Project 2. **Due date: March 9, 2021**

You may team up with a partner for this project. Do not share information or results with other groups.

Part 1.

General Information

Part 1 of this project deals with comparison of a first-principles neural network implementation and the corresponding calculation using the keras Python software.

Data for this physical system can be organized in an array of the form:

$$\begin{bmatrix} | & | & | & | \\ x_{01} & x_{02} & x_{03} & y_3 \\ | & | & | & | \\ | & | & | & | \end{bmatrix}, \quad \text{example:} \quad \begin{bmatrix} 20.5 & 13.4 & 270 & 34.5 \\ 19.2 & 12.8 & 290 & 33.2 \\ 22.5 & 10.9 & 275 & 36.2 \\ 17.4 & 10.2 & 303 & 32.7 \end{bmatrix}$$

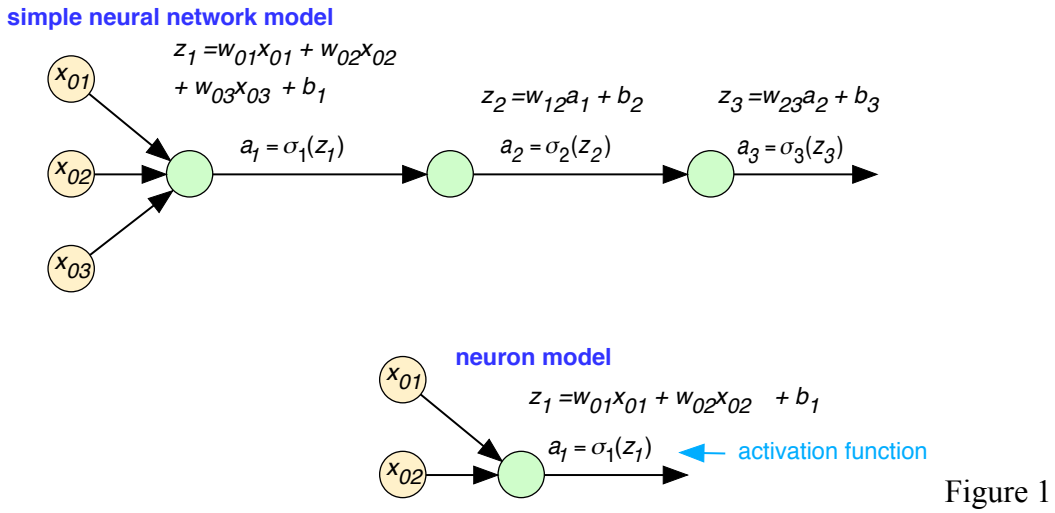

Figure 1

Backpropagation:

Squared error: $E_3 = (a_3 - y_3)^2$

We want to correct all the weights and bias values to make the squared error close to zero. When we achieve that fro a large number of data sets, we have trained the neural network to model the system.

Consider one of the weights w_{01}. If we knew the derivative $\partial E_3 / \partial w_{01}$, we could write a Newton-Raphson finite difference approximation as

$$\partial E_3 / \partial w_{01} = (0 - E_3) / (w_{01,n} - w_{01})$$

Rearranging yields a prediction of a new w_{01} value that would make the error go to zero

$$w_{01,n} = w_{01} + (0 - E_3) / (\partial E_3 / \partial w_{01})$$

However, here we have 8 weights and biases, so we only want to correct about 1/8th of the total error each. We therefore modify the above equation to

$$w_{01,n} = w_{01} + \gamma(0 - E_3) / (\partial E_3 / \partial w_{01})$$

where γ is a *learning rate* parameter that is on the order of one over the number of weights and biases in the neural network:

$$\gamma \sim \frac{1}{\text{total number of weights and biases in the neural network}}$$

If this logic is applied to all the weights and bias, the following set of relations is obtained

$$w_{01,n} = w_{01} + \gamma(0 - E_3) / (\partial E_3 / \partial w_{01}) \quad (1a)$$

$$w_{02,n} = w_{02} + \gamma(0 - E_3) / (\partial E_3 / \partial w_{02}) \quad (1b)$$

$$w_{03,n} = w_{03} + \gamma(0 - E_3) / (\partial E_3 / \partial w_{03}) \quad (1c)$$

$$b_{1,n} = b_1 + \gamma(0 - E_3) / (\partial E_3 / \partial b_1) \quad (1d)$$

$$w_{12,n} = w_{12} + \gamma(0 - E_3) / (\partial E_3 / \partial w_{12}) \quad (1e)$$

$$b_{2,n} = b_2 + \gamma(0 - E_3) / (\partial E_3 / \partial b_2) \quad (1f)$$

$$w_{23,n} = w_{23} + \gamma(0 - E_3) / (\partial E_3 / \partial w_{23}) \quad (1g)$$

$$b_{3,n} = b_3 + \gamma(0 - E_3) / (\partial E_3 / \partial b_3) \quad (1h)$$

To make this work, we have to be able to evaluate the derivatives. This is accomplished with successive application of the chain rule. For $\partial E_3 / \partial w_{01}$ this take the form:

$$\frac{\partial E_3}{\partial w_{01}} = \frac{\partial E_3}{\partial a_3}\frac{\partial a_3}{\partial w_{01}} = \frac{\partial E_3}{\partial a_3}\frac{\partial a_3}{\partial z_3}\frac{\partial z_3}{\partial w_{01}} = \frac{\partial E_3}{\partial a_3}\frac{\partial a_3}{\partial z_3}\frac{\partial z_3}{\partial a_2}\frac{\partial a_2}{\partial w_{01}} = \frac{\partial E_3}{\partial a_3}\frac{\partial a_3}{\partial z_3}\frac{\partial z_3}{\partial a_2}\frac{\partial a_2}{\partial z_2}\frac{\partial z_2}{\partial w_{01}}$$

$$= \frac{\partial E_3}{\partial a_3}\frac{\partial a_3}{\partial z_3}\frac{\partial z_3}{\partial a_2}\frac{\partial a_2}{\partial z_2}\frac{\partial z_2}{\partial a_1}\frac{\partial a_1}{\partial w_{01}} = \frac{\partial E_3}{\partial a_3}\frac{\partial a_3}{\partial z_3}\frac{\partial z_3}{\partial a_2}\frac{\partial a_2}{\partial z_2}\frac{\partial z_2}{\partial a_1}\frac{\partial a_1}{\partial z_1}\frac{\partial z_1}{\partial w_{01}}$$

The partial derivatives in the final expression above can be evaluated from the network relations shown in Fig 5b.

$$\frac{\partial E_3}{\partial a_3} = 2(a_3 - T_{db3}) \qquad \frac{\partial a_3}{\partial z_3} = \sigma_3'(z_3)$$

$$\frac{\partial z_3}{\partial a_2} = w_{23} \qquad \frac{\partial a_2}{\partial z_2} = \sigma_2'(z_2)$$

$$\frac{\partial z_2}{\partial a_1} = w_{12} \qquad \frac{\partial a_1}{\partial z_1} = \sigma_1'(z_1)$$

$$\frac{\partial z_1}{\partial w_{01}} = x_{01}$$

Substituting:

$$\frac{\partial E_3}{\partial w_{01}} = 2(a_3 - T_{db3})\sigma_3'(z_3)w_{23}\sigma_2'(z_2)w_{12}\sigma_1'(z_1)x_{01} \qquad (2a)$$

A similar analysis for the other derivatives yields the relations

$$\frac{\partial E_3}{\partial w_{02}} = 2(a_3 - T_{db3})\sigma_3'(z_3)w_{23}\sigma_2'(z_2)w_{12}\sigma_1'(z_1)x_{02} \qquad (2b)$$

$$\frac{\partial E_3}{\partial w_{03}} = 2(a_3 - T_{db3})\sigma_3'(z_3)w_{23}\sigma_2'(z_2)w_{12}\sigma_1'(z_1)x_{03} \qquad (2c)$$

$$\frac{\partial E_3}{\partial b_1} = 2(a_3 - T_{db3})\sigma_3'(z_3)w_{23}\sigma_2'(z_2)w_{12}\sigma_1'(z_1) \qquad (2d)$$

$$\frac{\partial E_3}{\partial w_{12}} = 2(a_3 - T_{db3})\sigma_3'(z_3)w_{23}\sigma_2'(z_2)a_1 \qquad (2e)$$

$$\frac{\partial E_3}{\partial b_2} = 2(a_3 - T_{db3})\sigma_3'(z_3)w_{23}\sigma_2'(z_2) \qquad (2f)$$

$$\frac{\partial E_3}{\partial w_{23}} = 2(a_3 - T_{db3})\sigma_3'(z_3)a_2 \qquad (2g)$$

$$\frac{\partial E_3}{\partial b_3} = 2(a_3 - T_{db3})\sigma_3'(z_3) \qquad (2h)$$

Note that completion of the forward calculation for a data set produces all the values on the right side of Eqs. (2a)-(2h).

As discussed in class, in neural network analysis, this correction approach described above is termed *backpropagation*.

Algorithm

To apply this correction scheme to a batch of data, the following approach was adopted:

1. For each data point (row), the first three values are input to the network. Calculations are done left to right to ultimately compute the output value a3 and the squared error $E_3 = (a_3 - T_{db3})^2$. From intermediate results of the calculations, we also compute all the derivatives in Eqs. (32a)-(32h).

2.Compute the batch average values of square error $\overline{E_3}$, and the derivatives ($\overline{\partial E_3 / \partial w_{01}}$, etc.).

3.If the batch average squared error $\overline{E_3}$ (or rms batch error = $\sqrt{\overline{E_3}}$) is small enough, the training is complete and the resulting weights and biases define the neural network model.

If the batch average squared error $\overline{E_3}$ is not small enough, new values of the weights and biases are computed using the averaged values:

$$w_{01,n} = w_{01} + \gamma(0 - \overline{E_3}) / (\overline{\partial E_3 / \partial w_{01}}) \quad (3a)$$

$$w_{02,n} = w_{02} + \gamma(0 - \overline{E_3}) / (\overline{\partial E_3 / \partial w_{02}}) \quad (3b)$$

$$w_{03,n} = w_{03} + \gamma(0 - \overline{E_3}) / (\overline{\partial E_3 / \partial w_{03}}) \quad (3c)$$

$$b_{1,n} = b_1 + \gamma(0 - \overline{E_3}) / (\overline{\partial E_3 / \partial b_1}) \quad (3d)$$

$$w_{12,n} = w_{12} + \gamma(0 - \overline{E_3}) / (\overline{\partial E_3 / \partial w_{12}}) \quad (3e)$$

$$b_{2,n} = b_2 + \gamma(0 - \overline{E_3}) / (\overline{\partial E_3 / \partial b_2}) \quad (3f)$$

$$w_{23,n} = w_{23} + \gamma(0 - \overline{E_3}) / (\overline{\partial E_3 / \partial w_{23}}) \quad (3g)$$

$$b_{3,n} = b_3 + \gamma(0 - \overline{E_3}) / (\overline{\partial E_3 / \partial b_3}) \quad (3h)$$

and the process loops back to step 1. Note that this is a *steepest descent* method, since we are using gradient (first-derivative) information to drive the solution towards the minimum in the error function. The flow chart for this backpropagation algorithm was presented in the class notes.

Implementation

The algorithm described above was implemented in the python program in the file CodeP2.1. It is set up to run for the following array of data $[x_{01}, x_{02}, x_{03}, y_3]$:

```
[[20., 13.0, 310.8, 30.99],
 [20., 14.5, 308.0, 32.4],
 [20., 15.3, 306.0, 31.4],
 [20., 13.0, 310.8, 30.99],
 [20., 14.5, 308.0, 32.4],
 [20., 15.3, 306.0, 31.4],
 [24., 13.0, 310.8, 35.49],
 [22., 14.5, 308.0, 34.4]]
```

where x_{01}, x_{02}, x_{03} are inputs and y_3 is the output. Note that this code uses the ELU activation function:

$$\sigma_i(z_i) = \begin{cases} z_i & for\ z_i \geq 0 \\ e^{z_i} - 1 & for\ z_i < 0 \end{cases}$$

In the CodeP2.1 program, note that the subscripts in the variables here are added as characters in the variable definitions. Since the pattern is fairly obvious (I think), I won't give a complete list of definitions, just some examples that illustrate the pattern:

Code variable	Write-up variable
w01	w_{01}
w01n	$w_{01,n}$
b1	b_1
b1n	$b_{1,n}$
E3ti	$\bar{E}_3$ batch total squared error
E3	E_3 batch average squared error
dE3dw01ti	sum of $\partial E_3 / \partial w_{01}$ for batch
dE3dw01	batch average $\partial E_3 / \partial w_{01}$
gam	γ
… etc.	

In this code, the learning rate parameter γ was set to 0.03 and made a bit smaller as the error got smaller (approaching the minimum).

The code prints results of each corrected set of weights and biases, and the squared error are presented as:

```
==============================
last w01, w02, w03, w12, w23:
last b1, b2, b3:
1.2395842004849886 0.3995251230041434 0.6995350552723261 0.7198562254547948 0.649860208771567
-0.15046548493782352 -0.12033514915523294 0.0797821530490986
E3 =  0.0018852257532186052 icount = 8
next ws: 1.239180605627035 0.399064501023702 0.6990839671244636 0.719716618546577 0.64972442934057
98next bs: -0.15091708613217156 -0.1206602370864072 0.07957089133827658
```

Here `icount` is the number of data points analyzed in the batch, which factors into the rms error calculation. If `E3` = 0.00188 → the rms fractional error = $\sqrt{0.00188/8.}$ = 0.015.

After the final epoch, a summary is printed comparing the y3 data to the neural network predictions:

```
x01,  x02, x03,y3(data), ypredicted:
20.0 13.0 310.8 30.99 32.04090648879553
20.0 14.5 308.0 32.4 31.964722082083668
20.0 15.3 306.0 31.4 31.90401248113656
20.0 13.0 310.8 30.99 32.04090648879553
20.0 14.5 308.0 32.4 31.964722082083668
20.0 15.3 306.0 31.4 31.90401248113656
24.0 13.0 310.8 35.49 34.72049973119962
22.0 14.5 308.0 34.4 33.30451870328571
```

Task 1.1

In the CodeP2.1 program, only the first two of equations (1) and (2) above have been implemented in code variables. Complete the implementation of the remaining equations (in the red boxes above), using the approach indicated in the table above. When this is done, change the total batch squared error cutout limit from 0.001 to 0.00035. Then run the program and see if you can adjust the parameters (gamma, number of epochs) to converge to a batch squared error less that 0.00035

When you think you have your best result, fill in the First principles portion of the tables below:

	First principles	Keras
w01		
w02		
w03		
b1		
w12		
b2		
w23		
b3		
	rms =	mae = loss =

Comparison: measured vs. two predictions

x01	x02	x03	y3 data	First Principles Predicted y3	keras Predicted y3

Task 1.2

In Task 1.2, you are to run the CodeP2.2 program, which is a keras neural network model for the ultra-simple neural network shown in Fig. 1. This program is set up to analyze the same data set as the first principles model, with the data normalized in the same way. The keras model also specifies the same activation functions for the network layers as the first principles model.

(a) Run the CodeP2.2 program as is to try to get a loss value below 0.03. To do this, run cells 1, 2 and 3 sequentially. Then set epochs = 400 in cell 4. Then make successive run of cell 4 to try to drive the loss below 0.025. When you get a last epoch with loss < 0.025, run cell 5 to analyze the results. The code in cell 5 illustrates how to use the **keras.layer.get_weights()** function to extract weights and bias values for the network after it has been trained.

Try varying the learning constant between repeated runs to squeeze down the value of the loss. After a run of cell 4, reset the learning function value in cell 3, run that cell to impellent the new value, and then run cell 4, which will continue, where it left-off, with the new learning constant value. Generally it seems to help to reduce the learning constant as the loss value gets smaller.

(b) Once you have achieved a good fit with a loss value below 0.025 (try for below 0.020 which is doable with a little effort). Try training the network with starting values that are all 50% different from the values in the original program. Roughly, how does the number of epochs to reach a loss value below 0.03 change with these initial values compared to the starting values in the original program?

(c) For your best fit (final epoch with the lowest loss value which should be less than 0.025), run cell 5 and collect the weights and bias values and the comparison of the predictions of the trained network with the data values. Use these data to fill in the keras portions of the tables above, and include the completed tables in you summary report. Also, using the data in the tables, create log-log plots of predicted y3 (horizontal axis) vs the data y3 values (vertical axis) for the the first-priciples and keras models. Include these plots inyour report.

Based on your results in the tables, how well doe the results of the two models agree? If the two models were given the same starting values for the weights and bias values, do you expect that they would yield exactly the same answer? Briefly explain your answer in your summary report.

Part 2.

General Information

Part 2 of this project deals with analysis of the hybrid solar fossil-fuel gas turbine system in the figures below.

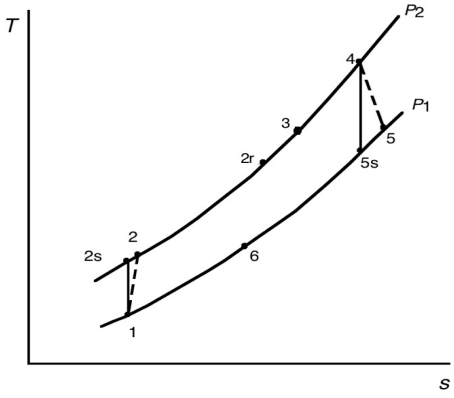

Figure 1.

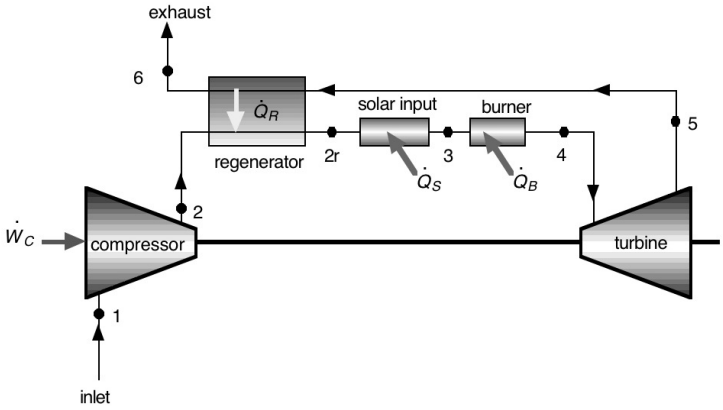

Figure 2.

In this system, air at atmospheric pressure P_1 = 101 kPa and at a temperature T_1 enters the compressor inlet at a flow rate typically about 6.0 kg/s. The air flowing out of the compressor at the high-side pressure first is heated to temperature T_{2r} in the regenerator, transferring waste heat from the turbine exhaust stream. The gas is then heated further in an exchanger that delivers solar thermal heat input, raising the temperature to T_3. Finally, the air flow into a burner where fuel is injected and burned to raise the temperature to T_4. Having T_4 as high as possible is thermodynamically advantageous, resulting in higher system efficiency for higher T_4. However, if T_4 is too high, the components of the turbine may be damaged, and for that reason, an optimal T_4 temperature is usually specified which is the highest value that can be tolerated by the blade materials in the turbine. Recent development efforts have resulting in turbine designs, with special blade materials, that can operate at 1600 K. Here the system is specified to operate at 1473 K (1200 °C), with a pressure ratio P_2 / P_1 of 14.

The burner in the system is designed to burn pure methane or a mixture of methane (CH_4) with some added propane (C_3H_8). The mixture ratio may be dictated by cost factors and availability, resulting in the mole fraction of the fuels ranging form 0% propane and all methane, to 50% propane and 50% methane, by mole. Key parameters here are the molar air-to-fuel ratio α and the fuel propane mole fraction γ defined as.

α = (moles of air)/(moles of propane and methane mixture) in inlet flow to burner
γ = (moles of propane)/(moles of propane and methane in fuel mixture)

A *stoichiometric* mixture has just enough oxygen in the air to convert all the propane and methane to H_2O and CO_2, with no oxygen left over. For arbitrary γ and a value of α greater than or equal to the stoichiometric value, the balanced chemical reaction can be written as

$$\left(\frac{\alpha}{4.76}\right)(O_2 + 3.76\ N_2) + [\gamma\ C_3H_8 + (1-\gamma)CH_4\ ] \text{--> } \left(\frac{3.76\alpha}{4.76}\right)N_2 + \left(\frac{\alpha}{4.76} - 3\gamma - 2\right)O_2$$

$$+ (2+2\gamma)\ H_2O + (1+2\gamma)CO_2$$

Note that the above equation scales the number of moles of other species on a per mole of fuel mixture basis. For stochiometric conditions,

$$\alpha = \alpha_{stoich} = 4.76(2+3\gamma)$$

The burner operates adiabatically, and the hottest exit temperature corresponds to stoichiometric conditions. For $\alpha > \alpha_{stoich}$, the exit temperature will be lower, and therefore, adding excess air (raising α) is a way of controlling the burner exit temperature.

Air inlet temperature T_1, solar heat input $\dot{Q}_S$, and the fuel mixture fraction vary during system operation, and consequently, the rate of fuel injection in the burner must be varied to hold the turbine inlet temperature T_4 at the target level of 1473 K. To facilitate model-based control of T_4, a model that predicts the required air-to fuel ratio to achieve this turbine inlet temperature under varying conditions can be used by a controller to set the fuel flow rate at the proper values. Two ways of constructing a model are:

1. Construct a physical model based on thermodynamics, fluid dynamics and heat transfer.

2. Run performance tests for the system over the expected range of operating conditions and construct a mathematical/computational fit to the experimental data using machine learning methods.

Option 1, construction of a physical model, has advantages and drawbacks. Use of a theoretical model can avoid the need for building a prototype system and obtaining test data. However, physical models generally incorporate idealizations that may not be completely accurate, and they may required knowledge of system parameters that may not be know with complete accuracy. For example, the regenerative heat exchanger may be modeled with the assumption that its heat transfer conductance is uniform throughout the unit, which may not be accurate. Also, the value of the conductance may not be known precisely. Other parameters such as the compressor isentropic efficiency and the turbine isentropic efficiency also may vary with conditions, and not be known to high accuracy. Creating an accurate model this way may be challenging.

Option 2 requires obtaining test data. However, creating a machine learning model to fit the multivariate data accounts for all the real system parametric effects and requires no assumptions or idealizations. If a good fit to the data is obtained, the predictions of the resulting model can be

accurate to the level of the uncertainty in the data and the mean absolute error or RMS error of the fit. In addition, data obtained during field operation of the systems can be used to update the model accounting for variation in its performance during its operational lifetime.

The objective of Part 2 of this project is to construct a neural network model of the performance of the gas turbine system shown in Figures 1 and 2 above. The physics of the system operation indicates that the parameters that would have to be specified to determine the performance at a given operating point from a physical model are:

pressures P_1, P_2

temperatures T_1, T_4

the compressor isentropic efficiency, η_c

the turbine isentropic efficiency, η_t

the regenerator effectiveness, ε_r

the mass flow rate of air into the compressor $\dot{m}_{air}$

the rate of solar thermal heat input , $\dot{Q}_S$

(moles of air)/(moles of propane and methane mixture) in the burner, α

(moles of propane)/(moles of propane and methane in fuel mixture) in the burner, γ

the system efficiency = $\eta_{sys} = \dfrac{\text{(turbine power out) - (compressor power in)}}{\text{combustion + solar heat input rates}}$

Here it is assumed that the system design fixes P_1 = 101 kPa, P_2 = 14*101 kPa, η_c, η_t, ε_r, and that the control scheme will adjust the air to fuel ratio to keep T_4 = 1473 K. The air flow rate into the compressor, $\dot{m}_{air}$, is assumed to be dictated by the design compressor speed that results in 6 kg/s at 298K, but varies significantly with inlet temperature (that affects air density) which may range from -5°C to 45°C.

The result is that for this system, the required air to fuel ratio α and the efficiency of the system η_{sys} are dictated by operating conditions that are primarily specified by values of T_1, γ and $\dot{Q}_S$:

$$[T_1, \gamma, \dot{Q}_S] \quad \Leftrightarrow \quad [\alpha, \eta_{sys}]$$

Task 2.1

The accompanying python code file CodeP2.3 contains two arrays `xdata` and `ydata` that contain the input (operating condition) parameters $[T_1, \gamma, \dot{Q}_S]$ and output parameters $[\alpha, \eta_{sys}]$, respectively for the system of interest. In the first cell of the skeleton code in provide file CodeP2.4, make the following modifications:

(i) Extend the five data point version of `xdata` and `ydata` to include all the data points in the CodeP2.3 file. To check they have been input correctly, print out the arrays in dimensional form before the normalizing them in the next step.

(ii) Determine the median value for each of the parameters in `xdata` and `ydata` (T_1, γ, $\dot{Q}_S$, α, and η_{sys}), Then, modify each array, replacing the raw data value by the raw data value divided by its respective median value. When this is complete, the new `xdata` and `ydata` arrays should contain data that has been normalized with the median value for that variable. As a final step, create print statements that print the normalized `xdata` and `ydata` arrays when cell 1 of the program is run.

Task 2.2

(a) In the second cell of the code in file CodeP2.4 , create a sequential neural network with three input variables (`input_shape=[3]`) in the first (hidden) dense layer of the network list. Initially set the number of neurons to 16 in this layer and set the activation function to 'K.relu.

Then add two more hidden dense layers: the second hidden layer of the network with 32 neurons and the third hidden dense layer with 16 neurons. In both of these layers, set the activation function to 'K.relu'. Finally, add one more output dense layer with 2 neurons (one for each output) and do not specify an activation function for the output layer.

When you are finished, the list of layers in your sequential network should have 16, 32, 16, and 2 neurons. Run the first and second cells of the modified CodeP2.4 code. You should get no error messages after the second cell if things are in order.

(b) In the third cell, just change the learning parameter to 0.001, and run that cell to check that no error message appears. In the 4th cell, set epochs equal to 600. Then run the first four cells in sequence. The fourth cell will train the network by successive forward passes for each point in the data set, and a backpropagation pass to update the weights and biases at the end of each epoch. For each epoch, the reported loss value indicates the mean absolute error. Since our normalized parameters make the data values of order 1, we want to achieve a value of mean absolute error that is small (a few percent) of one (~0.05 or less).

Note you can run cell 4 repeated times, and each time it will beginning with the constants from the last epoch in the previous run, so this is a way to extending the number of epochs to try to further reduce the mean absolute error. If you go back and start with cell 1, it starts over completely. Run cell 4 repeated times and try to get as low a loss value as possible (below 0.04 if possible). Report the resulting lowest loss value in your written summary. Fix γ equal to 0.25 and use the model to predict the variation of α for $268 < T_1 < 318$ K and $500 < \dot{Q}_S < 2500$ kW. Use the results to create a surface plot for alpha as a function of T_1 and $\dot{Q}_S$ over these ranges.

Task 2.3

(a) Cell 5 in the CodeP2.4 code indicates how to predict values of α, and η_{sys} for a submitted

set of the input variables T_1, γ, $\dot{Q}_S$. Below is a table of test data (distinct from the training data). Note that you have to normalize the input data using the same median values as the trained model, insert the normalized values into an array, and submit the array to the `model.predict` function. The output results must be multiplied by the median values using in training to get physical output values.

Test Data

$[T_1, \gamma, \dot{Q}_S]$	$[\alpha, \eta_{sys}]$
[318.0 , 0.0 , 500.0]	[35.13 , 0.3808]
[318.0 , 0.0 , 1500.0]	[47.46 , 0.3930]
[318.0 , 0.0 , 2500.0]	[73.12 , 0.4061]
[318.0 , 0.25 , 1500.0]	[66.34 , 0.4098]
[318.0 , 0.5 , 500.0]	[63.09, 0.4154]
[318.0 , 0.5 , 1500.0]	[85.23 , 0.4197]
[318.0 , 0.5 , 2500.0]	[131.32 , 0.4242]
[303.0 , 0.0 , 1000.0]	[38.99 , 0.4012]
[303.0 , 0.0 , 2000.0]	[53.80 , 0.4136]
[303.0 , 0.25 , 1000.0]	[54.51 , 0.4215]
[303.0 , 0.25 , 2000.0]	[75.22 , 0.4290]
[303.0 , 0.5 , 1000.0]	[70.04, 0.4337]
[303.0 , 0.5 , 2000.0]	[96.65, 0.4382]
[288.0 , 0.0 , 500.0]	[33.45 , 0.4091]
[288.0 , 0.0 , 2500.0]	[60.80 , 0.4334]
[288.0 , 0.25 , 2500.0]	[85.044, 0.4477]
[288.0 , 0.5 , 1500.0]	[77.56 , 0.4516]
[268.0 , 0.0 , 1500.0]	[40.68 , 0.4383]
[268.0 , 0.25 , 2000.0]	[65.24 , 0.4628]
[268.0 , 0.5 , 2500.0]	[98.23 , 0.4760]

Plot α for the test data versus the corresponding predicted α values on a log-log plot to evaluate the agreement. Also determined the rms deviation between the predictions and this collection of test data.

(b) Over the course of a typical summer the day, the table below indicates the anticipated combinations of air inlet temperature and solar heat input. For this variation, determine from the model the variation in air to fuel ratio with time for γ = 0 and 0.5. Note that you have to normalize the input data using the same median values as the model. Insert the normalized values into and array and submit the array to the `model.predict` function. Correct the predicted α values for the normalization factor and plot the predicted α values versus time over the course of the day.

Daylong Variation of Operating Conditions

Time of day (24 hr clock)	$[T_1,\ \gamma,\ \dot{Q}_S]$ (K, -, kW)
09:00	[287.0 , 0 to 0.5, 500.0]
10:00	[295.0 , 0 to 0.5, 750.0]
11:00	[301.0 , 0 to 0.5, 1000.0]
12:00	[305.0 , 0 to 0.5, 2450.0]
13:00	[307.0 , 0 to 0.5, 2600.0]
14:00	[308.0 , 0 to 0.5, 2400.0]
15:00	[308.0 , 0 to 0.5, 2100.0]
16:00	[305.0 , 0 to 0.5, 1800.0]
17:00	[295.0 , 0 to 0.5, 1300.0]
18:00	[292.0 , 0 to 0.5, 800.0]
19:00	[295.0 , 0 to 0.5, 250.0]

Task 2.4

Consider the neural network model trained in Task 2.2 to be you baseline neural network design. For this Task, repeat the network training in Task 2 for the following modifications to the baseline neural network:

(i) the baseline neural network with the relu activation function changed to ELU

(ii) the baseline neural network with an added layer and the number of neurons in the layers set to 16, 32, 16, 16, 2 (activation functions will be relu for all, as in the baseline)

(iii) the baseline neural network with the numbers of neurons changed to 8, 16, 8, 2

(iv) the baseline neural network with the number of neurons in the layers set to 20, 40, 20, 2.

Summarize your results for the models in a table indicating the minimum loss vale obtained for each model variation, and whether convergence to the minimum loss value took a larger or smaller number of epochs.

Project 2 Tasks to be divided between coworkers:

(1) Program modifications for Part 1
(2) Interpretations of results for Part 1
(3) Data prep for Part2

(3) Program modifications for neural network modeling in Part 2
(5) Plotting and analysis of the results for Part 2
(6) Write-up of the results and conclusions

Deliverables:
Written final report should include:

(1) Written summary of how the work was divided between coworkers.

(2) An assessment of the results comparisons for the first principles program and the keras neural network program.

(3) Plots requested in Parts 2

(4) A assessment of neural network design variations explored in Part 2.

(5) Conclusions regarding the best deign features for the network for the application in Part 2.

Grade will be based on:
(1) thoroughness of documentation of your analysis
(2) accuracy and clarity of interpretation
(3) thoroughness of the design variations investigations and the documentation of the reasons for your assessments.

Summary report due: March 9, 2021