Mason Rodriguez Rand 12/12/2023

ME 249 Project 4 Report

Introduction

Similar to our previous project in optimizing solar panel systems, this new project is also focused on harnessing solar energy for power production, specifically focusing on a boiler design akin to that utilized in the PS10 solar thermal power plant near Seville, Spain. The heart of the system lies in its four riser tube arrays, each measuring 4.8 m in width and 12.0 m in height, strategically positioned within the absorber cavity to optimize the capture of solar energy.

In the first part of the project, we seek to optimize in one case, the steam exit quality and the max wall temperature. These outputs are important as they affect the system efficiency and internal system protection. The input parameters for this portion of the project include the tube inside diameter D_i *(m)*, the incident solar flux q_o'' *(kW/m^2)*, and the water mass flow rate $\dot{m}$ *(kg/s)*. In the opposite case, D_i, q_o'', x_e, *and* max wall temp T_w *(˚C)* are taken as inputs to optimize the water mass flow rate, $\dot{m}$, as the output.

In the second part of the project, in the second part, our focus shifts to understanding and optimizing the natural convection heat transfer within the system. We closely examine the flow and heat transfer from a vertical heated surface in still air, a scenario that may reflect the conditions faced within the solar receiver tower, or even those found on circuit boards.

To do so, we use a sophisticated computational fluid dynamics (CFD) model to simulate the steady flow and temperature field across a vertical surface with specific boundary conditions, akin to those found in the power plant's design. We optimize the max surface temperature ($T_{s,\,max}$) characteristics of the setup by training our model from input two heat transfer parameters q_1'' and q_2'' *(W/m^2)*, representing the heat transfer of two parts of the system along the wall, as well as the average spacing between components, Δx_s *(m)*. The ultimate aim is to utilize this neural network as both an analytical tool to discern performance trends and design an instrument to evaluate the impact of various system configurations on operating temperatures.

Separation of Tasks

This work was done independently by Mason Rodriguez Rand.

Part 1: Predicting Plant Performance Metrics and Mitigating Overheating Risk

For our first machine learning application, we explored the effectiveness of a machine learning model trained on a tube inside diameter D_i, incident solar flux q_o'', and water mass flow rate $\dot{m}$ in order to predict exit quality x_e and max tube wall temp $T_{w,\,max}$ of the system. This can be a useful application for solar thermal system operators to aid in increasing the efficiency of their system while staying within safe operating temperatures.

Program Modifications for Task 1.1

I started by determining the median values of each parameter in *ME249Proj4F23data1a.ipynb* and normalizing the input and output data over those parameters:

```
median_values_x = np.median(np.array(xraw), axis=0)
Dimed = median_values_x[0] #
Qomed = median_values_x[1] #
Mdmed = median_values_x[2] #

#exit quality, max wall temperature (deg C)
median_values_y = np.median(np.array(yraw), axis=0)
Xemed = median_values_y[0]
Twmed = median_values_y[1]

print("Median values for each column:", median_values_x, median_values_y)

xnorm = (xraw/median_values_x)
ynorm = (yraw/median_values_y)
```

I then split the data into a random selection of 3/4 of the data and 1/4 of the data for the training and test tests, respectively:

```
from sklearn.model_selection import train_test_split

# Split the data
xarray, xval, yarray, yval = train_test_split(xnorm, ynorm, test_size=0.25, random_state=42)
```

I then constructed the neural network as described in Task 1.1.

```
from keras import backend as K
#initialize weights with values between -0.2 and 0.5
initializer = keras.initializers.RandomUniform(minval= -0.2, maxval=0.5)

model = keras.Sequential([
    keras.layers.Dense(6, activation=K.elu, input_shape=[3],  kernel_initializer=initializer),
    keras.layers.Dense(8, activation=K.elu,  kernel_initializer=initializer),
    keras.layers.Dense(16, activation=K.elu, kernel_initializer=initializer),
    keras.layers.Dense(8, activation=K.elu, kernel_initializer=initializer),
    keras.layers.Dense(2,  kernel_initializer=initializer)
  ])
```

Finally, during training I turned the learning parameter via the keras.optimizers.RMSprop function as well as the number of epochs and patience as described later in the discussion.

```
#from tf.keras import optimizers
rms = keras.optimizers.RMSprop(0.0002)
model.compile(loss='mean_absolute_error',optimizer=rms)
```

```
# Add an early stopping callback
es = keras.callbacks.EarlyStopping(
    monitor='loss',
    mode='min',
    patience = 80,
    restore_best_weights = True,
    verbose=1)
# Add a checkpoint where loss is minimum, and save that model
mc = keras.callbacks.ModelCheckpoint('best_model.SB', monitor='loss',
                      mode='min',  verbose=1, save_best_only=True)

historyData = model.fit(xarray,yarray,epochs=500,callbacks=[es])

loss_hist = historyData.history['loss']
#The above line will return a dictionary, access it's info like this:
best_epoch = np.argmin(historyData.history['loss']) + 1
print ('best epoch = ', best_epoch)
print('smallest loss =', np.min(loss_hist))

model.save('./best_model')
```

