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 1:

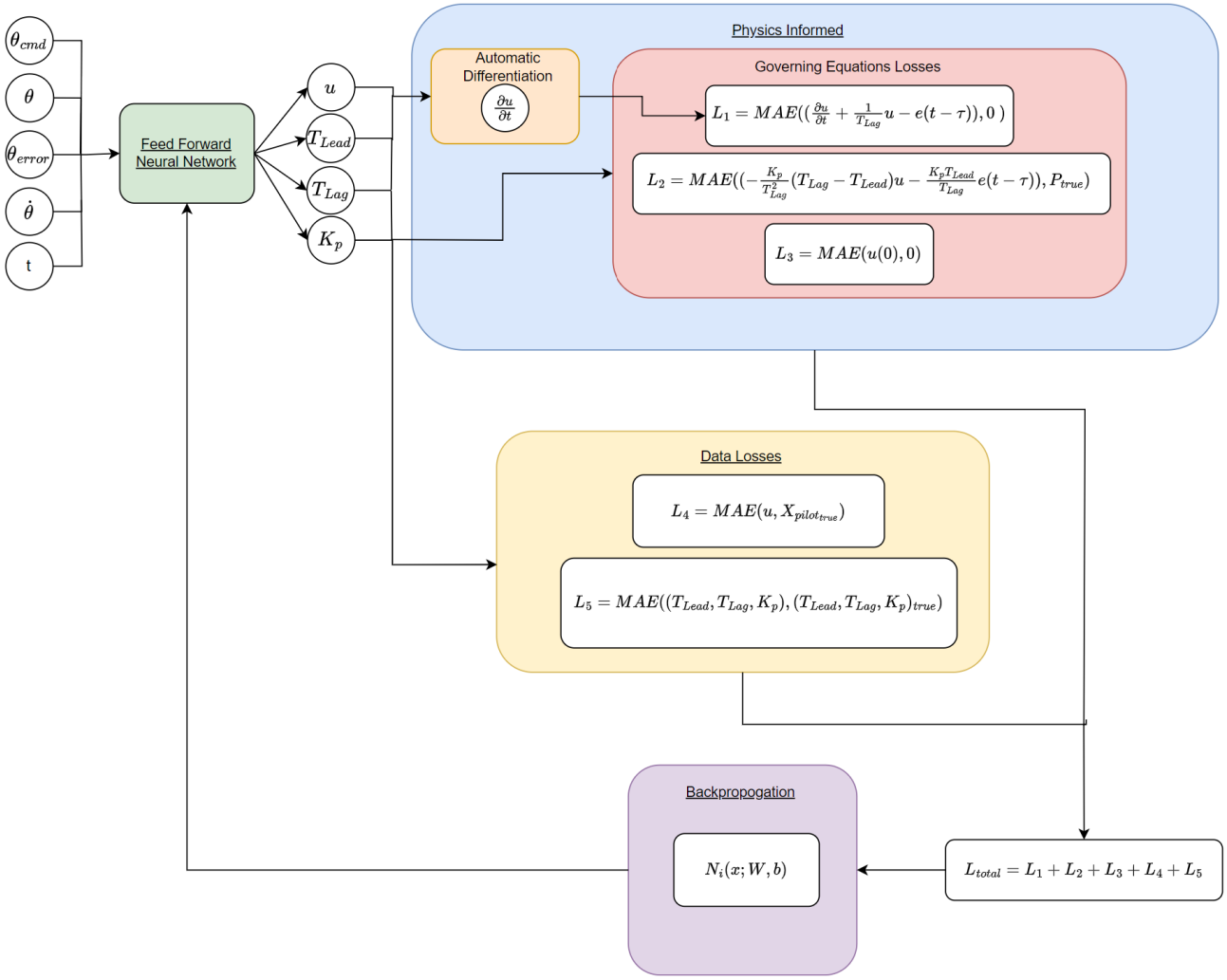
Intro

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

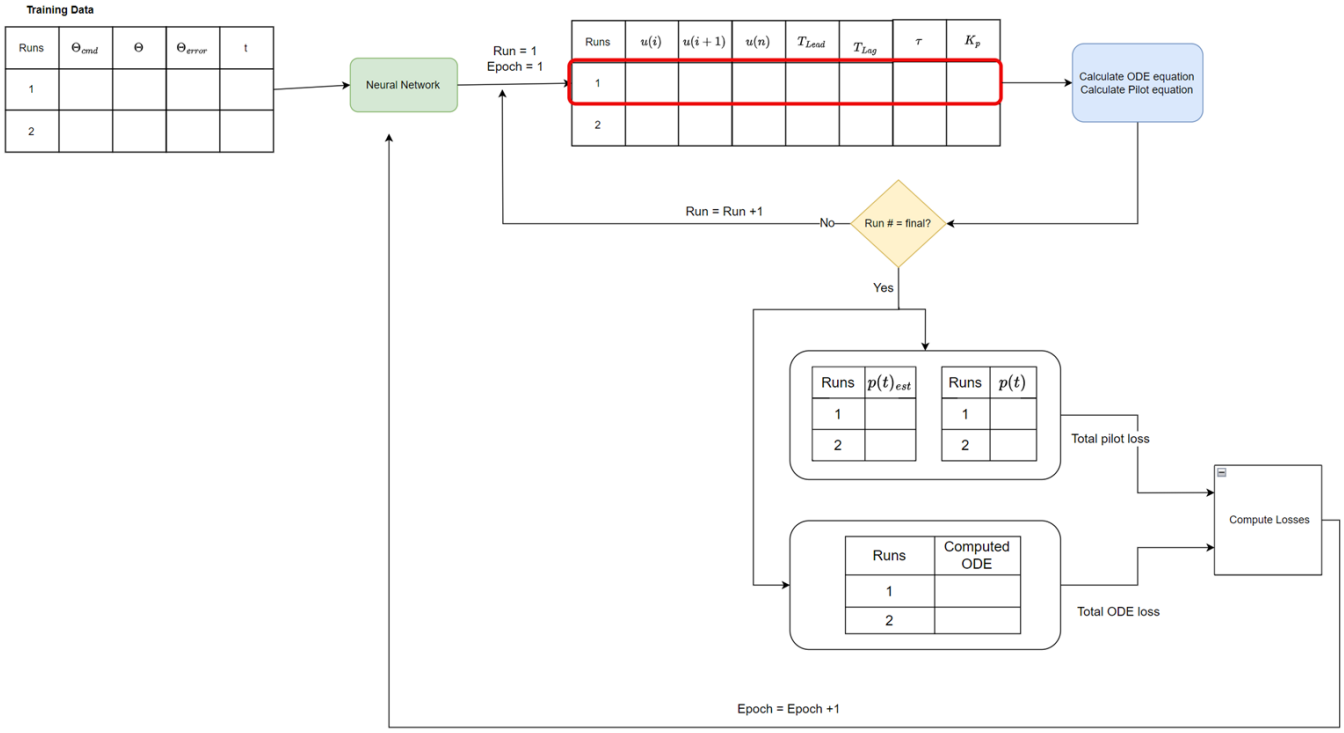
and properly estimate the x(t) and parameters for any run of data. For