Time-Series Deep Learning Models of a Electrolyzer

Jake Immonen, Jiwei Yao, and Scott Springer

Introduction and Problem Motivation

Currently, there is a large interest in producing green hydrogen, which is hydrogen that is produced without greenhouse gas emissions. Hydrogen is a common feedstock chemical with its main use in fertilizer production. The most common method to produce hydrogen is via steam methane reforming which has a large carbon footprint. The CO_2 footprint from global hydrogen production amounts to a 2.5% of all global CO_2 emissions, clearly indicating that the current ways of producing hydrogen need to be decarbonized [1]. In addition, hydrogen is a potential new fuel of the future that can be used for carbon-free transportation, process and space heating, and a useful energy storage medium, making its sustainable generation of great importance both for its current uses and for its future use.

One of the most promising methods to generate green hydrogen is via electrolysis where water molecules are split into hydrogen and oxygen in an electrolytic cell by passing a current through the cell. An efficient sub-type of electrolysis is high temperature steam electrolysis (HTSE) which utilizes steam, typically around 800℃ for its electrolysis reaction. Electrical input for the HTSE reaction can be reduced 37%, compared to lower temperature electrolysis methods.

As part of Jake Immonen's Ph.D. research, he has created a dynamic physics-based model of a high temperature steam electrolysis reactor. The reactor has two inlet streams, a cathode stream and an anode stream. The cathode stream contains steam and a small amount of recycled hydrogen gas, while the anode stream is air. The dynamic model is spatially varying in the length direction of the reactor and embodies first principles, physics-based electrochemical and mass balance equations. The reaction depends on various manipulated parameters including temperature of the two inlet streams and concentration of each species in the inlet streams. The conversion of steam to hydrogen depends on the applied amperage through the reactor. The applied amperage, in turn, changes the voltage of the reactor. At higher amperages, the voltage increases and the reactor goes into an exothermic mode where the temperature of the streams exiting the reactor increases. With lower voltages, the reactor goes into an endothermic mode where temperatures of the streams exiting the reactor decrease. There is also a thermoneutral voltage where the temperature remains constant in the reactor. This means that there is a relationship between desired conversion of steam, voltage, and outlet temperature of the reactor with the inlet flow rates, species concentrations, and applied amperage. Figure 1 shows a simple diagram of the electrolyzer along with the independent, input variables in blue and the dependent, output variables in red. In the process, the blue variables can be manipulated to change the outlet variables.

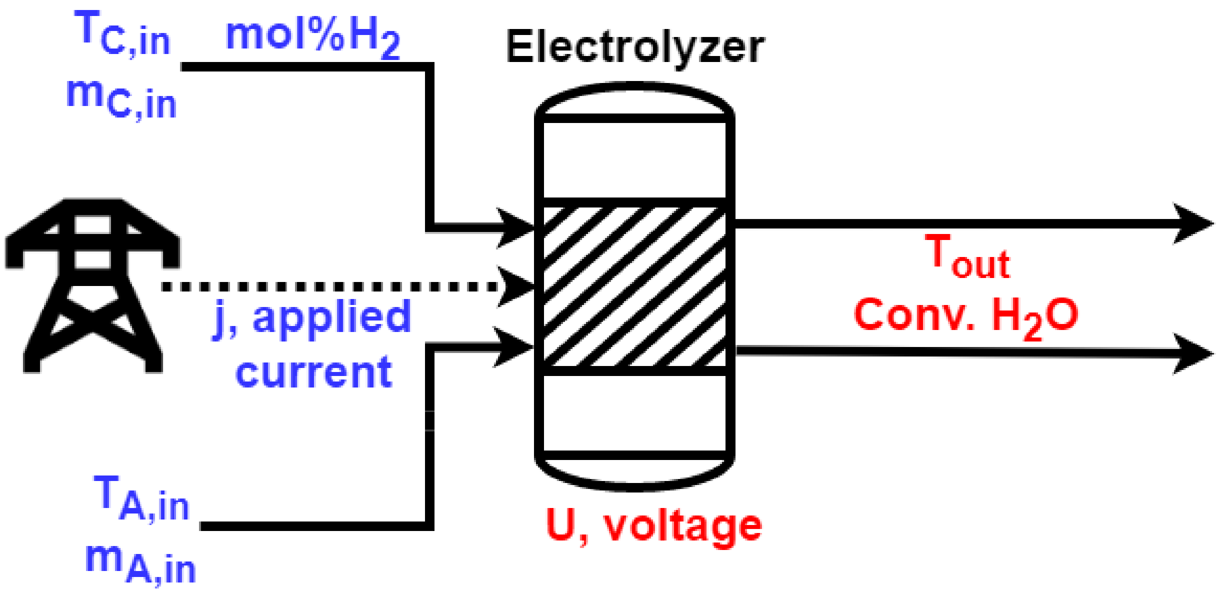


Figure 1: Simple electrolyzer diagram with blue values showing model input data and red values showing what values want to predict into the future. The red values are also used as input data as they become measured.