Discussion and Analysis of Task 1.1

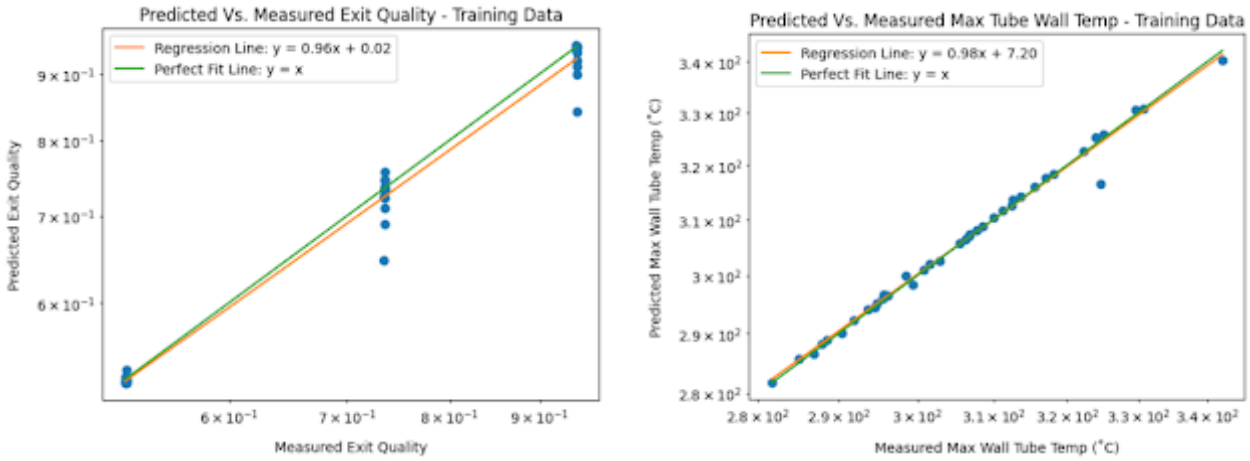
After the model was constructed from the code above, the following training process displayed in **Table 1.1** was undertaken. Results from the process in the form of model performance are displayed in **Figures 1-4.**

Table 1. Training Process 1.1

Trial Number	Epochs	Learning Rate	Loss (MAE)
1	560	.020	.0306
2	228	.0050	.0212
3	800	.00035	.0185
4	220	.00030	.0183
5	800	.00025	.0166
6	800	.00025	.0125
7	335	.00035	.00902
8	500	.0002	.00913

The training process outlined in Table 1.1 involved eight separate trials, each varying in the number of epochs and learning rates to optimize the machine learning model. The trials demonstrate a progressive fine-tuning approach, starting with a higher learning rate of 0.02 and gradually reducing it to achieve lower mean absolute errors (MAE). As the learning rate decreased across trials, there was a notable improvement in the loss, with the initial MAE of 0.0306 in Trial 1 decreasing to a lowest MAE of 0.00902 by Trial 7. The epochs ranged from as few as 220 to as many as 800, indicating a rigorous effort to balance the model's exposure to the training data against the risk of overfitting. The final trial ended with an MAE of 0.00913, slightly higher than the best result, suggesting a close optimization of the model's parameters to the given data set.

Figures 1, 2. Predicted vs Measured Exit Quality, Max Tube Wall Temp - Training Data

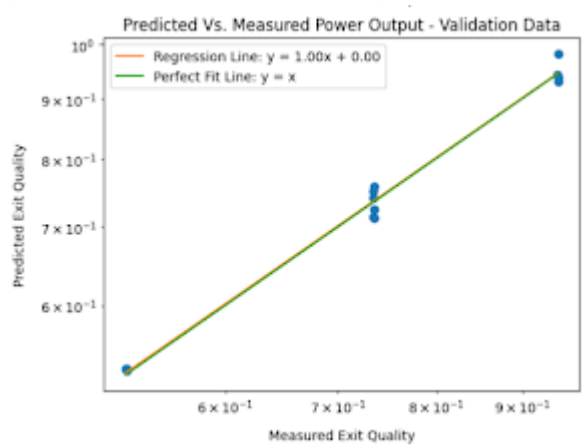


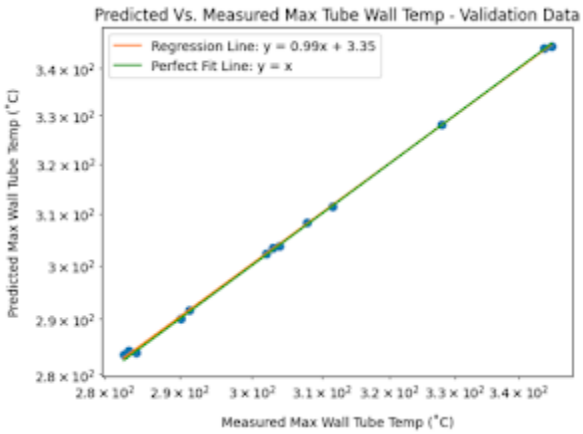
As shown in **Table 2**, for exit quality, the mean absolute error (MAE) is relatively low at 0.01230, and the mean absolute relative error (MARE) is 0.01654. This indicates that the model predicts exit quality with a high degree of accuracy, as evidenced by the regression line nearly overlapping the line of perfect fit (y = x) in the left plot. The small errors suggest that the model's predictions are consistent with the actual measured values for the majority of the training data points, making it a reliable tool for predicting this particular output.

