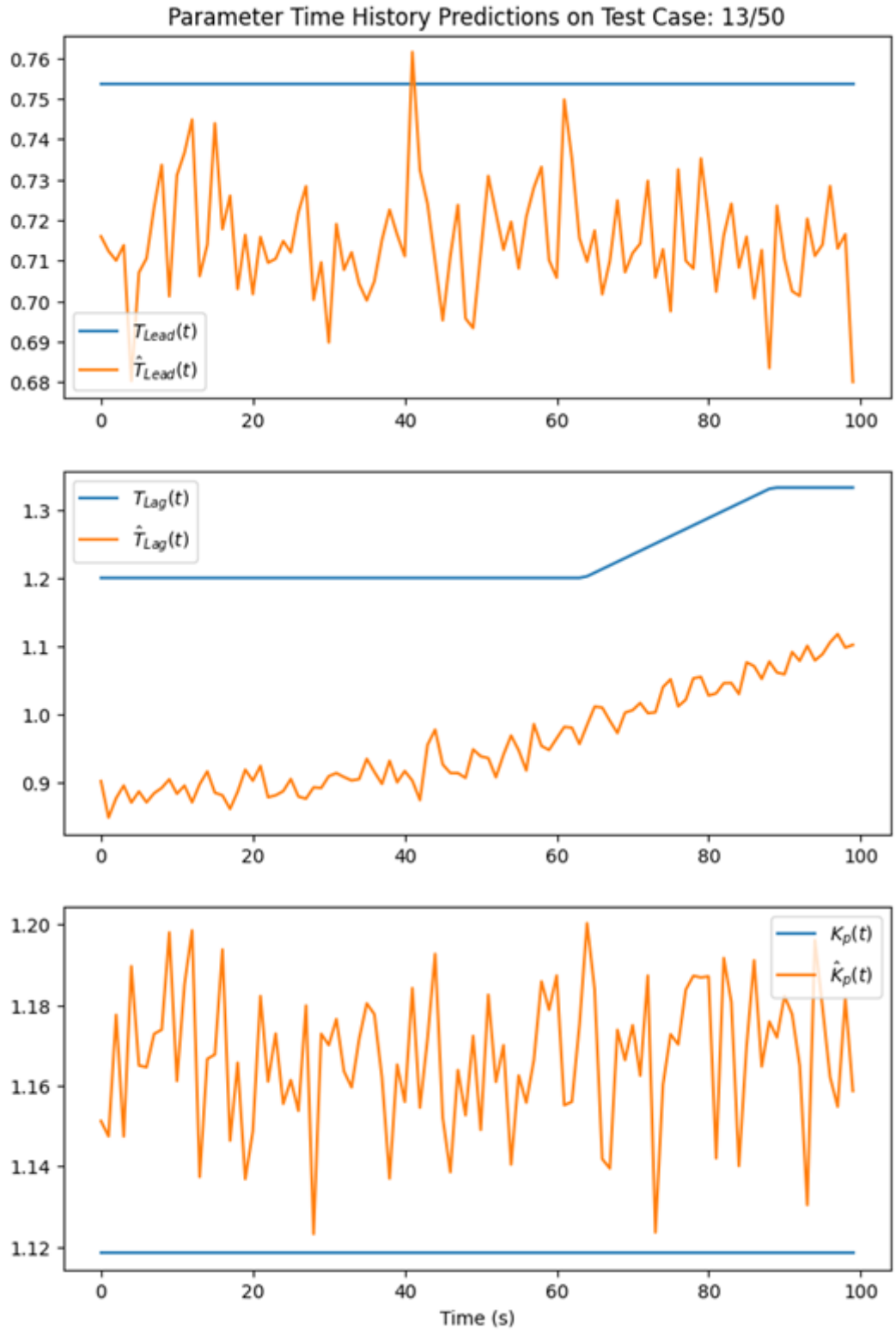
Right now all I have worked with is just time invariant cases, where our neural network outputs our predicted pilot state (nx1) and the 3 parameters (3x1) to create an output size of n+3. All we will do for the time varying case is estimate the parameters time history. The architecture is the same the only thing we are changing is the output size is now:
pilot state: nx1
parameters: nx3
total output size = 4n

4.1: Time varying TLag

First we decided to look at only time varying TLag. We do this because TLag is the only parameter that plays a role in the pilot state equation so its value directly ties to the pilot state, was the thought. Although it may be unrealistic to expect the pilot's behavior to change within 100 seconds, this was just done to create a framework of estimating these changes as creating hundreds of simulation runs consisting of minutes of data inorder to create something realistic would become very expensive to obtain and train. Because of this we opt for 100 seconds.
To make it as random as possible the time at which the parameter changes is randomized to within a range, and the slope at which it changes at is also randomized to within a realistic range.
The results can be seen below with the same architeture from previous chapter:


Pilot Parameters MAE on the Test Data
Error
0.2507
0.2740
0.1374
0.00
0.05
0.10
0.15
0.20
0.25
$T_{Lead}(t)$ MAE
$T_{Lag}(t)$ MAE
$K_P(t)$ MAE


X(t) Prediction on Test Case: 13/50
0.20
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
$\hat{x}_p(t)$
$x_p(t)$
0
20
40
60
80
100
Time (s)