The goal of this project is to create two Deep Learning models, an LSTM and an informer, that predict the behavior of the reactor into the future given input conditions and past time behavior of the output variables. These time series models could then be used in future research for model predictive control[1] which uses the time series model to optimize control over the reactor through time. Additionally, the models developed could be used as surrogate models to replace the physics-based models to drastically speed up computation time making other problems involving the simulation more tractable.

Deep Learning Data

The data for the model to be used is simulated data coming from the physics-based model of the electrolyzer. First, low discrepancy sampling[2] is performed using a 'MiniMax' routine to find conditions to run the simulation. This generates data for disperse conditions that cover a range of values in the input space. The input variables and the ranges that they could be sampled from are described below.

[1] https://en.wikipedia.org/wiki/Model_predictive_control
[2] https://en.wikipedia.org/wiki/Low-discrepancy_sequence

Table 1: Input variables for simulation to create data.

Variable	Description	Range	Units
$T_{C,in}$	Temperature of cathode stream into reactor	1023-1123	K
$T_{A,in}$	Temperature of anode stream into reactor	1023-1123	K
$m_{C,in}$	Mass flow rate of cathode stream into reactor	6-9	kg/s
$m_{A,in}$	Mass flow rate of anode stream into reactor	6-9	kg/s
j	Current applied to electrolyzer	4,000-8,000	mA/cm^2
mol% H_2	Mole percent of hydrogen in cathode stream	6.7-13.3	%

Simulations are then run where the model reaches steady state for an initial condition, all 6 of the input variables are then changed to sampled conditions via first-order transfer functions (step changes tend to be to drastic and can cause simulation errors). Then the model is allowed to come to steady state again. After that, the input values are changed back to the initial condition by a first-order transfer function and again come to steady state. Simulation values are made to be every minute after interpolation (the model uses a variable time-step solver so some time steps are more than a minute and some are less). Data is collected for 1,620 of these types of runs which amounts to 7,335 hours of simulated data. Next, the data is strung together into one large continuous dataset which is possible because at the end of every run the model goes back to the same conditions. This was done because the model has numerical issues for some of the conditions data was generated at, so in some cases the model will fail to run. Having every run go back to the same conditions allowed for a continuous data set to be created easily for the purposes of this project, but can skew what the model is learning. Particularly, the model will always learn the dynamics to and from this initial condition. In the future, data should be created in a different manner, but due to time constraints it was generated this way to move to the model development sooner.

Lastly, data is also collected or 'measured' in the same manner for the output variables, or the y variables, we are interested in. These will be the prediction variables which are described below. It should be noted that the temperature out of the reactor equilibrates for both streams, so only one value is necessary there.

Table 2: Output variables, y, that eventually want to predict.

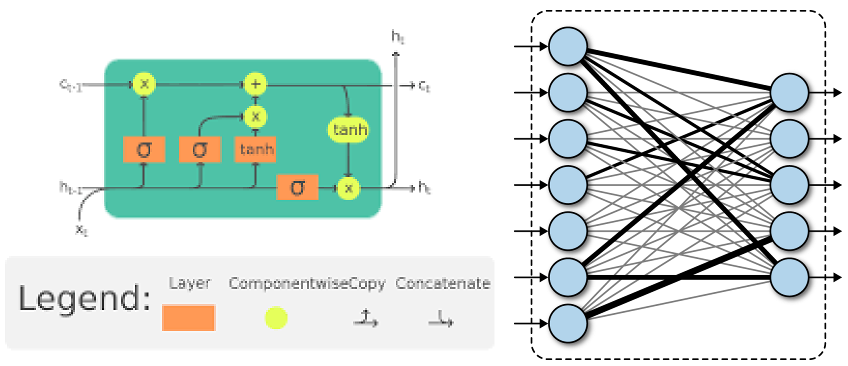
Variable	Description	Units
T_{out}	Temperature of both streams out of the reactor	K
Conversion of H_2O	Percentage of H_2O that is converted to H_2	%
U	Voltage of the electrolyzer	V

Both models use a window of looking back at data to make the next prediction. This corresponds to the previous datapoints for the input variables and the previous 'measured' output variables to make the prediction for the next y variables.

LSTM Model Development

General RNNs allow for the ability to have variable length inputs and to have dependence on previous time steps in the current prediction. However this runs into a vanishing signal problem when there needs to be dependence on the distant past. LSTMs try to solve this problem by being able to hold a long term 'attention' state. This allows for the model to have predictions that are based upon long time scale behaviors.