this we make some modifications to be able to extend PINNS to multiple simulation run datasets.

Architecture:

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



Details on L_1, L_2 calculations see below:



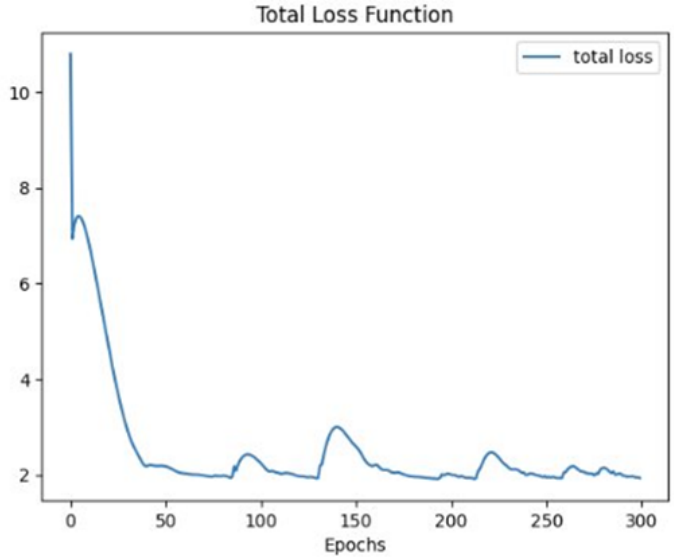
Analysis:

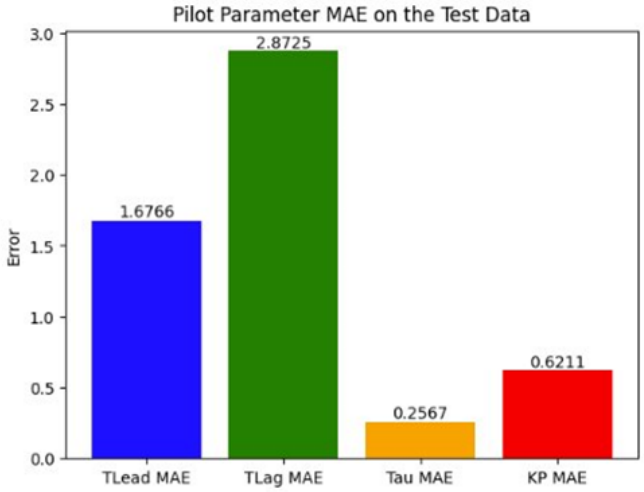
It was found that the ODE loss term is very computationally expensive to run, thus any attempts to do so with any substantial dataset size would result in a crash in the Google Colab. Because of this we experimented only with small datasets. The most we could really train for any not crash was about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

Results:

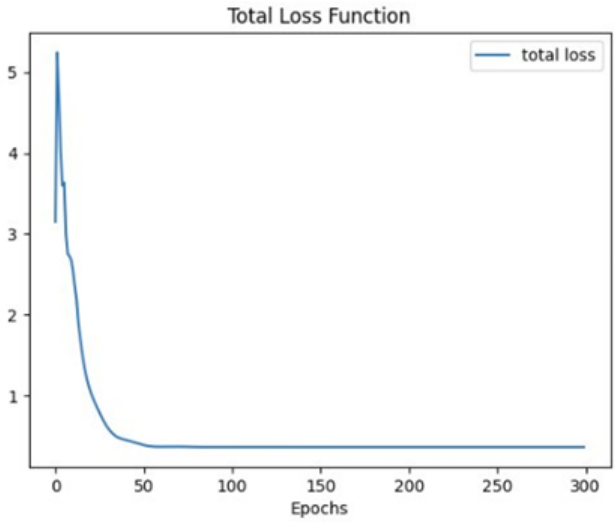
These results were so so, it is inconclusive if this approach would have worked just because our dataset was so small. The following results are from a dataset of 25 simulation runs, split into train/test set of 20/5, the loss function = $L_{params} + L_{xp} + L_{p(t)} + L_{ODE}$ so from the above architecture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

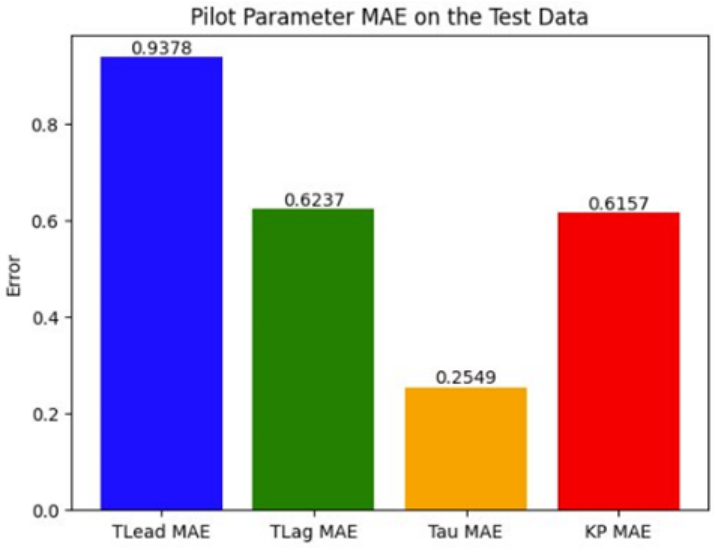
Trial 1:



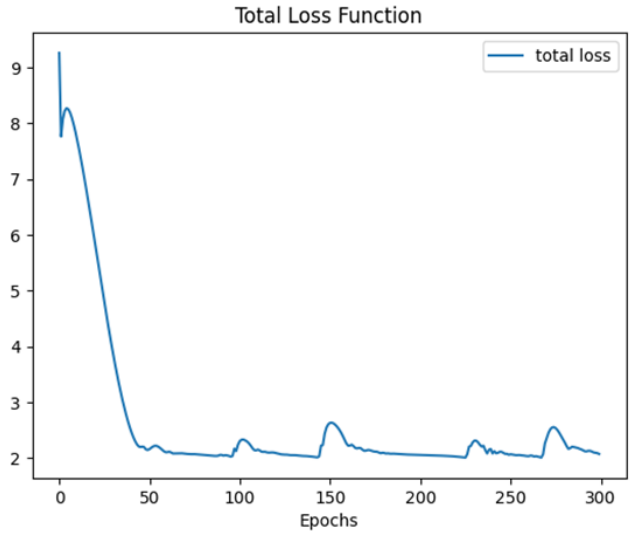


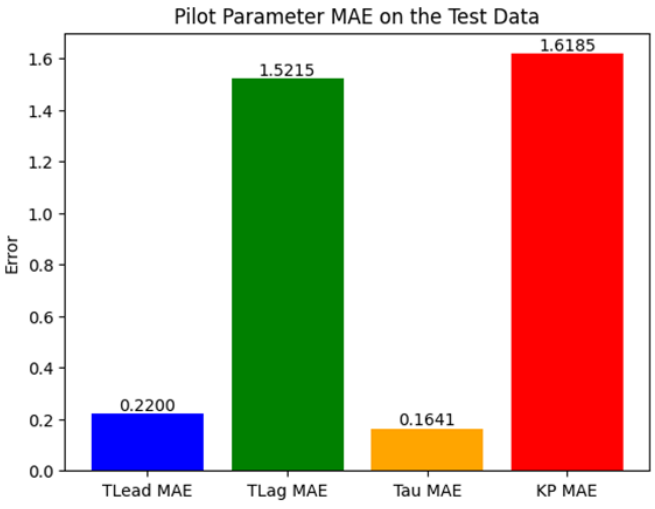
Trial 2:





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. 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