Regarding the max wall tube temperature, the MAE is higher at 0.683, but the MARE is still very low at 0.00218, as shown in the right plot. Despite the absolute error being larger in magnitude compared to the exit quality, the relative error remains small, which indicates that the model's predictions are still proportionally close to the actual measurements. This is also reflected by the regression line closely matching the line of perfect fit. However, the higher absolute error could suggest that the max wall tube temperature measurements have a larger variance or that this output is inherently more challenging to predict accurately.

Overall, the model exhibits strong performance on the training data for both outputs. It maintains a high predictive accuracy with exit quality and manages to keep relative errors low for max wall tube temperature, despite a higher absolute error, which may require further investigation to understand the source of this discrepancy and to potentially improve the model's predictive capability for this output.

Figures 3, 4. Predicted vs Measured Exit Quality, Max Tube Wall Temp - Validation Data





As displayed in **Table 2**, the left plot, which compares the predicted vs measured exit quality validation data, shows a mean absolute error (MAE) of 0.0135 and a mean absolute relative error (MARE) of 0.0176. These error metrics are slightly higher than those for the training data, which is expected since models often perform slightly better on the data they were trained on. However, the errors are still within a reasonable range, indicating that the model has generalized well and is performing consistently when applied to new, unseen data. The regression line's slope is very close to 1, suggesting that the model's predictions are almost identical to the actual values, with very minimal systematic deviation.

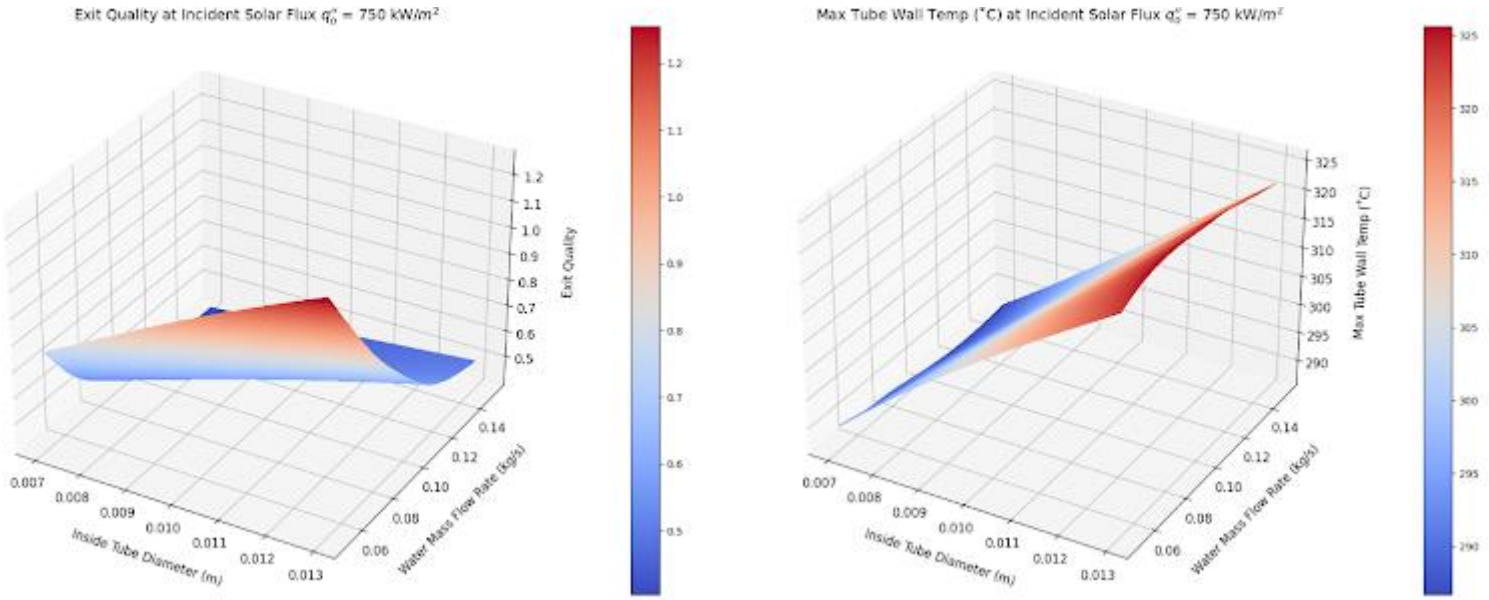
On the right plot, which depicts the "Predicted vs. Measured Max Wall Tube Temperature - Validation Data," the MAE is 0.396 and the MARE is 0.00133. Compared to the training data, the absolute error has decreased significantly, which is quite promising as it suggests that the model is not overfitting and is capable of generalizing to new data. The low relative error further affirms the model's accuracy in predicting the max wall tube temperature. The regression line again closely matches the line of perfect fit, underscoring the reliability of the model in terms of this particular output.