The LSTM we used was constructed using the LSTM module in PyTorch and then two linear layers with batch normalization. The LSTM had input size of 9, 6 known input variables and 3 variables that are the previous predictions/measure values. In training mode the 3 prediction input variables were known from the dataset. In prediction mode the 3 prediction variables were the previously predicted values.



$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

Figure 2: LSTM model overview.

The LSTM was 2 stacked classic LSTMs with hidden matrices of size 9x30. The stacked method was used to give the LSTM more depth. The output of the final recurrence step of the LSTM was inputted into the first linear layer of size 30x128. This was then fed into the second linear layer of size 128x3. This gave us an output size of 3.

We used a mean squared error loss function as we were predicting continuous values.

To optimize the model we used ADAM with a learning rate of $1\text{x}10^{-4}$, batch size of 128, and 3,500 iterations.

Informer Model Development

In this study, we adopt the informer to perform the time-series forecasting task [2]. The structure of the informer is demonstrated in Fig. 3. Unlike the LSTM which only processes the data step by step, the informer is able to take a long time series of input and predict the long-time series prediction at once.

The informer needs two inputs, one for the encoder and the other one for the decoder. In this project, the encoder takes the past 50 timestep information as an input to generate the concatenated feature map. The decoder takes past 30 timestep and future 15 timestep information as the input. For the future 15 timestep, the predicted features are masked. In this study, two encode layers are stacked within the encoder layer, and two decode layers are stacked in the decoder. The model has a dimension of 512.

To optimize the model we used ADAM with an initial learning rate of 0.0001, with a batch size of 64. MSE loss is used to train the model. For the training, 32,400 iterations were performed.

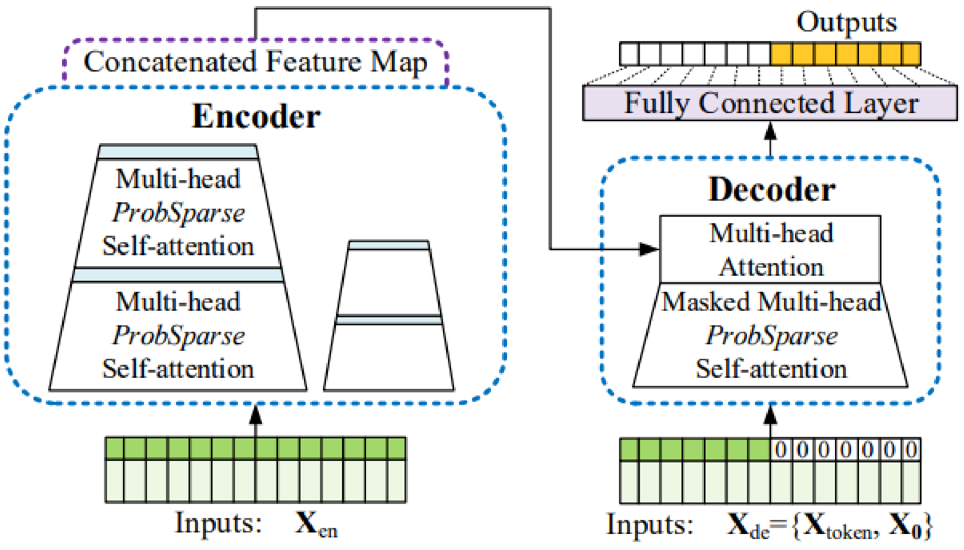


Figure 3: Informer model overview [2].

Results

Each model is trained with the first 80% of the data collected and tested on the remaining 20%. The way the data was generated was already randomized so splitting the data in this way shouldn't cause any issues. The mean squared error of the next predicted value y values make up the loss functions. The MSE of training and testing are shown below.

Table 3: MSE of LSTM and informer predicting 1 timestep into the future on training and testing data.

Variable	**Training MSE**	**Testing MSE**
LSTM	0.0007	0.02087
Informer	0.0073	0.00644

The MSE of the LSTM is very low for the training data showing that the model performs well on training data. On the testing data, though it is two orders of magnitude higher indicating that the model performs much poorer on data it hasn't seen before. For the informer the MSE is slightly higher than the LSTM on training data but when it encountered data that it had not seen before (testing) it performed much better than the LSTM. This suggests that the LSTM is overfitting on the training data whereas the informer is 'learning' more general properties of the system and can be seen as a better model.

The LSTM was easy to train and had no problems converging to the low MSE value seen above. The informer took longer to train when compared to the LSTM. This is to be expected as the informer has more learnable parameters. This is likely one reason why the informer was able to 'learn' the general structure of the system.

The informer is shown to have better capacity to predict future conditions of the plant when compared to the LSTM. This is consistent with what other groups have seen and the general consensus of the machine learning community and is why the LSTM has fallen out of favor.