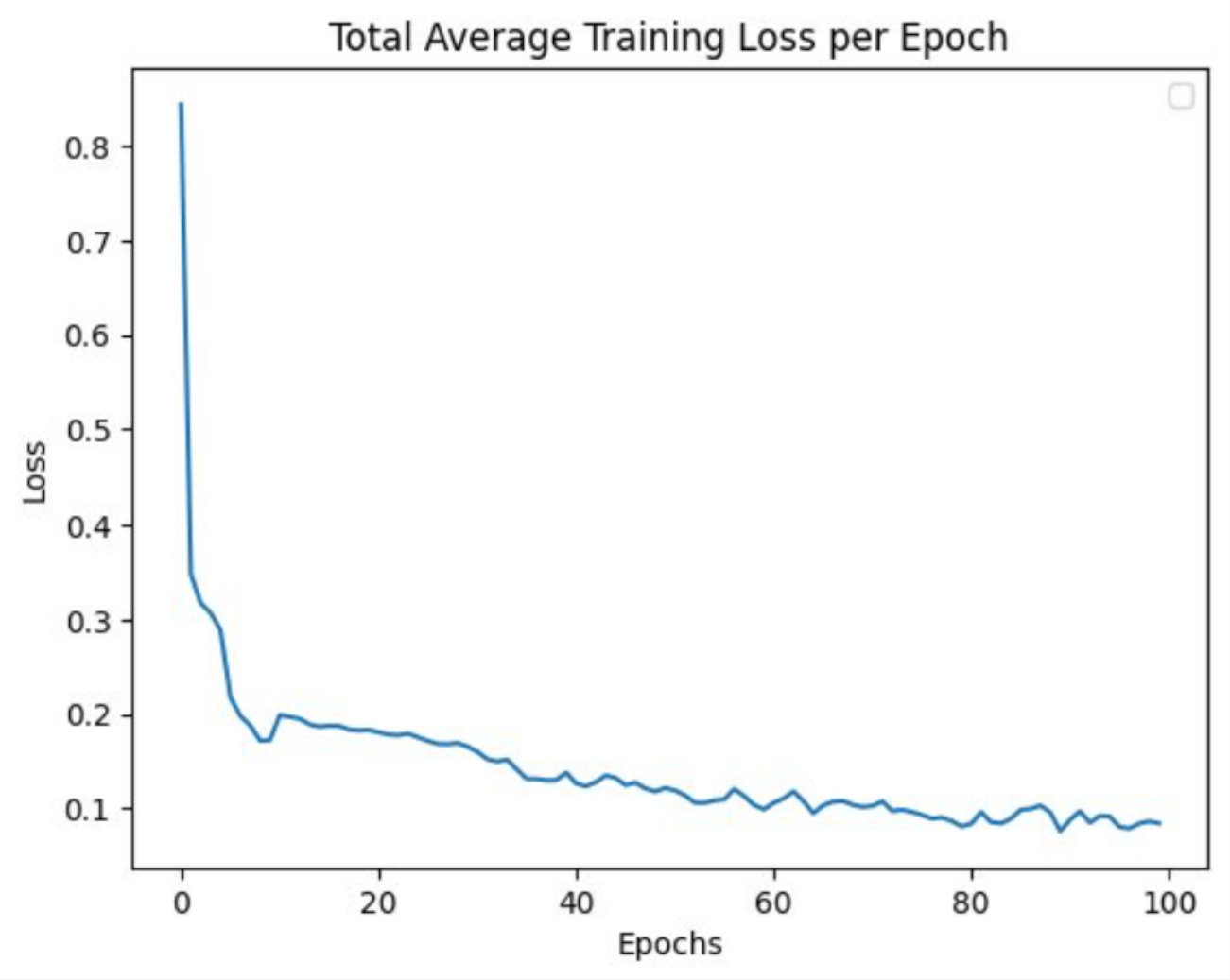
-X(t) MAE = 0.03308
-P(t) MAE = 0.0403

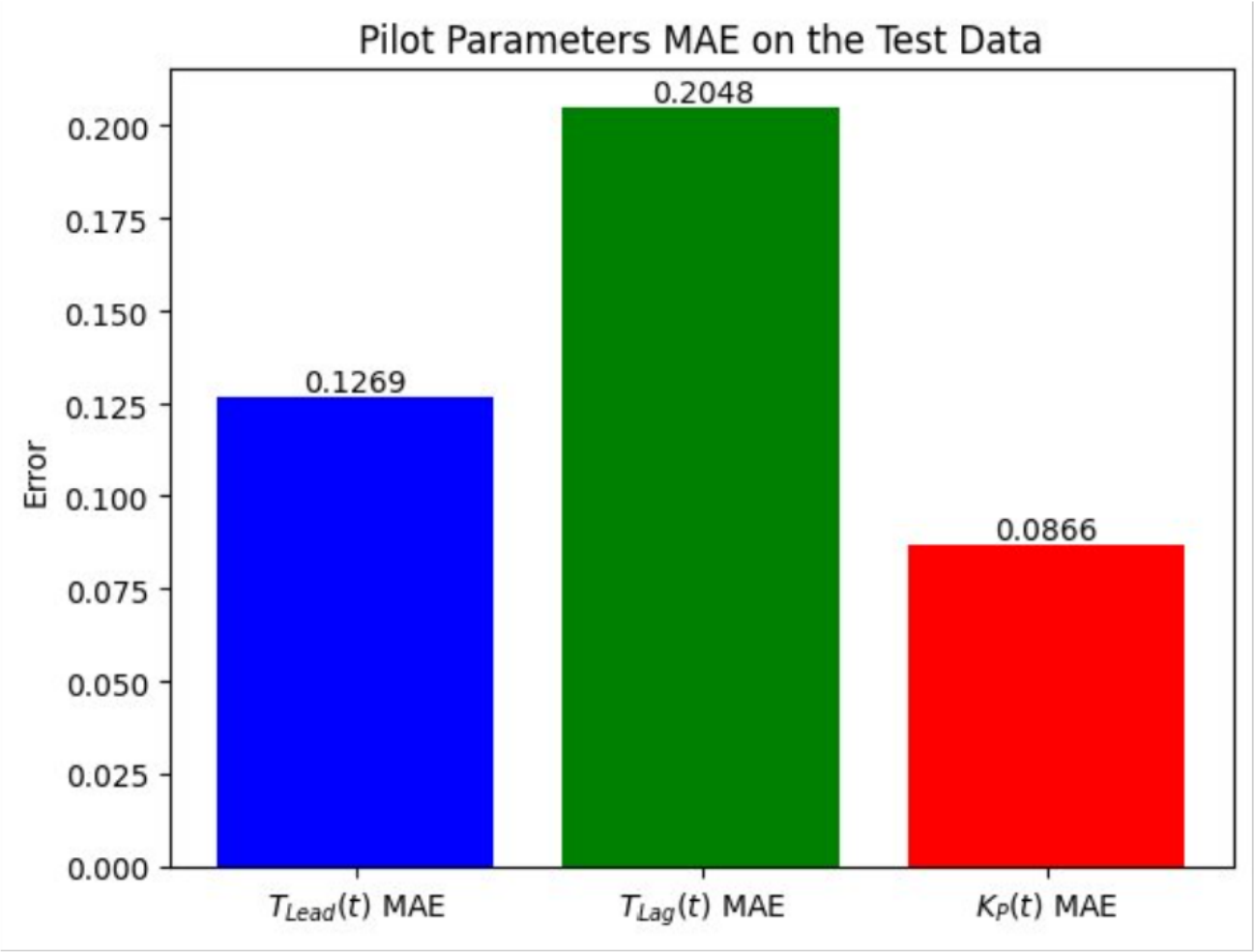
It can be noticed from above that the time varying signal consits of a constant portion, a flat portion and another flat portion. This configuration is consistent with the literature. Also notice that for the other 2 parameters that are time invariant their time history are just flat lines.

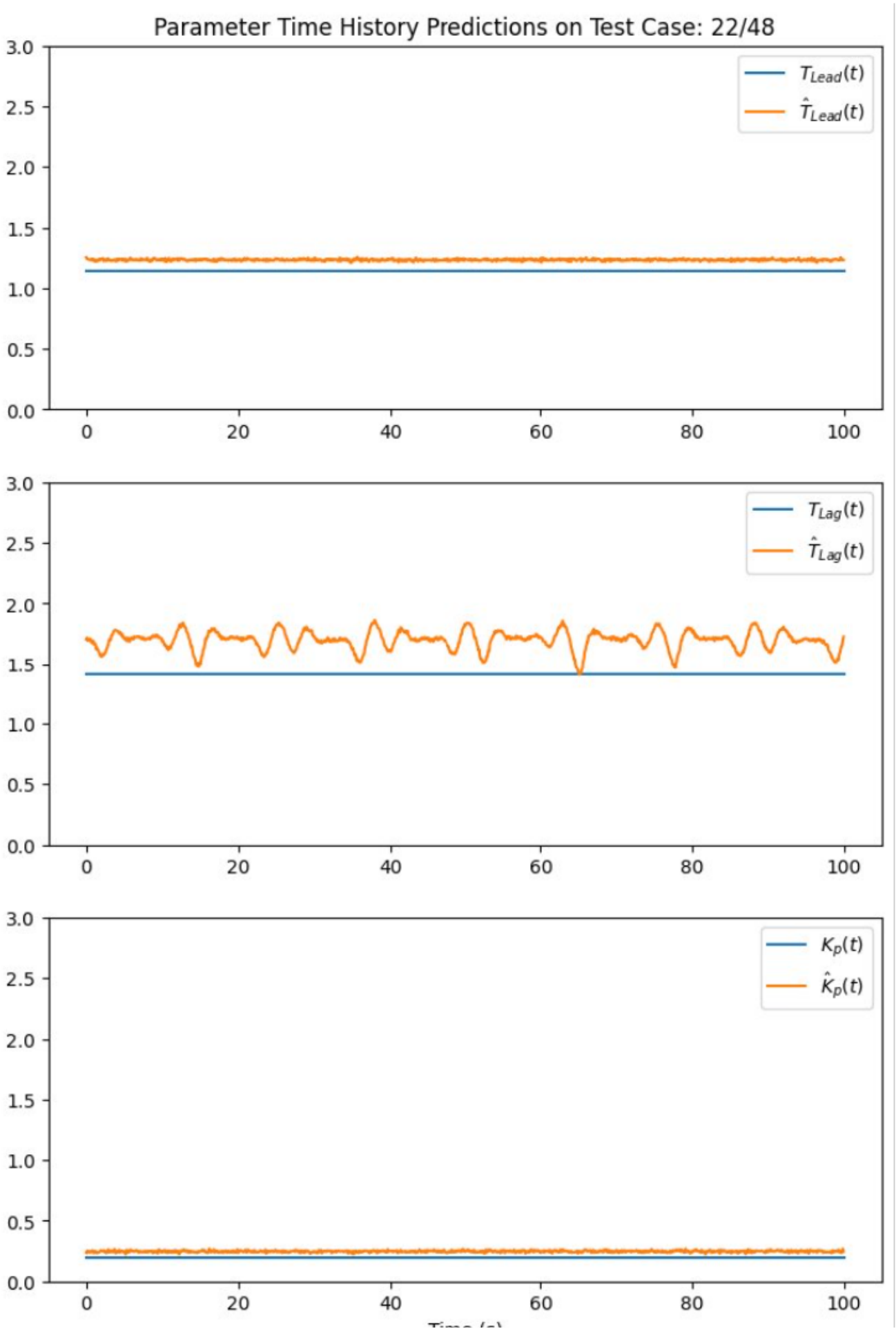
These results although not perfect provide a framework to which improve upon.