Overall, the model shows an ability to predict both exit quality and max wall tube temperature with high accuracy on the validation data. The slightly higher errors for exit quality as compared to the training data are not unusual and do not detract from the model's effectiveness. The substantial improvement in absolute error for max wall tube temperature when transitioning from training to validation data highlights the model's robustness and its potential applicability in practical scenarios.

Table 2. Mean Absolute Error and Mean Absolute Relative Errors for Task 1.1 Plots

Plot	Mean Absolute Error	Mean Absolute Relative Error
Exit Quality - Training Data	0.01230	0.01654
Max Tube Wall Temp - Training Data	0.683	0.00218
Exit Quality - Validation Data	0.0135	0.0176
Max Tube Wall Temp - Validation Data	0.396	0.00133

Figures 5 and 6. Surface Plots of Exit Quality at Constant Incident Solar Flux, Max Tube Wall Temp (˚C) at Constant Incident Solar Flux Versus Inside Tube Diameter and Water Mass Flow Rate



The surface plots depicted in Figures 5 and 6 illustrate the relationship between exit quality and maximum tube wall temperature at a fixed incident solar flux of 750 kW/m², as functions of the inside tube diameter and water mass flow rate. To achieve an exit quality target of approximately 0.75, while also ensuring that the maximum tube wall temperature remains at or below the critical threshold of 310 °C, a careful balance between the tube diameter and mass flow rate is necessary.

The plots suggest that the inside tube diameter should be maintained below 0.0115 meters. A diameter larger than this increases the likelihood of surpassing the desired maximum wall temperature. Concurrently, to attain the exit quality of around 0.75, the model indicates that the water mass flow rate should be approximately 0.08 kg/s, particularly when the inside tube diameter hovers near 0.011 meters. This specific mass flow rate is crucial as it plays a significant role in the heat exchange efficiency, thereby directly affecting the exit quality.

In essence, these surface plots serve as a predictive guide for system design and operation. By adhering to these parameters, solar thermal system operators can optimize their systems to maintain high efficiency and reliability, ensuring the output and temperature remain within the desired specifications.

Program Modifications for Task 1.3

I started by cleaning the *ME249Proj4F23data1b.ipynb* data in the same manner as Task 1.1.

```
median_values_x = np.median(np.array(xraw), axis=0)
Dimed = median_values_x[0] #
Qomed = median_values_x[1] #
Xemed = median_values_x[2]
Twmed = median_values_x[3] #

#exit quality, max wall temperature (deg C)
median_values_y = np.median(np.array(yraw), axis=0)
Mdmed = median_values_y[0]

print("Median values for each column:", median_values_x, median_values_y)

xnorm = (xraw/median_values_x)
ynorm = (yraw/median_values_y)
```

Next, the data was split into a training and test set.

```
from sklearn.model_selection import train_test_split

# Split the data
xarray, xval, yarray, yval = train_test_split(xnorm, ynorm, test_size=0.25, random_state=42)
```

Lastly, I tried three different models at different times, with each one detailed below.

```
from keras import backend as K
#initialize weights with values between -0.2 and 0.5
initializer = keras.initializers.RandomUniform(minval= -0.5, maxval=0.7)

#First model try
# model = keras.Sequential([
#     keras.layers.Dense(13, activation=K.elu, input_shape=[4],  kernel_initializer=initializer),
#     keras.layers.Dense(26, activation=K.elu,  kernel_initializer=initializer),
#     keras.layers.Dropout(0.05),
#     keras.layers.Dense(13, activation=K.elu, kernel_initializer=initializer),
#     keras.layers.Dropout(0.05),
#     keras.layers.Dense(1, kernel_initializer=initializer)
#   ])

#Second model try
# model = keras.Sequential([
#     keras.layers.Dense(6, activation=K.elu, input_shape=[4],  kernel_initializer=initializer),
#     keras.layers.Dense(8, activation=K.elu,  kernel_initializer=initializer),
#     keras.layers.Dense(16, activation=K.elu, kernel_initializer=initializer),
#     keras.layers.Dropout(0.03),
#     keras.layers.Dense(8, activation=K.elu, kernel_initializer=initializer),
#     keras.layers.Dense(1,  kernel_initializer=initializer)
#   ])

#Third model try
model = keras.Sequential([
    keras.layers.Dense(6, activation=K.elu, input_shape=[4],  kernel_initializer=initializer),
    keras.layers.Dense(13, activation=K.elu,  kernel_initializer=initializer),
    keras.layers.Dense(26, activation=K.elu, kernel_initializer=initializer),
    keras.layers.Dense(8, activation=K.elu, kernel_initializer=initializer),
    keras.layers.Dense(1,  kernel_initializer=initializer)
  ])
```

Model learning rate, epochs, patience, and weights were adjusted using the same mechanisms as in Task 1.1

```
rms = keras.optimizers.RMSprop(0.00035)
model.compile(loss='mean_absolute_error',optimizer=rms)
```

```
# Add an early stopping callback
es = keras.callbacks.EarlyStopping(
    monitor='loss',
    mode='min',
    patience = 80,
    restore_best_weights = True,
    verbose=1)
# Add a checkpoint where loss is minimum, and save that model
mc = keras.callbacks.ModelCheckpoint('best_model.SB', monitor='loss',
                     mode='min',  verbose=1, save_best_only=True)

historyData = model.fit(xarray,yarray,epochs=800,callbacks=[es])

loss_hist = historyData.history['loss']
#The above line will return a dictionary, access it's info like this:
best_epoch = np.argmin(historyData.history['loss']) + 1
print ('best epoch = ', best_epoch)
print('smallest loss =', np.min(loss_hist))

model.save('./best_model')
```

Discussion and Analysis of Task 1.3

Table 3. Training Process 1.3

Model Design #	Trial Number	Epochs	Learning Rate	Loss (MAE)
1	1	500	.00035	.0768
1	2-10	All ~150	.00035-.0020	.0711
2	11	280	.00035	.0593
2	12	124	.0002	.0530
2	13	172	.0002	.0428
3	14	742	.020	.0519
3	15	474	.002	.0125
3	16	800	.00035	.00784

The first model was constructed with an input shape of four and an output layer with a single neuron. The model's complexity was increased by incorporating two hidden layers with 26 and 13 neurons, respectively, and reinforced with two dropout layers with dropout percentages of 5%. These adjustments aimed to bolster the model's robustness without significantly extending the need for more data or computational time. However, the training process revealed challenges; frequent early stoppings occurred—about ten instances, each halting around 150 epochs and resulting in incremental loss reductions. Subsequent attempts, around ten in total, failed to progress, indicating that while the model

might be robust, the training time was excessive, and the available data possibly insufficient to confirm its effectiveness.

The second model streamlined the architecture by including only one dropout layer with a lowered dropout percentage of 3% to cut down on training time. It also featured simpler layers with fewer neurons, in line with our previous task 1.1. However, this configuration also encountered difficulties, with prolonged training durations and numerous stoppages. In response, we experimented with expanding the range of weight values and increasing the patience parameter, yet the results remained suboptimal, prompting a pivot to the third model attempt.

The third model discarded dropout layers entirely, maintaining the complexity of the hidden layers to ensure robustness but eliminating the associated training time complexity of the dropouts. This was after discovering an error where the dropout had been mistakenly set at a very high rate of 0.5 instead of the intended 0.05 on one of the training runs, which had significantly inflated training time. Correcting this in the subsequent model in the next section in Task 2.1 below allowed for a more timely and effective training process.

Each model iteration informed the next, allowing an iterative refinement of our approach to achieve a delicate balance between model complexity, data requirements, and computational efficiency, ultimately leading to a model capable of reliable regression predictions.

Figures 7, 8. Predicted vs Measured Mass Flow Rate - Training Data, Validation Data

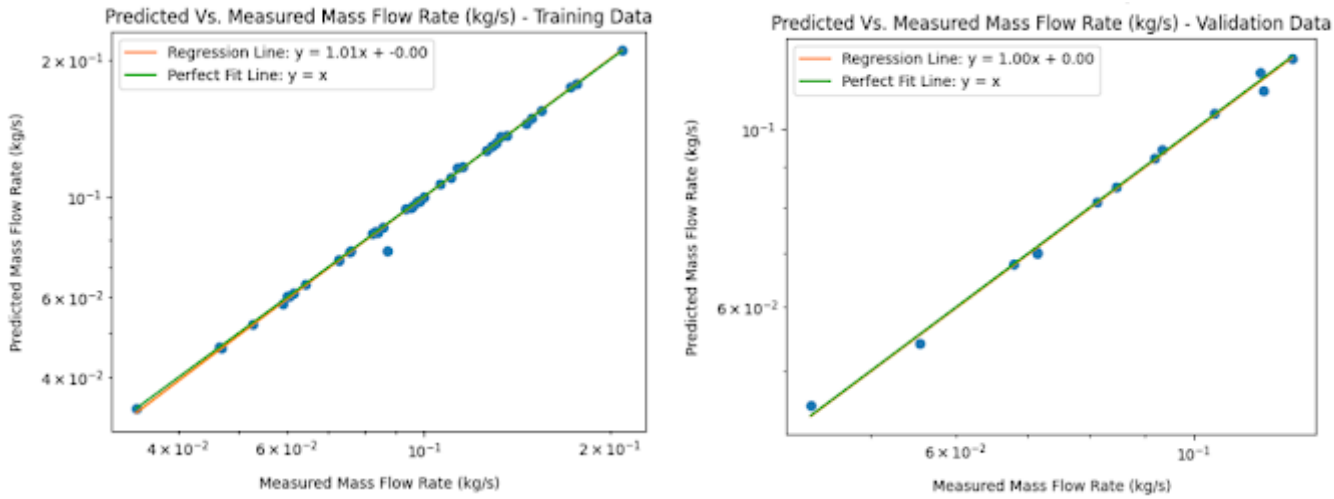


Table 4. Mean Absolute Error and Mean Absolute Relative Errors for Task 1.1 Plots

Plot	Mean Absolute Error	Mean Absolute Relative Error
Mass Flow Rate - Training Data	.000761	.00812
Mass Flow Rate - Validation Data	0.00116	.0138

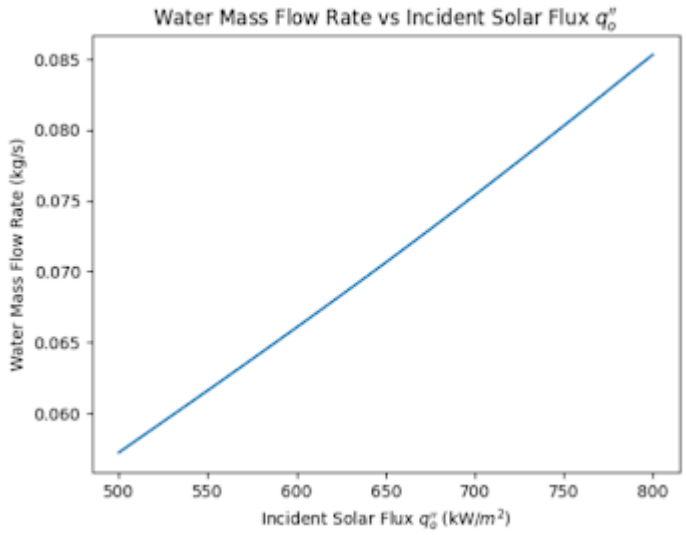
The provided data shows the model's performance in predicting mass flow rate for both training and validation datasets, as illustrated in Figures 7 and 8.

For the training data, the model demonstrates a high degree of accuracy in predicting mass flow rate, with a mean absolute error (MAE) of 0.000761 and a mean absolute relative error (MARE) of 0.00812. The regression line, with a slope of 1.01, is nearly identical to the perfect fit line (y = x), indicating that the model predictions are very close to the actual values.

In the case of the validation data, the model's predictions are still quite accurate, although there is a slight increase in the error values, with an MAE of 0.00116 and a MARE of 0.0138. Despite this increase, the errors remain low, and the regression line for the validation data also closely aligns with the line of perfect fit, reinforcing the model's consistency and reliability in generalizing to new data.

Overall, the model's ability to predict mass flow rate is robust across both datasets, showing only a marginal difference in error rates between training and validation. This suggests that the model is well-calibrated and not overfitting to the training data, capable of maintaining its predictive performance on unseen data. The low relative error in particular underscores the model's precision, making it a trustworthy tool for predicting mass flow rate in practical applications.

Figure 9. Water Mass Flow Rate vs Incident Solar Flux



The graph in Figure 9 displays the relationship between the water mass flow rate and the incident solar flux, showing a linear increase in the mass flow rate as the solar flux intensifies. This pattern suggests that as the solar energy available to the system grows, there is a corresponding need to boost the water's flow rate to absorb this extra heat, which is crucial for maintaining the designated exit quality and the maximum allowable wall temperature within a solar thermal system.

Analysis from this graph can be incorporated into a model-based control scheme for a solar thermal power plant. With a set inside tube diameter (e.g., D_i = 0.010 m), a desired exit quality (e.g., x_e = 0.70), and a maximum wall temperature limit (e.g., $T_{w,\,max}$ = 300°C), the model can offer real-time predictions for the required mass flow rate ($\dot{m}$) as the incident solar heat input (q_o") varies between 500 and 800 kW/m2.

Under the specified conditions, as the incident solar flux increases from 500 to 800 kW/m2, the model indicates a need to raise the mass flow rate. This predictive capability allows operators to adjust the water

flow rate dynamically in response to changing solar conditions, ensuring the system operates within the set exit quality and maximum tube wall temperature parameters. The linear correlation depicted suggests that the system's response to varying levels of solar flux is predictable and consistent over the range shown, which is beneficial for designing and managing the operations of solar thermal plants.

Part 2: Numerical Analysis of Boundary Layer Flow

Part 2 of the project shifts the focus toward optimizing thermal management within electronic systems. Leveraging the power of neural networks, we aim to model the complex heat transfer and temperature distribution dynamics in a two-component circuit board system. Employing a computational fluid dynamics (CFD) approach, we simulate the natural convection cooling process, crucial for maintaining component temperatures below critical thresholds to ensure reliability and prevent overheating.

The primary objective is to predict the maximum surface temperature based on a range of heat flux and device separation distance values. Such predictions are pivotal for today's electronic devices, which are expected to operate within safe thermal limits.

Program Modifications 2.1

I started by first cleaning and normalizing the data from *ME249Proj4F23data2.ipynb* around the median parameter values as done in Tasks 1.1 and 1.3.

```
#qflux1 (W/m^2), qflux2 (W/m^2), separation distance (m)
median_values_x = np.median(np.array(xraw), axis=0)
Qf1med = median_values_x[0]
Qf2med = median_values_x[1]
Xsmed = median_values_x[2]

#Wall surface temperature (deg C)
median_values_y = np.median(np.array(yraw), axis=0)
Tsmed = median_values_y[0]

print("Median values for each column:", median_values_x, median_values_y)

xnorm = (xraw/median_values_x)
ynorm = (yraw/median_values_y)
```

I then split the dataset into its respective training and tests sets.

```
from sklearn.model_selection import train_test_split

# Split the data
xarray, xval, yarray, yval = train_test_split(xnorm, ynorm, test_size=0.25, random_state=42)
```

Finally I created the model,

```
from keras import backend as K
#initialize weights with values between -0.2 and 0.5
initializer = keras.initializers.RandomUniform(minval= -0.5, maxval=0.7)

model = keras.Sequential([
    keras.layers.Dense(6, activation=K.elu, input_shape=[3],  kernel_initializer=initializer),
    keras.layers.Dense(8, activation=K.elu,  kernel_initializer=initializer),
    keras.layers.Dropout(0.05),
    keras.layers.Dense(16, activation=K.elu, kernel_initializer=initializer),
    keras.layers.Dense(8, activation=K.elu, kernel_initializer=initializer),
    keras.layers.Dropout(0.05),
    keras.layers.Dense(1,  kernel_initializer=initializer)
  ])
```

And used the same setup as before to tune the learning parameter, epochs, patience, and weights as in Tasks 1.1 and 1.2.

```
rms = keras.optimizers.RMSprop(0.0003)
model.compile(loss='mean_absolute_error',optimizer=rms)
```

```
# Add an early stopping callback
es = keras.callbacks.EarlyStopping(
    monitor='loss',
    mode='min',
    patience = 120,
    restore_best_weights = True,
    verbose=1)
# Add a checkpoint where loss is minimum, and save that model
mc = keras.callbacks.ModelCheckpoint('best_model.SB', monitor='loss',
                     mode='min',  verbose=1, save_best_only=True)

historyData = model.fit(xarray,yarray,epochs=800,callbacks=[es])

loss_hist = historyData.history['loss']
#The above line will return a dictionary, access it's info like this:
best_epoch = np.argmin(historyData.history['loss']) + 1
print ('best epoch = ', best_epoch)
print('smallest loss =', np.min(loss_hist))

model.save('./best_model')
```

Discussion of Task 2.1

Table 5. Training Process 2.1

Trial Number	Epochs	Learning Rate	Loss (MAE)
1	229	.020	.0363
2	193	.020	.0317

3	170	.00035	.0292
4	531	.020	.0348
5	543	.0020	.02215
6	800	.0003	.0154

The model design was informed by previous learnings, incorporating a simple yet effective structure with three inputs, reflecting the physical parameters, and one output, corresponding to the maximum surface temperature.

Two dropout layers were included based on successful outcomes from previous models, providing a balance between robustness and training efficiency. The use of dropout layers, set at around 5%, was instrumental in preventing overfitting, allowing the model to generalize better without the need for excessive training times. I decided to maintain the number of neurons under 20 for each layer from a past observation that simpler models achieved very favorable losses in the previous part of our project.

I also widened the minimum and maximum weight values from the range -.2-.5 to -0.5-0.7. This helped increase the "search area" of the model and overcome a loss barrier of 0.03, which I had identified as a bottleneck before modifying the maximum and minimum weight search parameters.

A review of the training trials in Table 5 reveals a clear evolution in model performance. Initial trials with a higher learning rate of 0.02 experienced higher mean absolute errors (MAE). A significant improvement was observed as the learning rate was reduced and the model refined, with Trial 6 achieving the lowest MAE of 0.0154, which indicates a well-fitting model.

Figure 10, 11. Predicted vs. Measured Max Surface Temp - Training Data, Validation Data

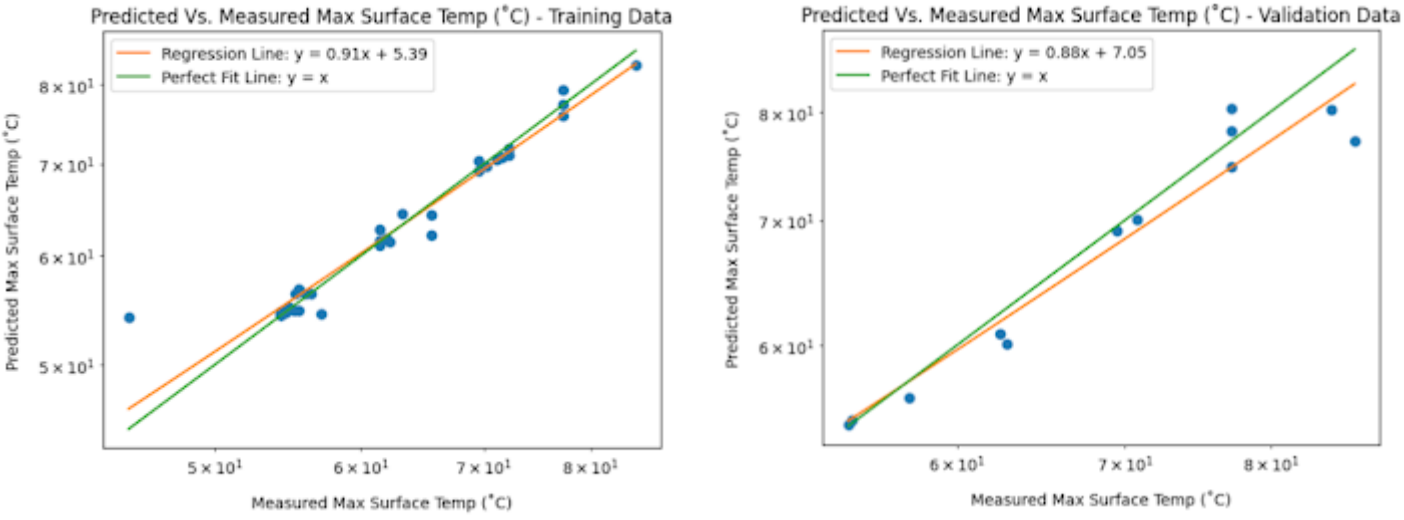


Table 6. Mean Absolute Error and Mean Absolute Relative Errors for Task 1.1 Plots

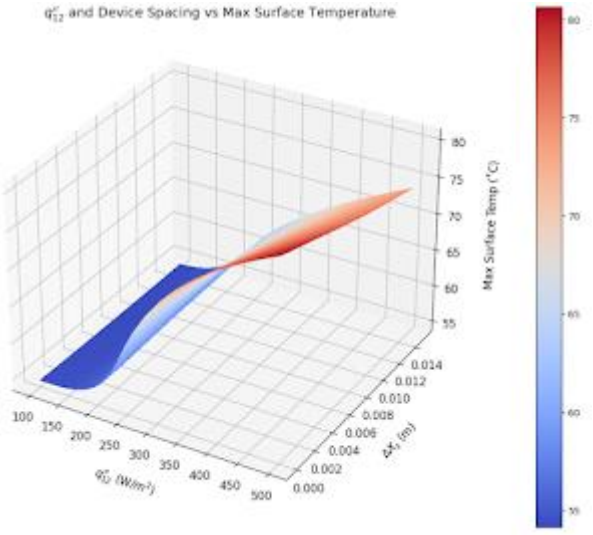
Plot	Mean Absolute Error	Mean Absolute Relative Error
Mass Surface Temp - Training Data	1.01	.0174
Max Surface Temp - Validation Data	2.25	.0295

For the training data, the regression line (y = 0.91x + 5.39) is close to the perfect fit line (y = x), which indicates that the model has a good fit to the training data with a mean absolute error (MAE) of 1.01 and a mean absolute relative error (MARE) of 0.0174. These values suggest that the model predictions are, on average, close to the actual measured temperatures, with a relatively small deviation considering the range of temperatures being predicted.

The validation data shows a regression line (y = 0.88x + 7.05) that deviates slightly more from the perfect fit line compared to the training data. The MAE has increased to 2.25, and the MARE has also increased to 0.0295. While these errors are larger than those for the training data, they are still within acceptable bounds for many practical applications, implying that the model generalizes well to unseen data. However, the higher error metrics in the validation data suggest that the model may not predict as accurately under all conditions and that there might be room for further optimization.

The slope of the regression lines being less than 1 for both training and validation indicates that the model tends to under-predict the maximum surface temperature slightly. In the context of thermal management for electronic components, this could mean that the model is conservative in its predictions, which may be beneficial in preventing overheating by suggesting earlier intervention.

Figure 12. Heat Transfer and Device Spacing vs. Measured Max Surface Temp



Based on the results of our model and this surface plot, if we want a maximum component temperature ≤72 °C, we should stick to q_{12}" values less than 450 W/m^2 and spacing in the range of 6-8mm or higher.

As the heat flux approaches 450 W/m², the spacing between components becomes increasingly critical. A spacing range of 6-8 mm or greater is recommended to ensure adequate thermal management, especially as the heat flux nears the upper limit of the safe operating range. At lower heat flux values, there is more flexibility in component spacing. However, as the heat flux increases, the spacing between components needs to be adjusted accordingly to dissipate heat more effectively. For instance, at heat flux values greater than 450 W/m², even larger spacings may be necessary to maintain safe temperature levels.

Conclusion

In conclusion, this project has demonstrated the significant potential of machine learning as applied to solar and electronic component thermal management systems. In Part 1, we successfully predicted key performance metrics of solar thermal systems, providing operators with a tool to enhance efficiency and manage temperatures. In Part 2, we extended a similar machine learning approach to model natural convection cooling, allowing for precise predictions of maximum surface temperatures and informed decisions on component spacing and heat flux.

The success of both parts underscored the importance of careful model architecture selection, the strategic use of dropout layers to prevent overfitting, and the availability of high-quality training data to ensure the reliability and accuracy of predictions. Overall, the study offers contributions to sustainable energy solutions through optimized solar thermal systems and improved reliability and lifespan of electronic devices via enhanced thermal controls. Moving forward, these models can be refined further, possibly incorporating real-time data online training for dynamic model adjustments. Future research should focus on integrating these models into live systems for adaptive control, paving the way for even more efficient and responsive thermal management solutions. This next step could greatly enhance the practical application and real-world impact of machine learning in thermal system optimization.