To better visualize how the models performed some data is selected for visualization purposes. The figures below show the 6 input variables during training and the corresponding prediction variables for each model to better gauge the training performance.

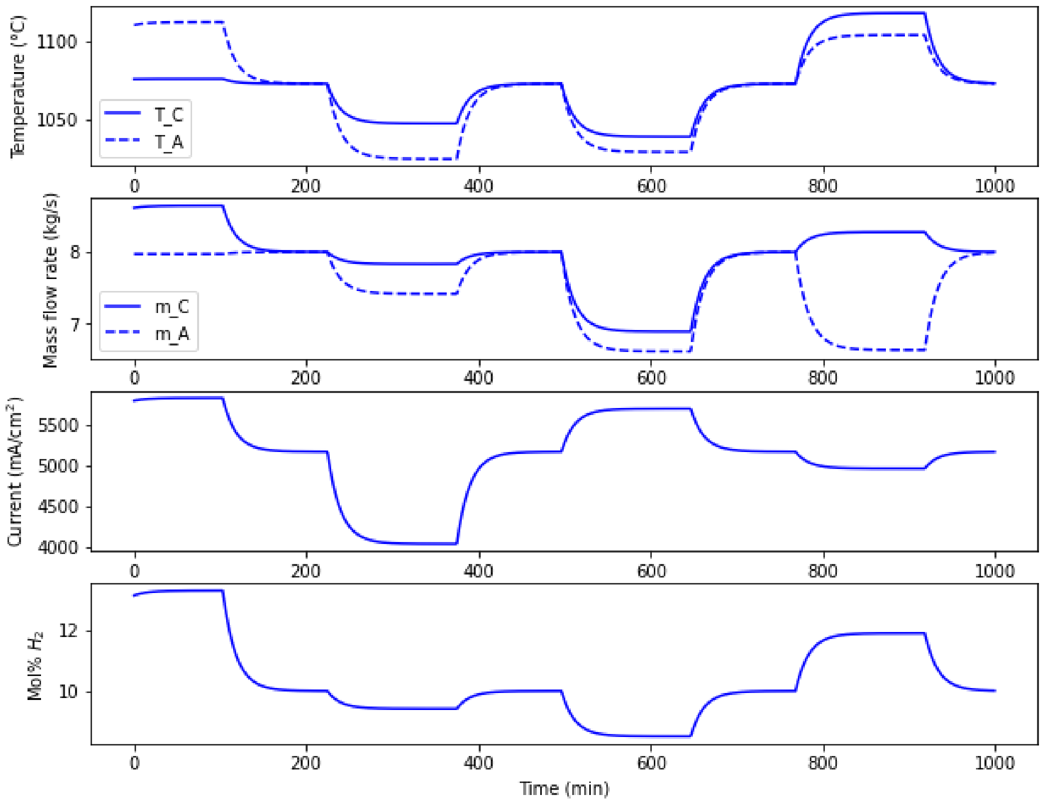


Figure 4: Sample of input variables for training data.

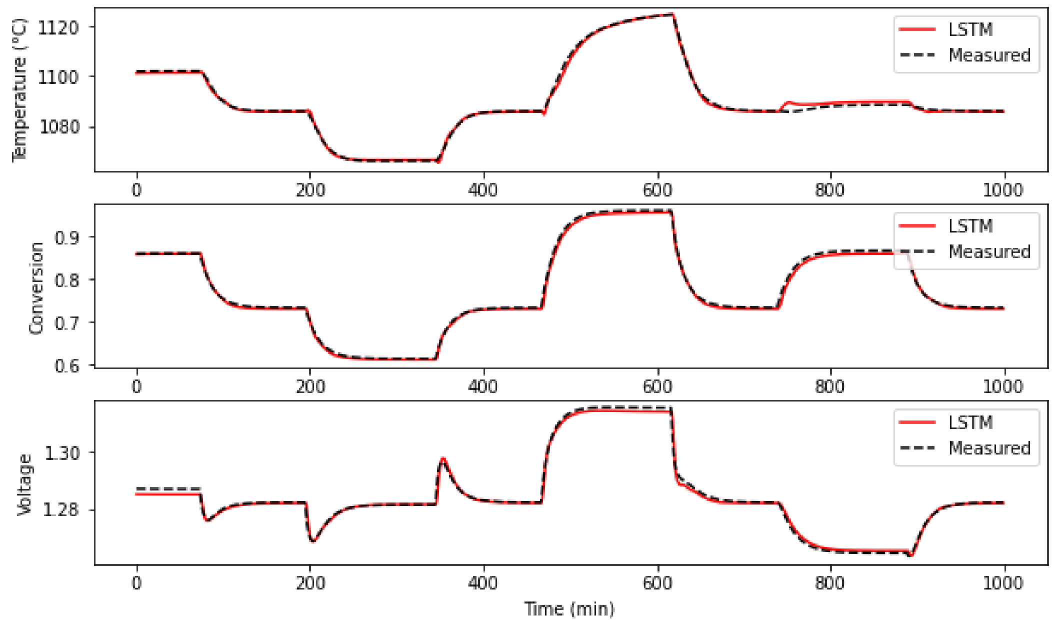


Figure 5: Corresponding prediction variables, y, for training data from LSTM and actual values measured from the simulation.