4.2 Time history of time invariant parameter

As previously stated we look to have 1 model to estimate variant and invariant parameters. So we test to see if our architecture can properly estimate the time history of constant parameters. The architecture remains the same. This training on ~500 test cases




Pilot Parameters MAE on the Test Data
Error
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175
0.200
0.1269
0.2048
0.0866
$T_{Lead}(t)$ MAE
$T_{Lag}(t)$ MAE
$K_P(t)$ MAE

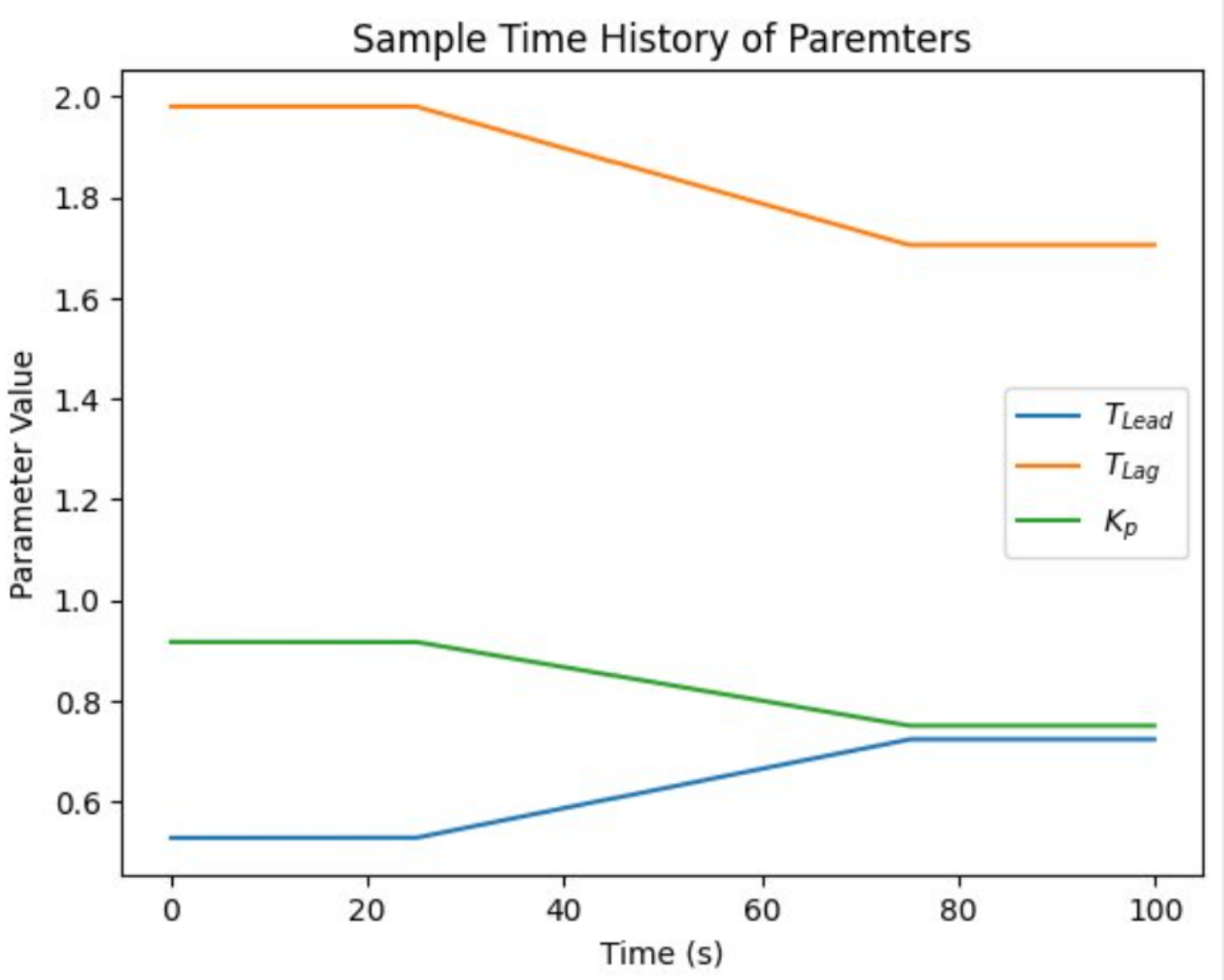

Parameter Time History Predictions on Test Case: 22/48
$T_{Lead}(t)$
$\hat{T}_{Lead}(t)$
$T_{Lag}(t)$
$\hat{T}_{Lag}(t)$
$K_p(t)$
$\hat{K}_p(t)$
0.0
0.5
1.0
1.5
2.0
2.5
3.0
0
20
40
60
80
100

Time (s)

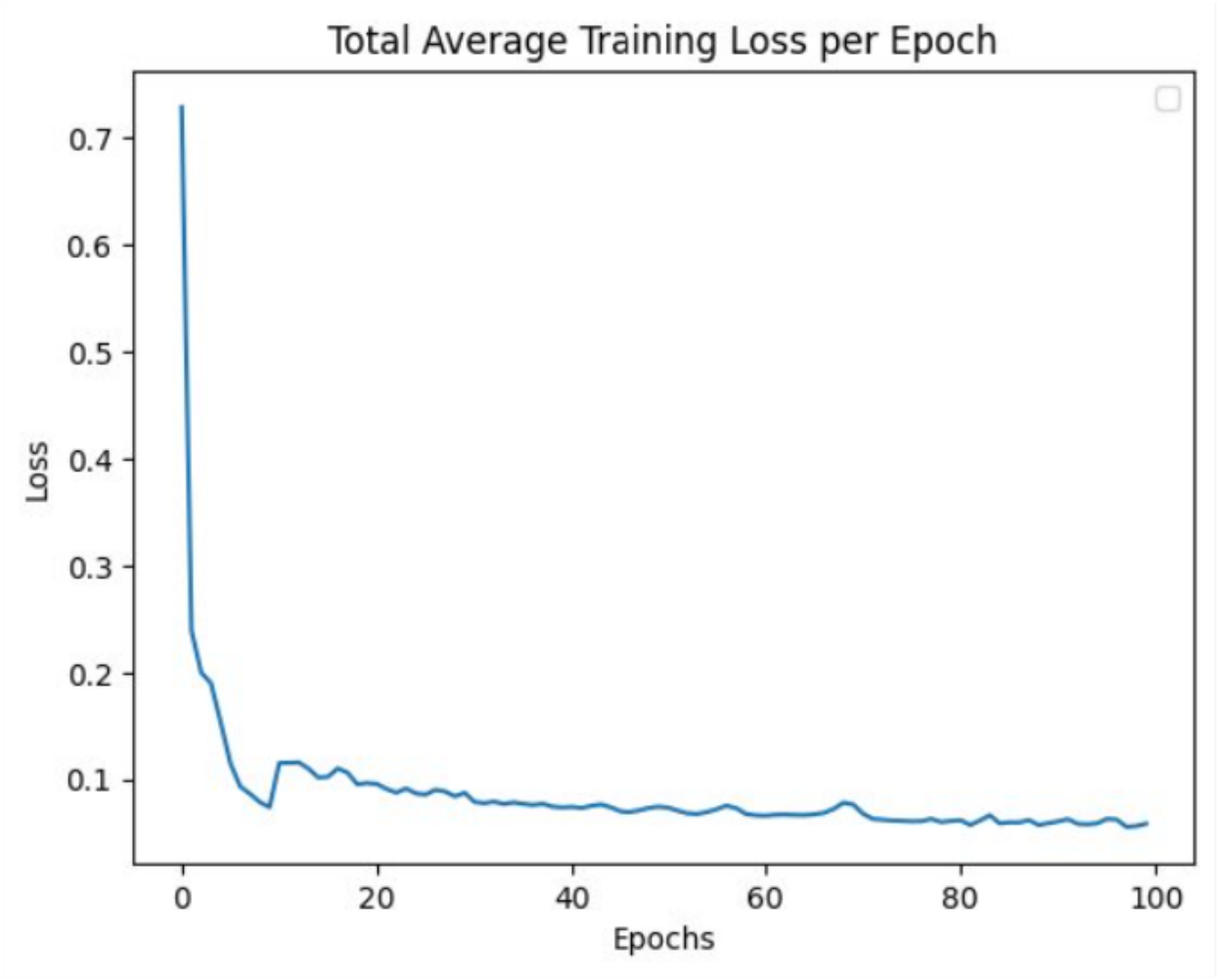
Pilot State MAE on Test Data	Pilot Output MAE on Test Data
0.0244609188	0.0103

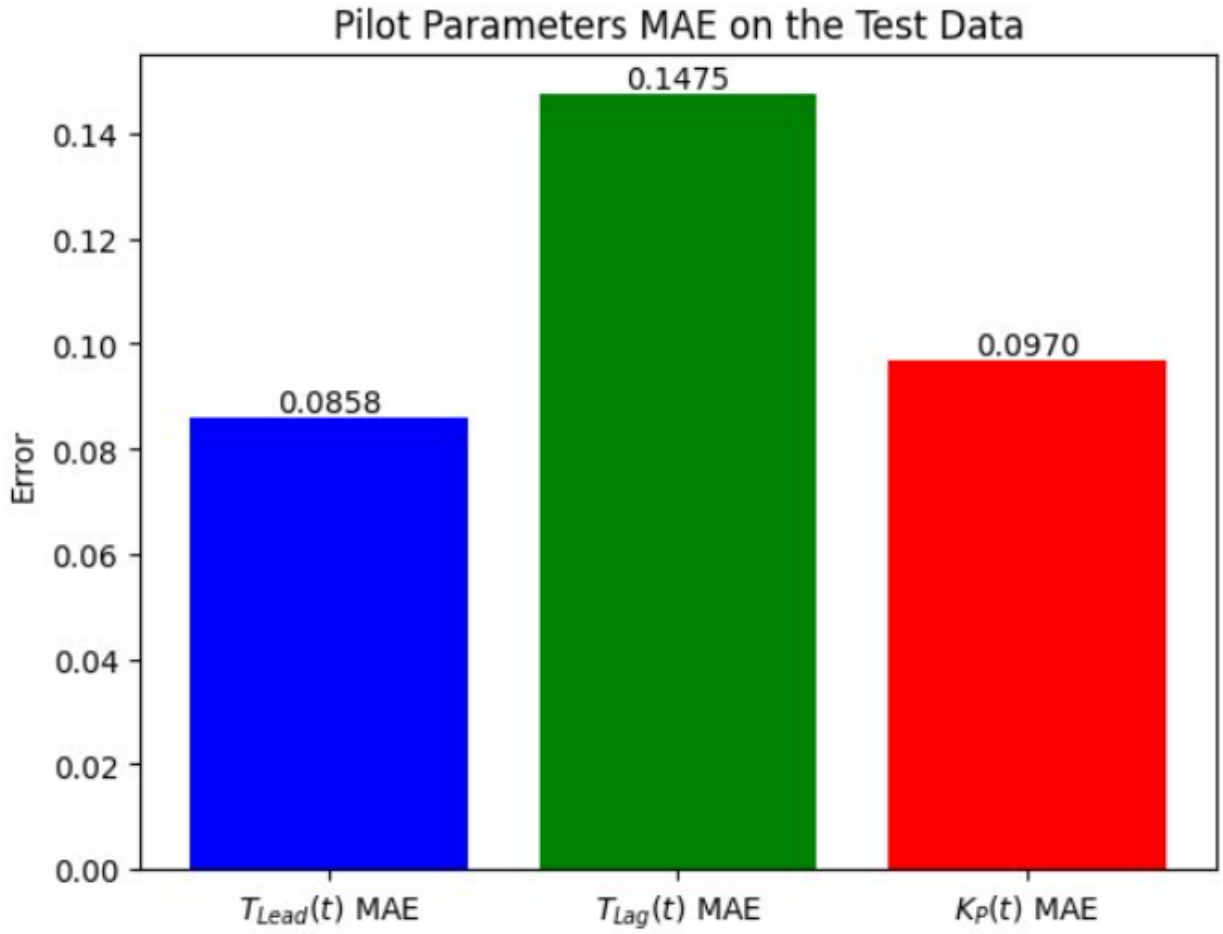
4.3 time history of time variant parameters

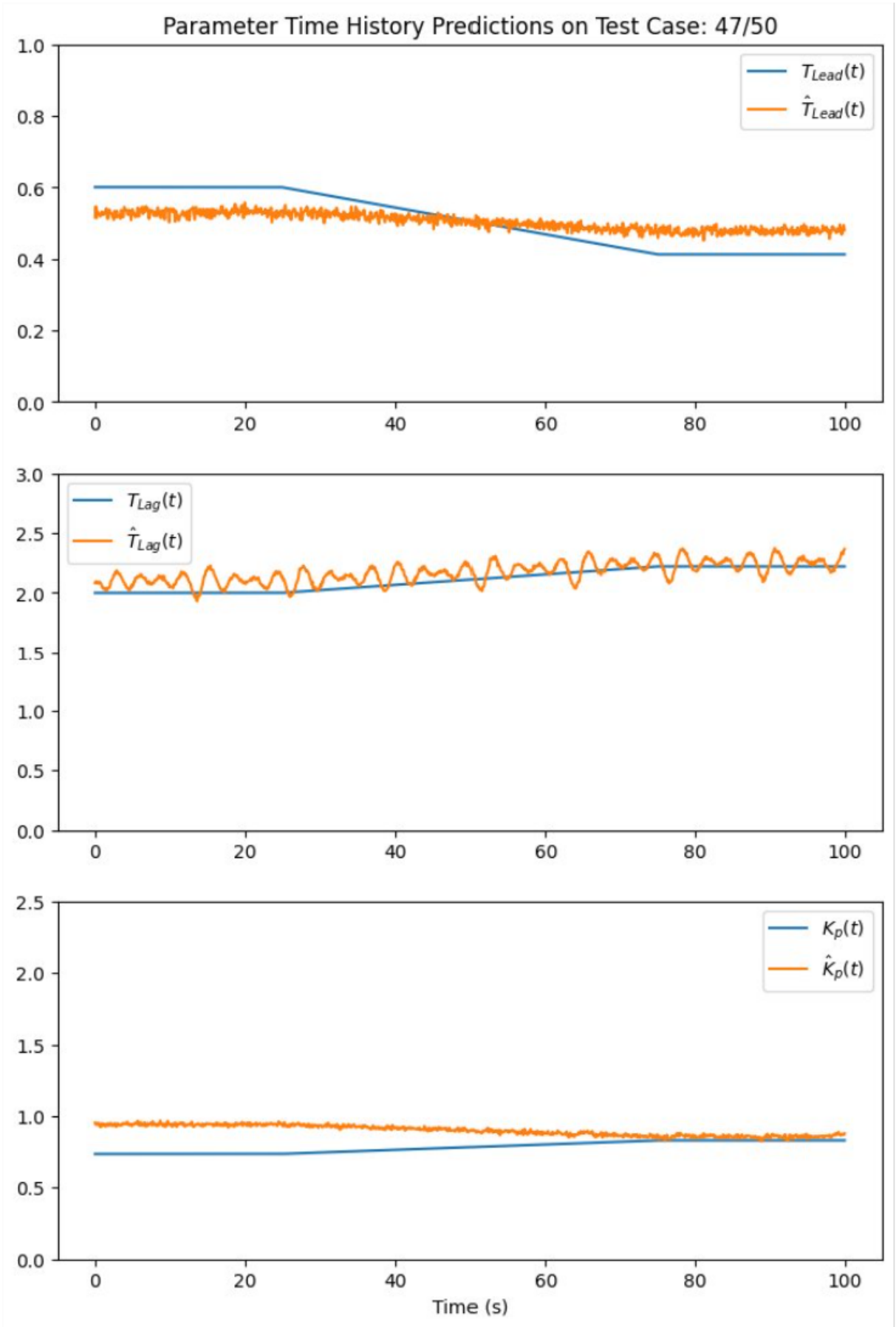
we saw some success predicting on data with only time varying TLag, but now lets try to have all 3 parameters be time varying. In the simulation there is already setup how to run time varying parameter cases, see sample below,



We will utilize the same architecture and algorithm to now estimate the time history of time varying parameters looking at 500 samples of data


Total Average Training Loss per Epoch
Loss
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
20
40
60
80
100
Epochs


Pilot Parameters MAE on the Test Data
Error
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0.14
0.0858
0.1475
0.0970
$T_{Lead}(t)$ MAE
$T_{Lag}(t)$ MAE
$K_P(t)$ MAE

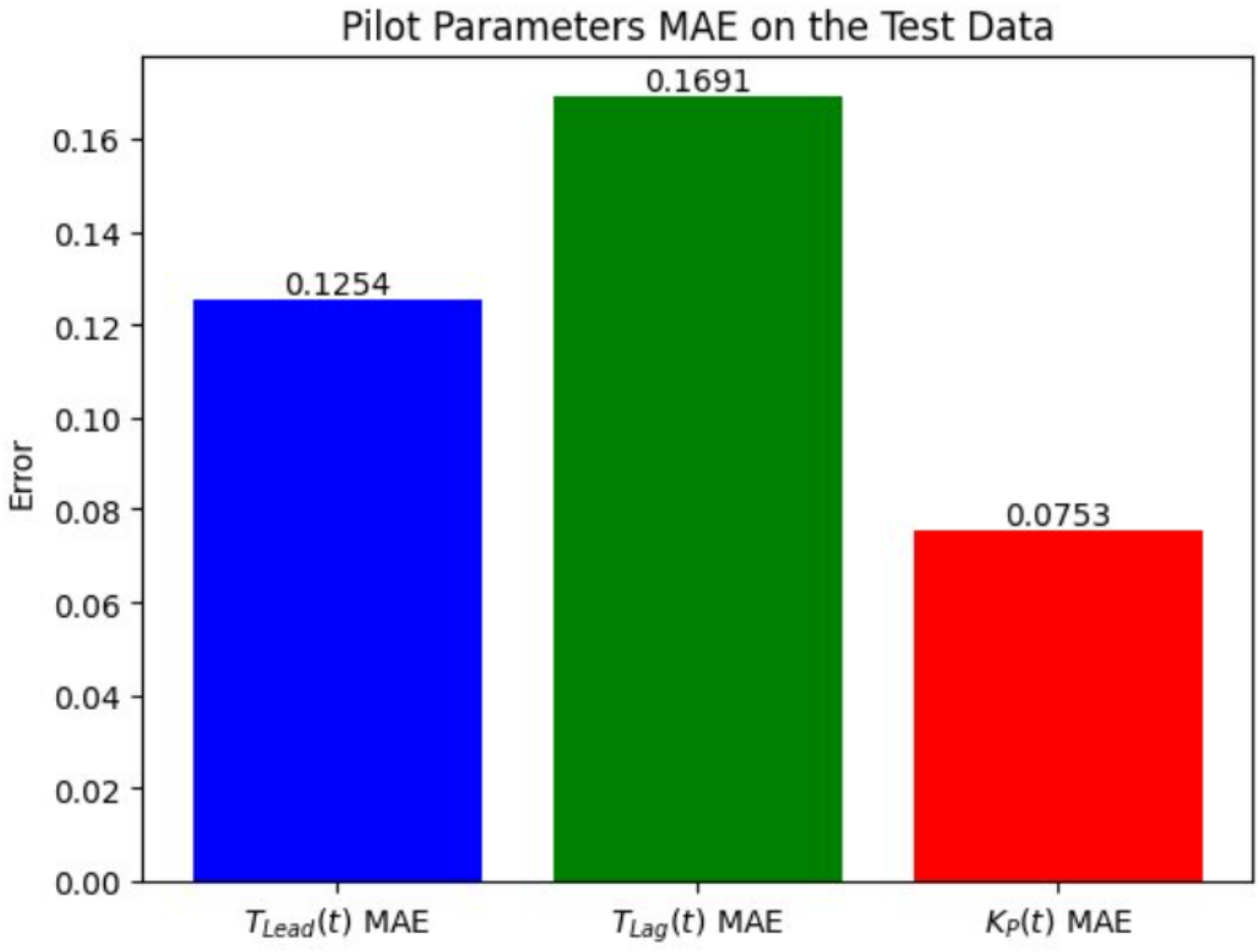

Parameter Time History Predictions on Test Case: 47/50
$T_{Lead}(t)$
$\hat{T}_{Lead}(t)$
$T_{Lag}(t)$
$\hat{T}_{Lag}(t)$
$K_p(t)$
$\hat{K}_p(t)$
Time (s)

Pilot State MAE on Test Data	Pilot Output MAE on Test Data
0.029833335	0.0145

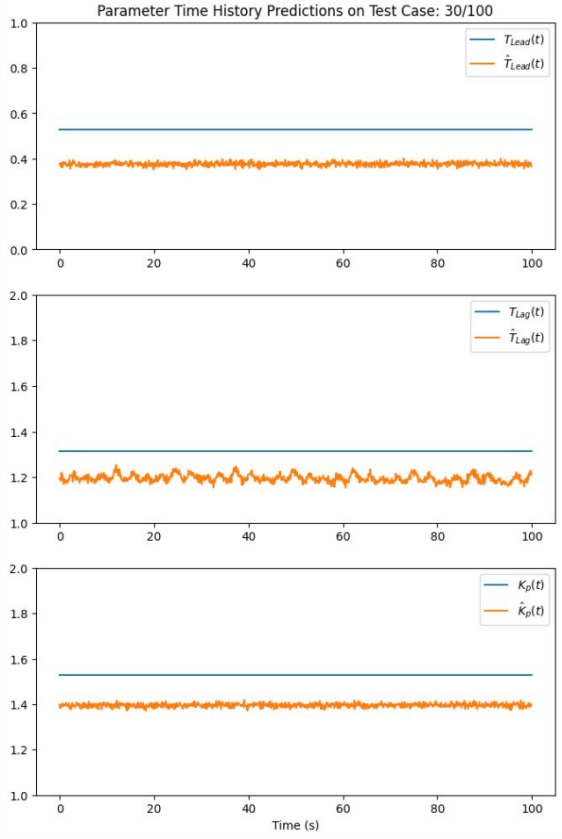
We see decent results however inspecting the time history comparison graphs it can be seen that the network does not really predict a time varying parameter, it predicts a near time constant parameter. maybe this is because of the small change in the parameters or maybe due to how the parameters change causes not much of a total change in pilot behavior... idk.

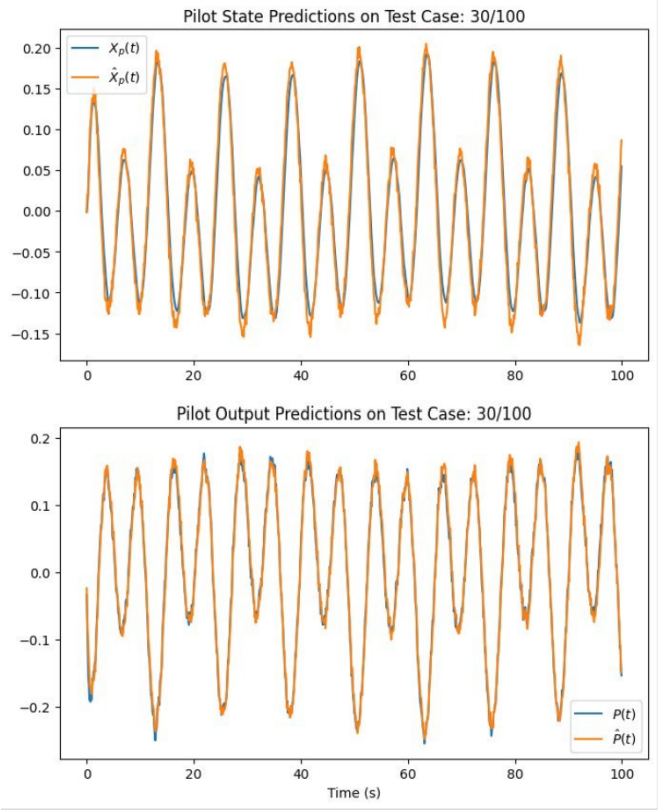
4.4 training on time variant + time variant

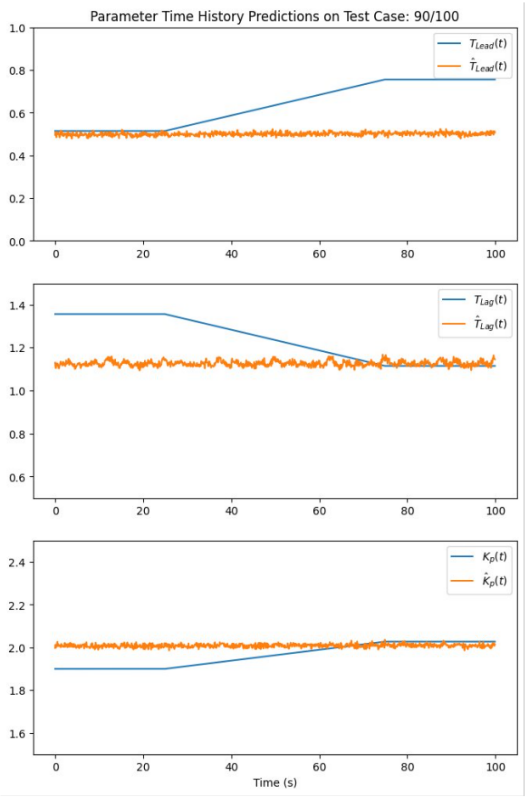
Here we trained a model on time varaint then continued training on time invariant data. then created 100 new samples of evenly created time varying/time invarying test cases. So here are the validation results after training on both types of parameters

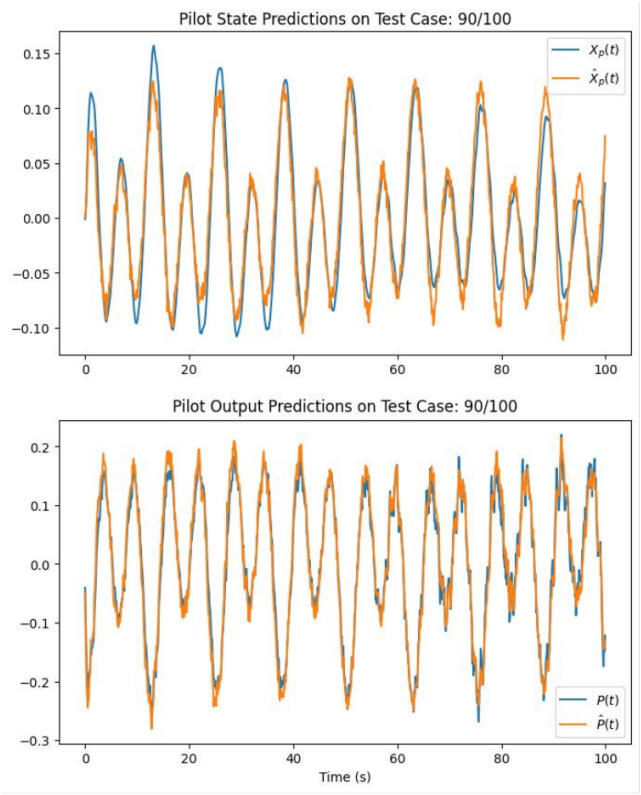


Pilot State MAE on Test Data	Pilot Output MAE on Test Data
0.0287	0.0106







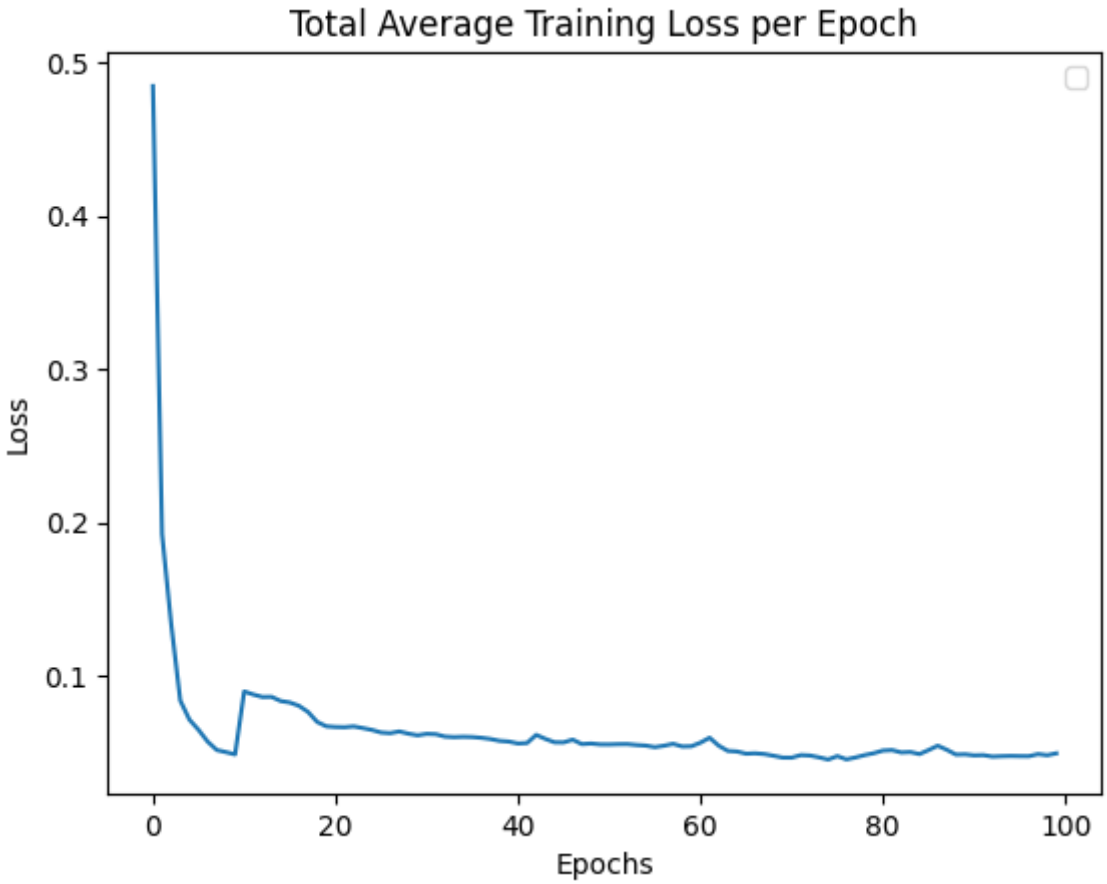


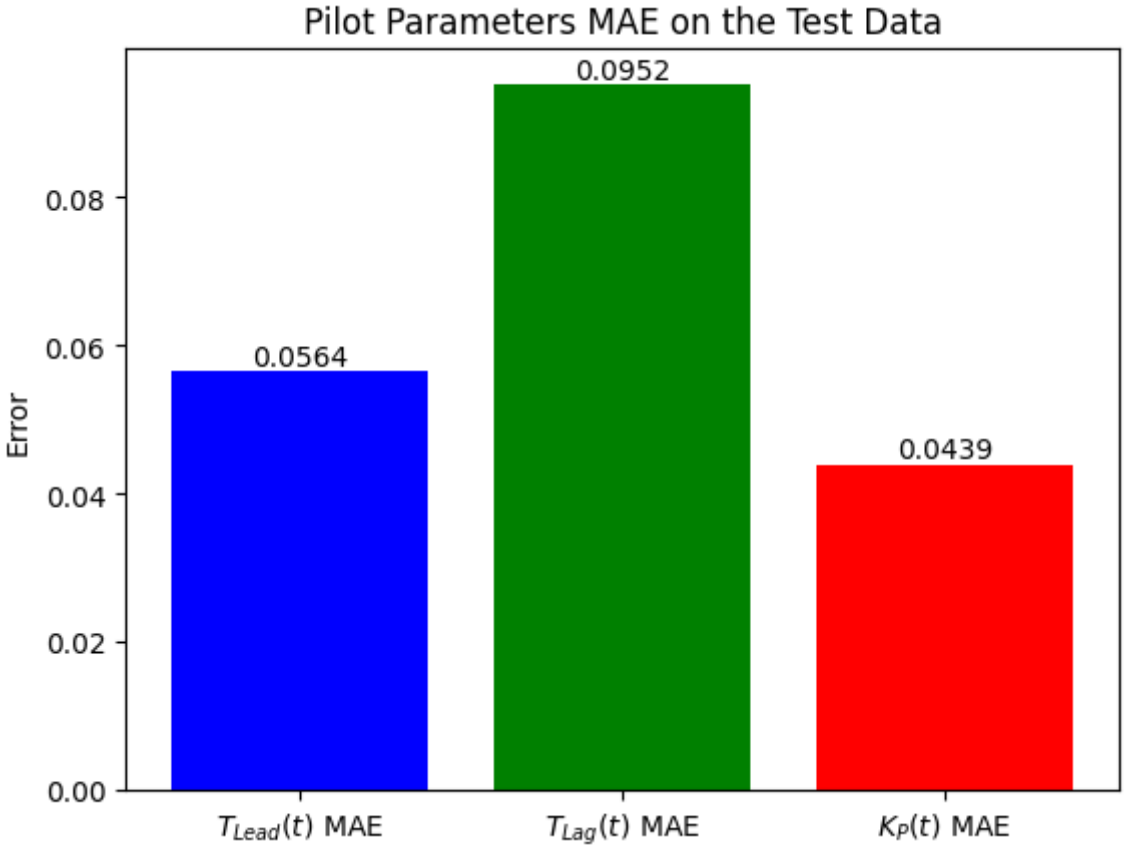
4.5 trying different architectures

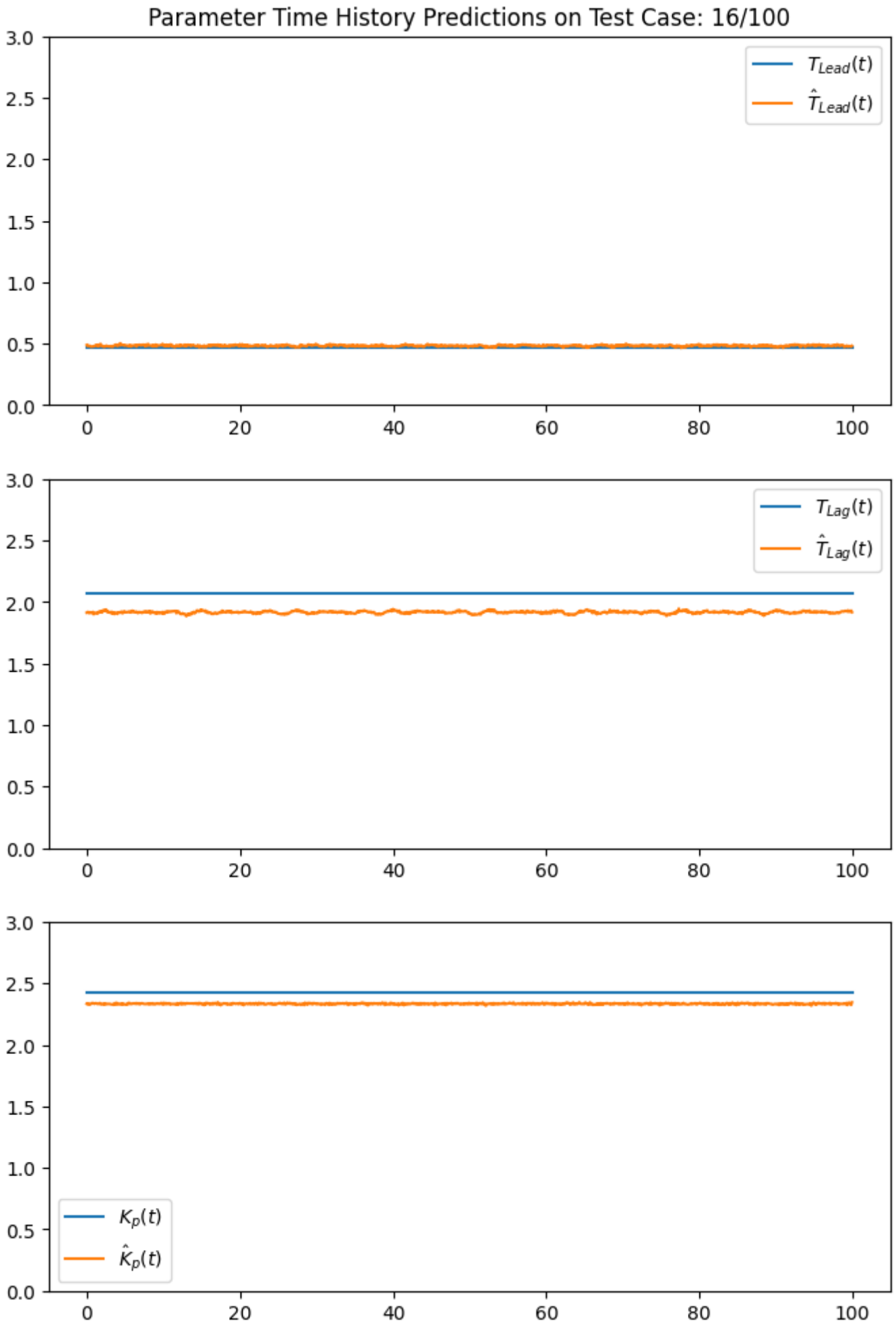
we tried to extend our algorithm to utilize different model layers. we tried using an RNN layer however this became very expensive and broke our system ram only after 13 epochs. we looked to downsample our data to 1hz and this was able to run but only for about 100 epochs or else run the risk of breaking system ram. these results did not prove to be that good.

4.6 increasing training size to time invariant

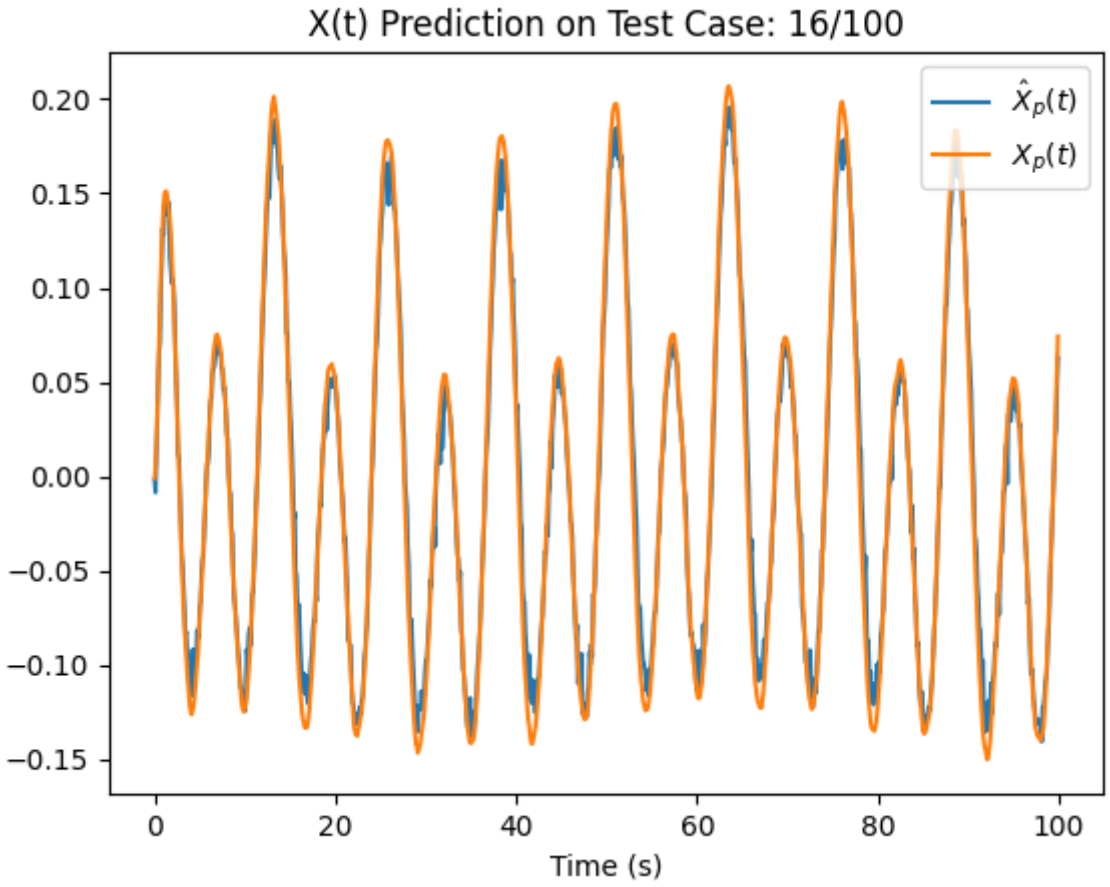
increased training size to 1000 simulation runs split into 90/10 train/test split to see if more data increase results

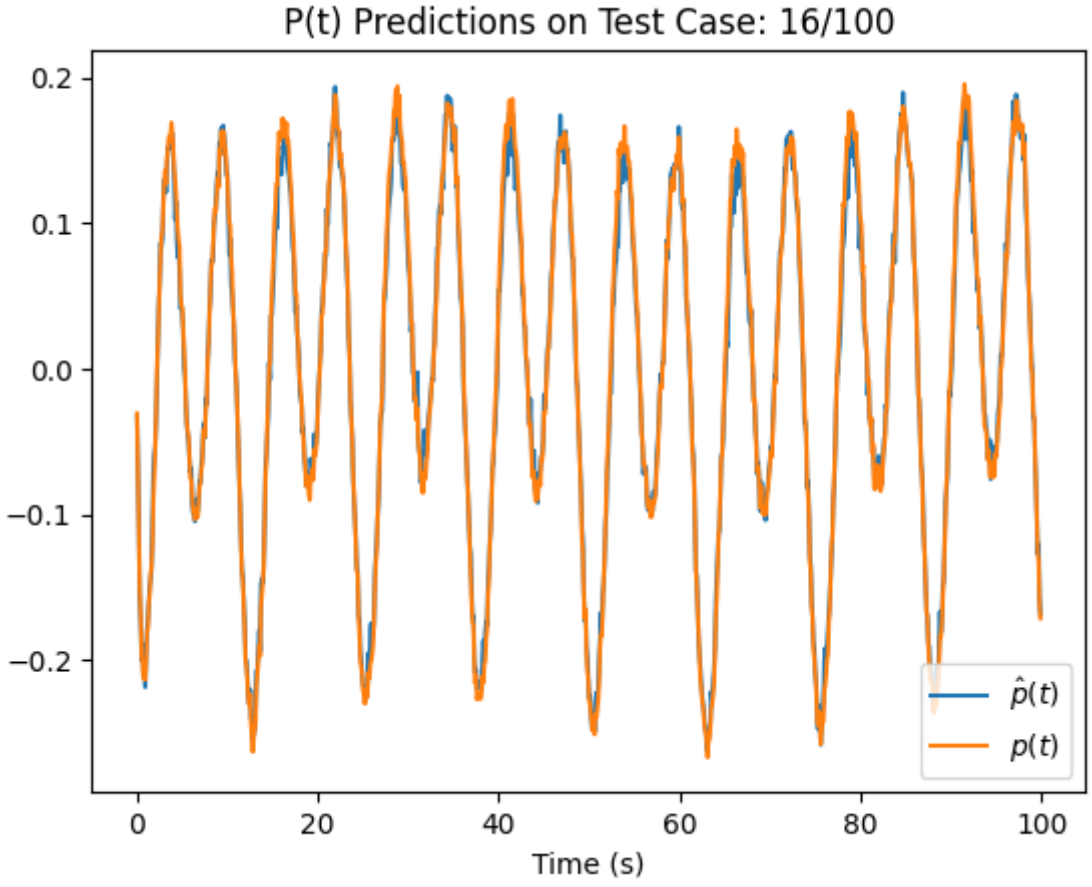

Total Average Training Loss per Epoch
Loss
Epochs
0.5
0.4
0.3
0.2
0.1
0
20
40
60
80
100


Pilot Parameters MAE on the Test Data
Error
0.0564
0.0952
0.0439
0.08
0.06
0.04
0.02
0.00
$T_{Lead}(t)$ MAE
$T_{Lag}(t)$ MAE
$K_P(t)$ MAE


Parameter Time History Predictions on Test Case: 16/100
$T_{Lead}(t)$
$\hat{T}_{Lead}(t)$
$T_{Lag}(t)$
$\hat{T}_{Lag}(t)$
$K_p(t)$
$\hat{K}_p(t)$
0.0
0.5
1.0
1.5
2.0
2.5
3.0
0
20
40
60
80
100

Time (s)


X(t) Prediction on Test Case: 16/100
$\hat{X}_p(t)$
$X_p(t)$
0.20
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
0
20
40
60
80
100
Time (s)


P(t) Predictions on Test Case: 16/100
$\hat{p}(t)$
$p(t)$
0.2
0.1
0.0
−0.1
−0.2
0
20
40
60
80
100
Time (s)

MAE(x(t)) = 0.01605515
MAE(p(t)) = 0.0075

only 10 minutes to train

4.7 extend the time varying to time invarying

The assumptions are that the parameters are low frequency and within a certain range of time (easily 100 seconds) that the parameters will be constant. so to simplify we will take our time constant model which looks at 100 seconds and evaluate it on a longer time history for example (1000 seconds) and break that 1000 seconds test run into 100 seconds chunks to evaluate our machine learning model