Project 2. **Due date: October 20, 2022**

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

Part 1. Code files used in Part 1: CodeP2.1F22.pynb, CodeP2.2F22.pynb

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:

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

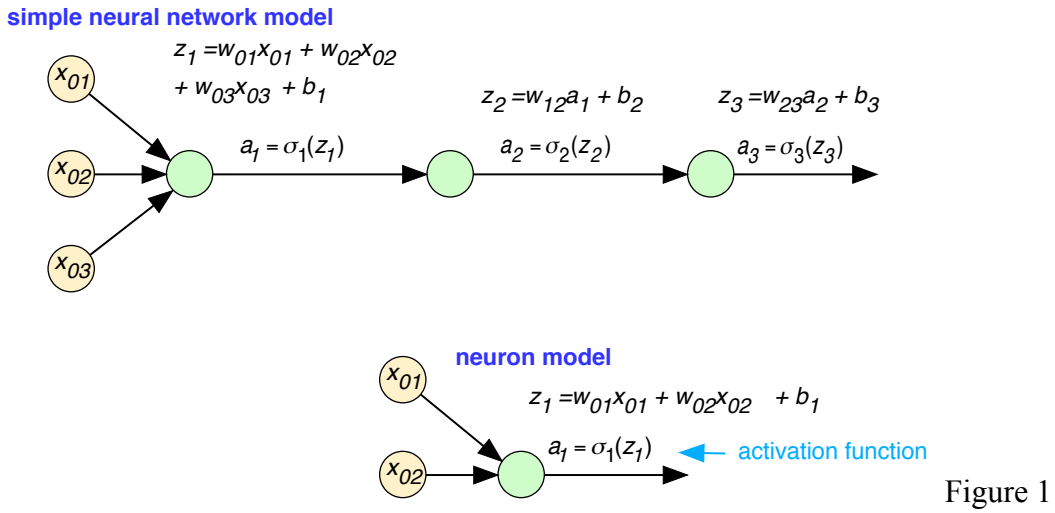

Figure 1

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 for 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 1.

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

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

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

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

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. (2a)-(2h).

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 CodeP2.1F22.pynb. 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 CodeP2.1F22 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 below. Note these are example numbers to show the format, and are not actual results.

```
==============================
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, as shown below. (Again these are example numbers to show the format.)

```
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 CodeP2.1F22 program, only the first two of equations (1) and (2) above have been implemented in code variables. Complete the implementation of the remaining equations (those 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. Note that the initial values for the weights and biases are specified. Keep these initial values fixed and adjust hyperparameters gamma, and number of epochs to get the best fit possible using them for these initial values.

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

Note: Each member of your group should do **Task 1.1 and Task 1.2(a) and 1.2(b)** individually on their computer and a pair of these tables should be included for each member of you group in the project report. In doing so, identify the name of the student associated with each set of tables. Results may differ slightly if different computer type and/or operating systems are used.

Task 1.2

In Task 1.2, you are to run the CodeP2.2F22.pynb 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 used in Task 1.1, 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, and the initial values of the weights and biases are set specifically to be the same as the first principles model.

(a) Run the CodeP2.2F22 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 runs of cell 4 to try to drive the loss below 0.014 (or as close as you can get to 0.014). **Do not change the specified initial weights and biases (so they are the same as those used in Task 1.1).** When you get a last epoch with loss < 0.014, 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. Note that code has been included in the 4th cell to save the lowest loss results among the epochs.

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 rate value in cell 3, run that cell to implement 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 rate constant as the loss value gets smaller.

(b) 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.

Note that all group members must do Task 1.1 and Task 1.2(a) and 1.2(b) and include their tables in the report.

(c) Using the first principles and keras models on one of the computers in your group, do the following:

Using the data in the tables, create log-log plots of predicted y3 (horizontal axis) vs the data y3 values (vertical axis) for the first-principles and keras models. Include these plots in your report.

Based on your results in the tables, how well do the results of the two models agree? For the two models given the same starting values for the weights and bias values, do they yield exactly the same answer? Does the answer to this question make sense? Briefly explain your answer in your summary report.

(d) Using the selected first-principles and keras models, try training the network with starting values that are all 20% higher than the values in the original program. Again, run the training to achieve a fit with a loss value below 0.025. Create a table comparing the weights and biases for these starting values with those for the original starting values (that were the same as the first principles model). What do you conclude about the sensitivity of the trained model weights and biases to the initial conditions?

Summary report for Project 2 due: **October 20, 2022**