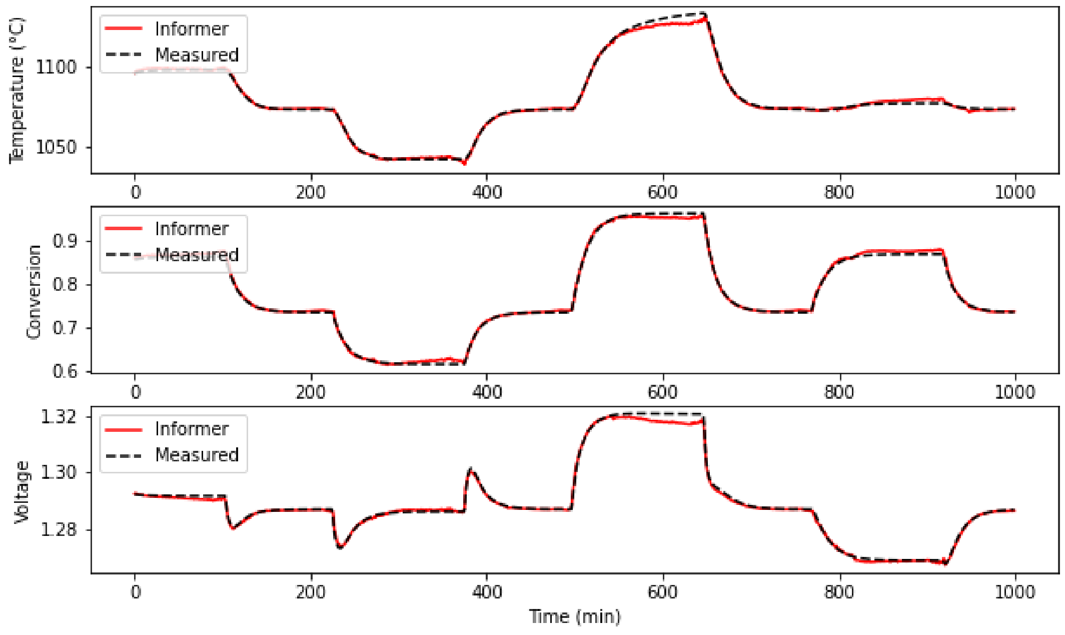


Figure 6: Corresponding prediction variables, y, for training data from informer and actual values measured from the simulation.

As can be seen from the above figures, both the LSTM and informer model are able to very accurately predict the predicted variables on the training set showing that the models have learned the training data well. However, the informer performs relatively poorly on regions where the predicted values are close to their maximum value.