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

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

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

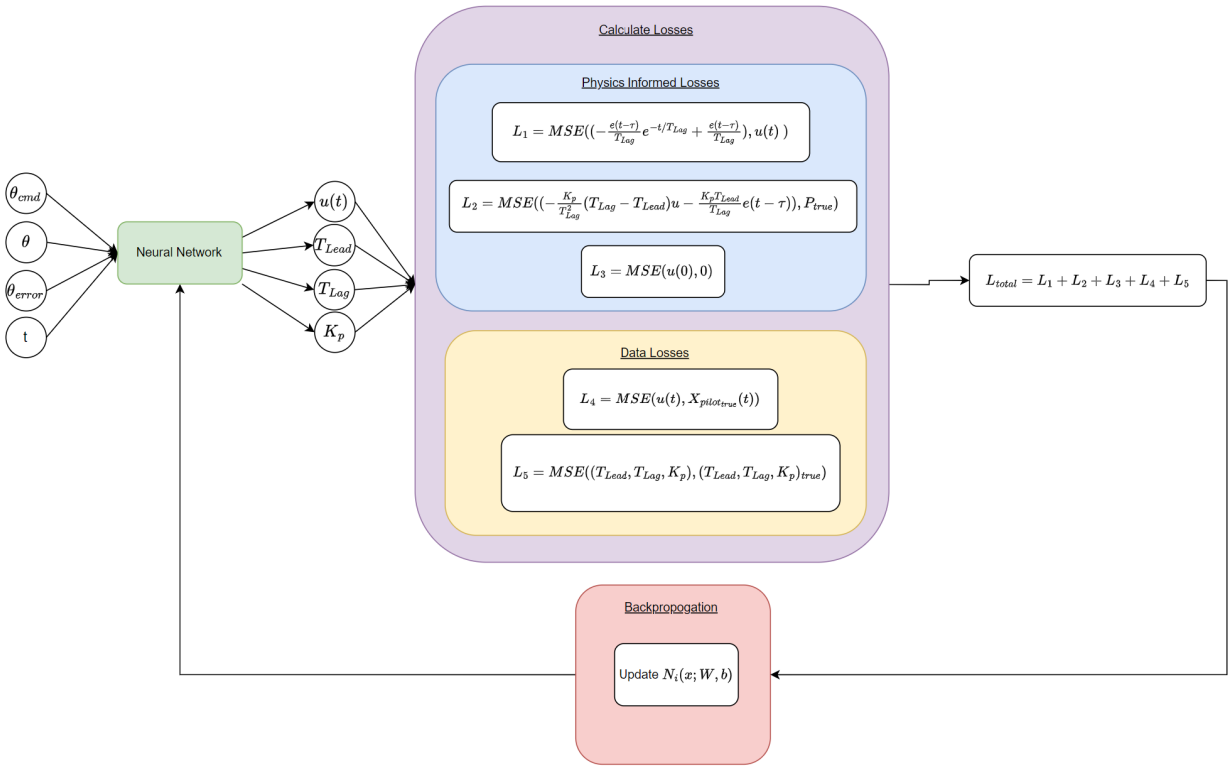
$C = \frac{-e(t-\tau)}{T_{Lag}}$

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

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,

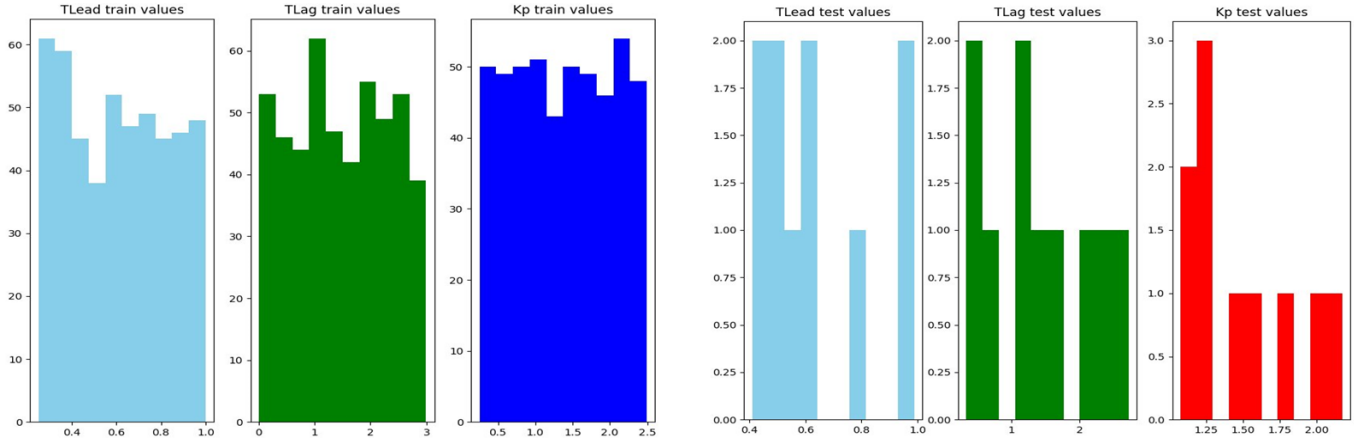


just to note because we solve for the analytical solution of x with initial condition equal to zero we can get rid of/omit L3 which is the initial condition loss term. With this updated architecture we will forgo the automatic differentiation because it is computationally expensive on large datasets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

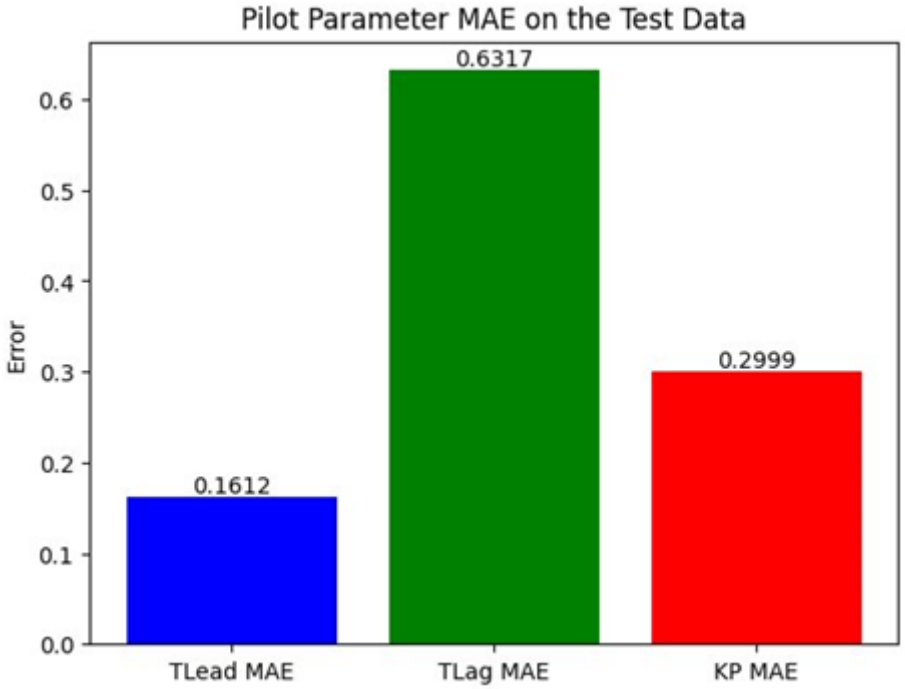
Training Setup:

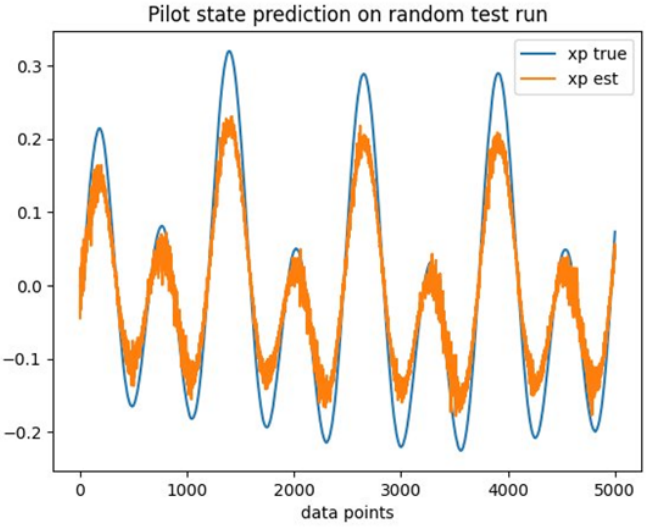
1- Load in 500 simulation data runs (490/10 train/test)
2- Pitch commanded: sum of sines
3- Dense layers model, 100 epochs
4- True pilot generated parameters in simulation: 0.1 < TLead <1 , 0.1 < Tlag < 3 , 0.1 < Kp < 2.5

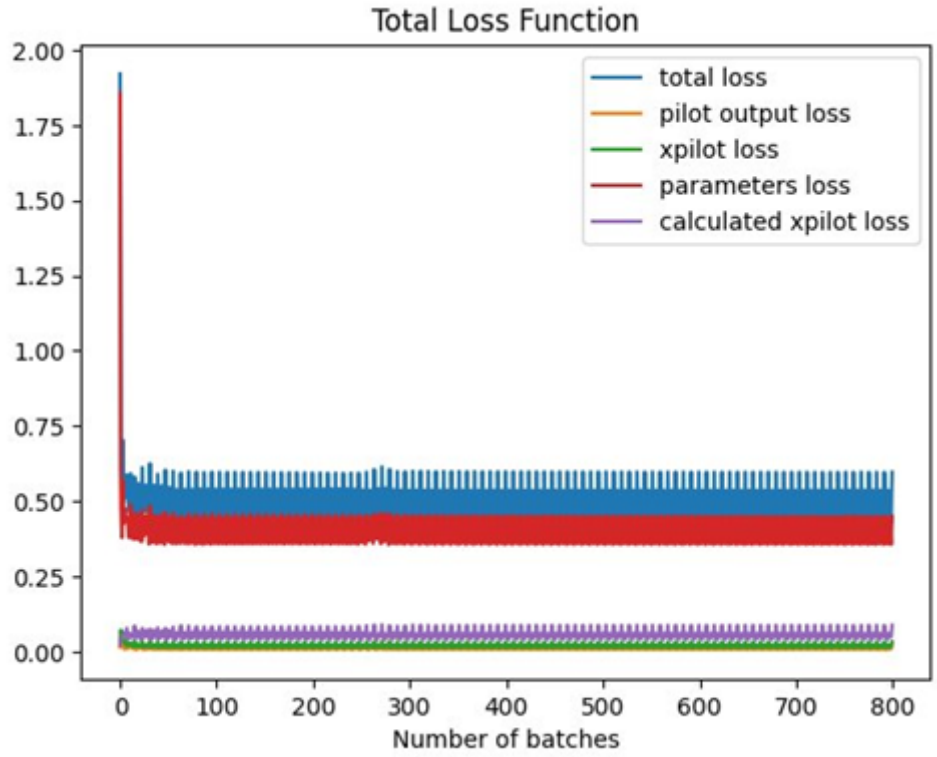
5- Loss = L1+L2+L4+L5



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