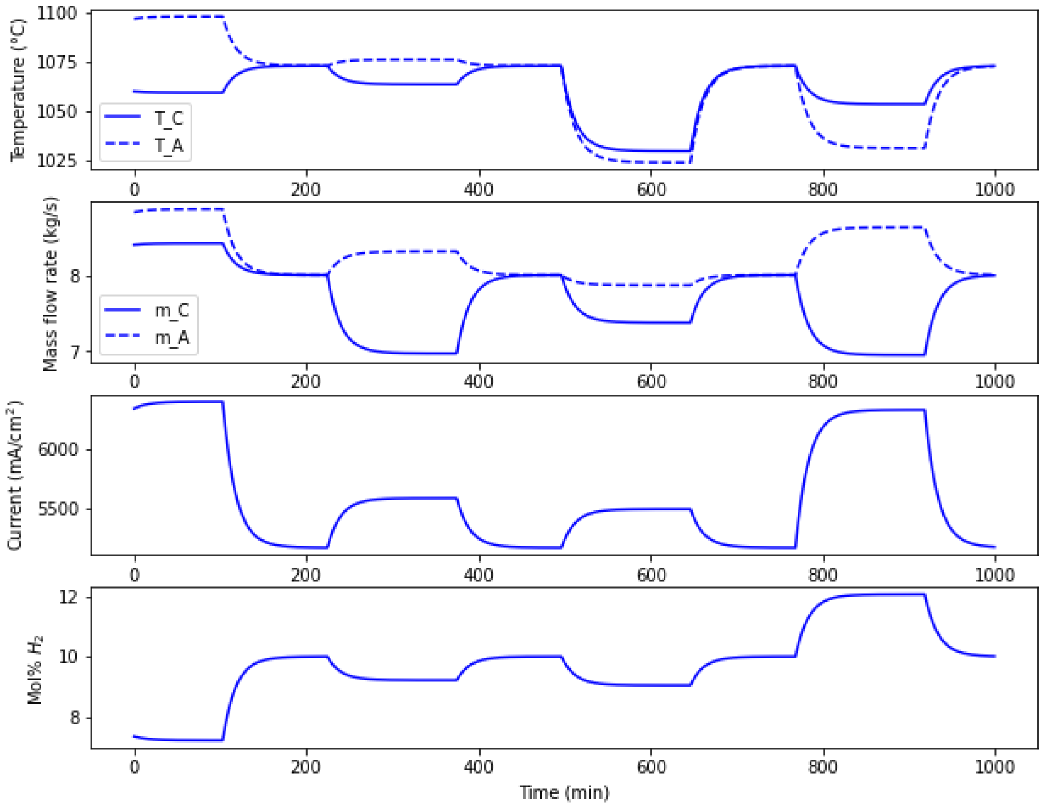


Figure 7: Sample of input variables for testing data.

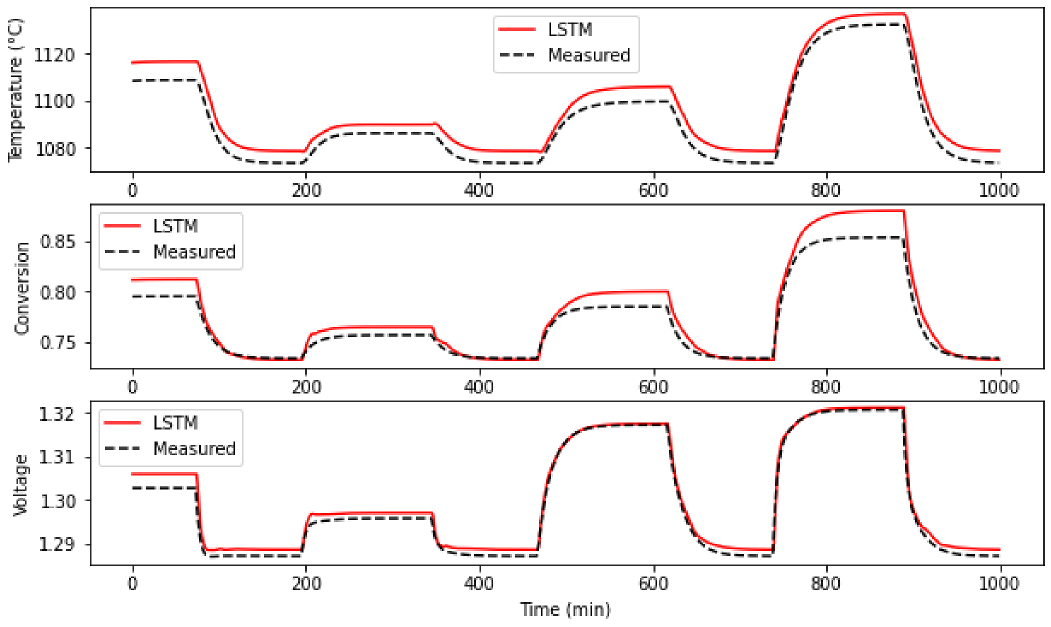


Figure 8: Corresponding prediction variables, y, for testing data from LSTM and actual values measured from the simulation.

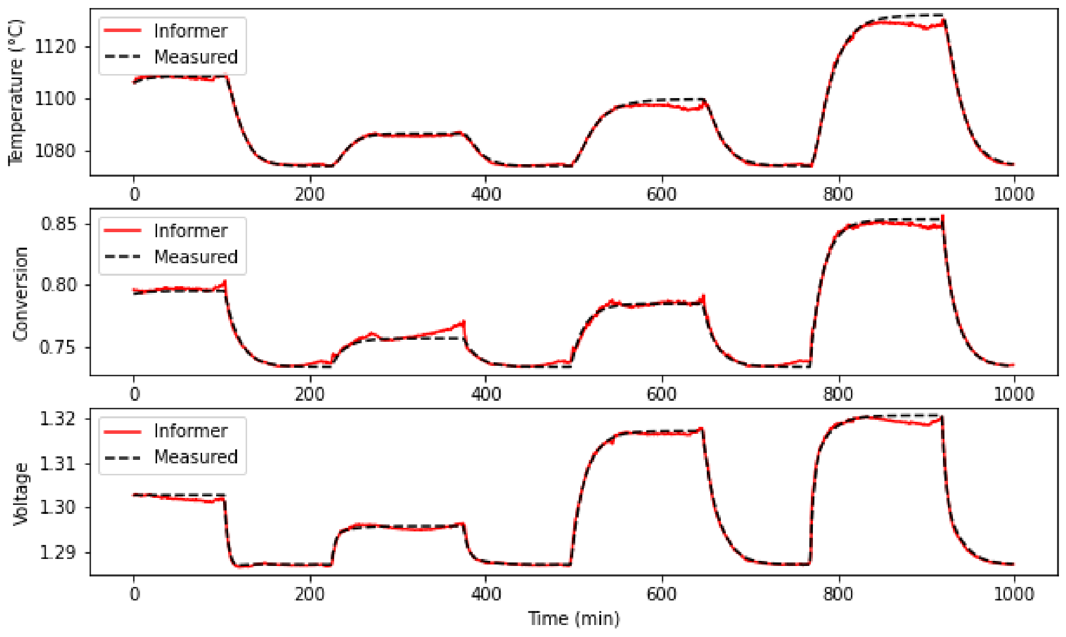


Figure 9: Corresponding prediction variables, y, for testing data from informer and actual values measured from the simulation.

From the figures above, the LSTM predicts less accurately on data it has never seen before. It follows the dynamics of the process nicely, but is in general off by a small buffer. Interestingly, the voltage prediction appears to be more accurate than the other two predictors. In contrast, overall the informer provides an accurate result especially for the unsteady-state region. The informer predicts some values off during steady state times, but on the whole provides better results than the LSTM.

Lastly, both models are used in a prediction mode where the y values from the LSTM and informer are then used as values in the past so that the model can step forward multiple time steps into the future to make longer term predictions. This means that none of the measured y data is ever used except to initialize the first 30 timesteps. The figure below shows the predictive capability of the LSTM model.

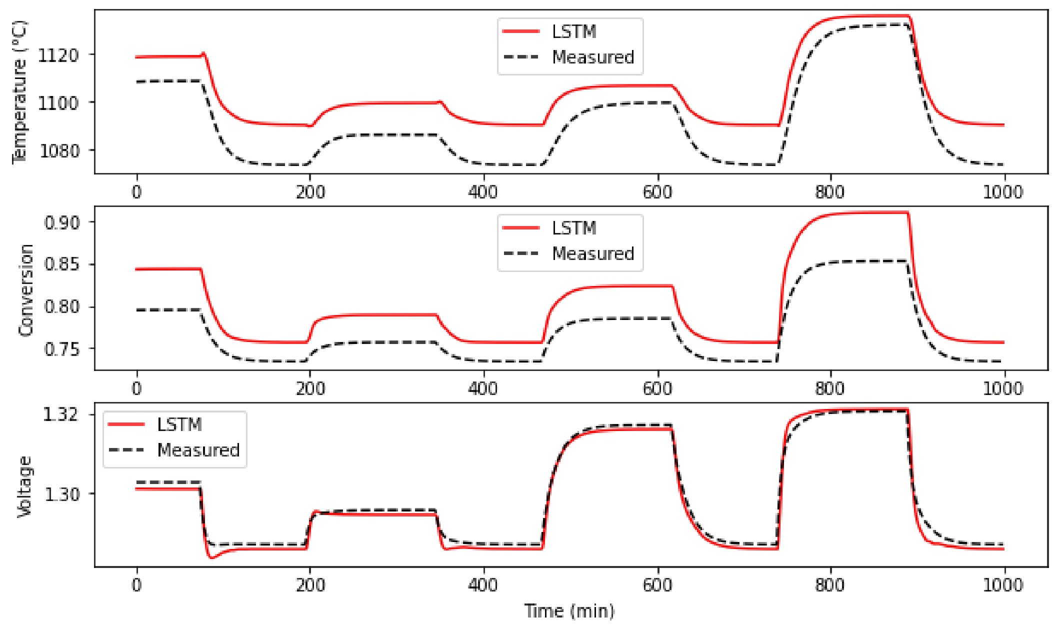


Figure 10: LSTM in prediction mode, where the measured y values are never used as inputs into the model. Rather the predicted y values are used as measured values.

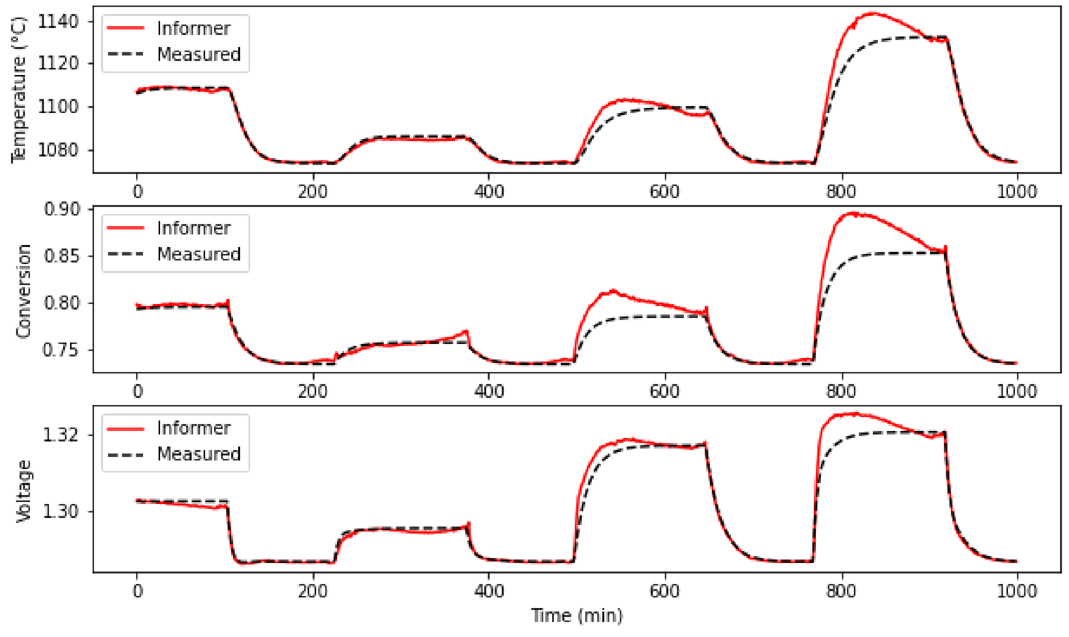


Figure 11: Informer in prediction mode, where the measured y values are never used as inputs into the model. Rather the predicted y values are used as measured values.

Interestingly, looking at Figure 10, the LSTM in prediction mode behaves similarly to how it does on the testing set, and as time goes on, is never able to get rid of the buffer. This is likely due to not having the measured values to use as a sort of feedback. Contrast this with Figure 8, which is the LSTM on test data where it doesn't have nearly as high of a buffer especially as time goes on. This makes sense, intuitively, as the prediction mode is never able to self-correct with measured data.

The informer on the other hand, performs quite well in prediction mode especially within the first 300 minutes of prediction. This is likely due to the fact that the informer is less overfit and behaves better on testing data as well as the nature of how the informer is set up where it used the past 50 minutes to predict the future 15 minutes. However, as the prediction error accumulates, the informer fails to predict the steady-state after 500 minutes. But surprisingly, the informer still performers relatively well for the transition state (ramping down).

Conclusions and Future Work

In this work we can see that the informer is better at learning the properties of the system and is able to predict future behavior more reliably when compared to the LSTM. However the informer is more expensive to train. In the future we could try different architectures for both models to improve their performance. One specific change to the LSTM is to add a regularization technique to penalize overfitting to training data. As for the informer, due to the limited time, hyperparameters are not fine tuned. Therefore, fine tuning hyperparameters may generate a better result.

On the data generation side, it would be beneficial to train the models with more diverse data. Specifically, changing the time scale of how each input variable was changed for data generation, adding quicker and slower dynamics to the transfer functions used. In addition, all the data was created where the input variables are changed from an initial condition to a new condition and then back to that initial condition. As mentioned in the Data section, this was done for time sake, but more data should be created for training that doesn't go back to this initial condition so that the models can generalize the physics of the system better and predict any conditions based on the input variables.

Lastly, with the models that have been developed, advanced control of the reactor could be developed where the prediction mode of the models is used to optimize control moves into the future to quickly and effectively hit setpoints for the three y variables. This type of control of the process will likely be necessary as opposed to more simple feedback PID methods.

Supporting Material and Code

Zipped together with this file are two .ipynb files and supporting modules (LSTM.ipynb and Informer.ipynb), and they each contain the creation, training and testing of each of the models. The informer model uses all information contained in the Informer Model folder and you should be able to run the Informer.ipynb if it is run out of the folder.

Also included is the data to run the notebooks which is called big_data_cont.mat. It needs to be uploaded into the current workspace memory of each of the notebooks for them to run.

References

[1] “The Future of Hydrogen,” *Int. Energy Agency*, 2019, doi: 10.1787/1e0514c4-en.

[2] Zhou, H., Zhang, S., Peng, J., Zhang, S., Li, J., Xiong, H., & Zhang, W. (2021, May). Informer: Beyond efficient transformer for long sequence time-series forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence* (Vol. 35, No. 12, pp. 11106-11115).

[3] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., … Chintala, S. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32* (pp. 8024–8035). Curran Associates, Inc. Retrieved from http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf