Assignment 2

Neural Network, Computational Intelligence Assignment 2 - By B00134706 - Sarah McCabe

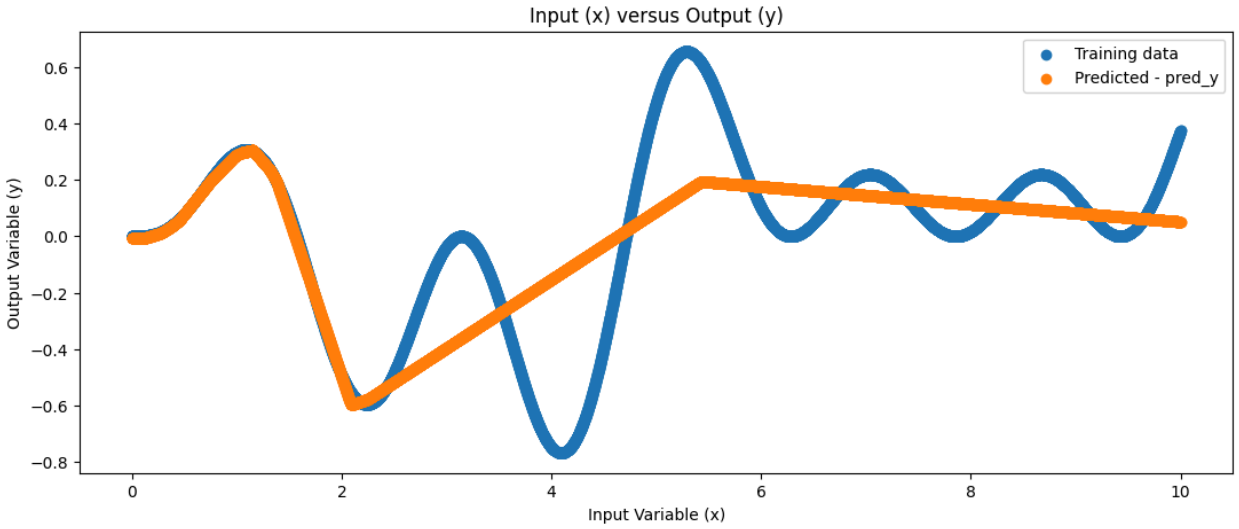
Steps taken to build the network

The first part of setting up the project was downloading and running the template files from Moodle. For this, I used VS Code, Jupiter Notebook, Python, and a virtual environment.

The parts of the template code that were edited were the train file supplied, where the number of epochs, the number of layers, the learning function, and the learning rate were edited.

Initially, the template code only used 10 Epochs this was too little so it was updated to 150 for the final model version. The different values tried where 150 was the second value tried which worked very well however 50, 250, 300, 350, and 500 were all tested. 150 was settled on as the best option as it provided the best overall results because more than 150 seemed to cause overfitting of the dataset, and both 10 as well as 50 did not seem to be enough for the neural network to train on the data properly, causing underfitting.

The learning algorithm was changed from the initial Standard Gradient Descent (SGD) to Adam. This was because SDG seemed to just provide worse results an example of using SDG with the final model values chosen for everything else can be seen in the screenshot below.

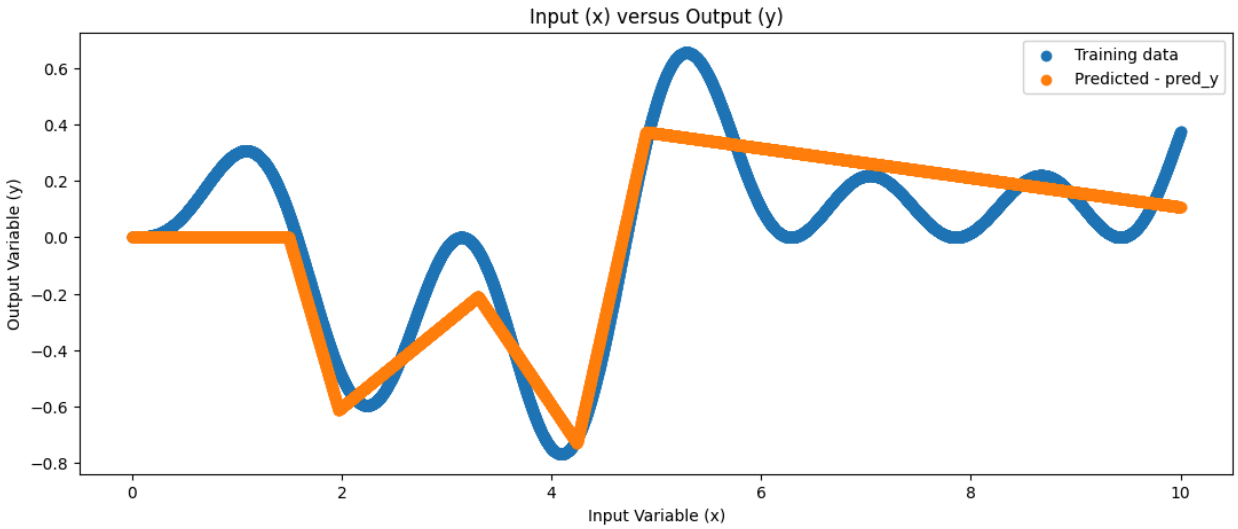



```
opt = SGD(learning_rate=0.01, momentum=0.01)
```

Instead of the final Adam learning algorithm chosen `opt = Adam(learning_rate=0.01)`

The learning rate was experimented with quite a lot because it seemed to provide the best changes in the network when being trained on the data. The initial learning rate being used was 0.1, as this was the value in the template code, and it was a good place to start however, upon experimenting with lowering this value, a much better result was gotten from the learning rate being lowered to 0.09. The result was just a straight line no matter what I changed about the layers but when I used two dense layers, one was a hidden layer and one was a normal layer the normal layer had 8 neurons and the hidden one had 16. With the updated learning rate of 0.09, with 150 epochs and Adam this provided the results shown in the screenshot below.

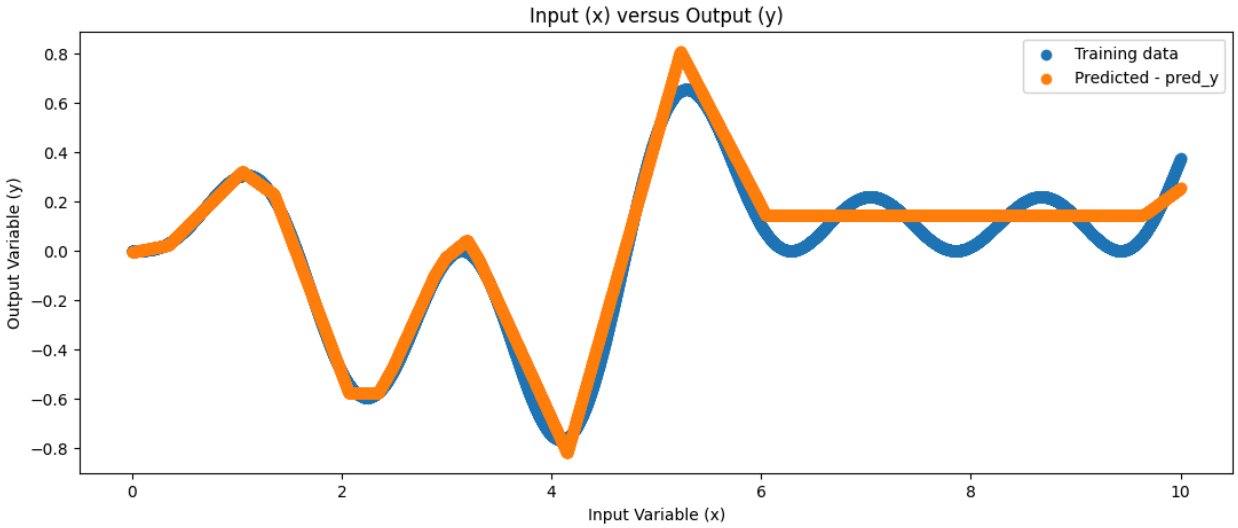
```
model.add(Dense(8, input_dim=1, activation='relu',
kernel_initializer='he_uniform'))
model.add(Dense(16, activation='relu', kernel_initializer='he_uniform'))
```



This was the first time the result was changed from a straight orange line for me so this showed that the learning rate being lower provided better results. I think this was due to the model needing longer to learn and adapt, so the lower value helped however, with how jagged the peeks seemed to be, it still seemed like it was having a difficult time learning, so a lower learning rate of 0.01 was attempted for the next iteration. This was tested with a small improvement. However, when the layers and neuron numbers were changed to be

```
model.add(Dense(16, input_dim=1, activation='relu',
kernel_initializer='he_uniform'))
model.add(Dense(16, activation='relu', kernel_initializer='he_uniform'))
```

The upgrade was much better than testing with the original 8 in the 0.09 learning rate tested, this did indicate that perhaps the model needed more complexity, however when tested with 32 it seemed to again lower the accuracy results and higher the error level. The shot below shows the updated results from the changes suggested above.

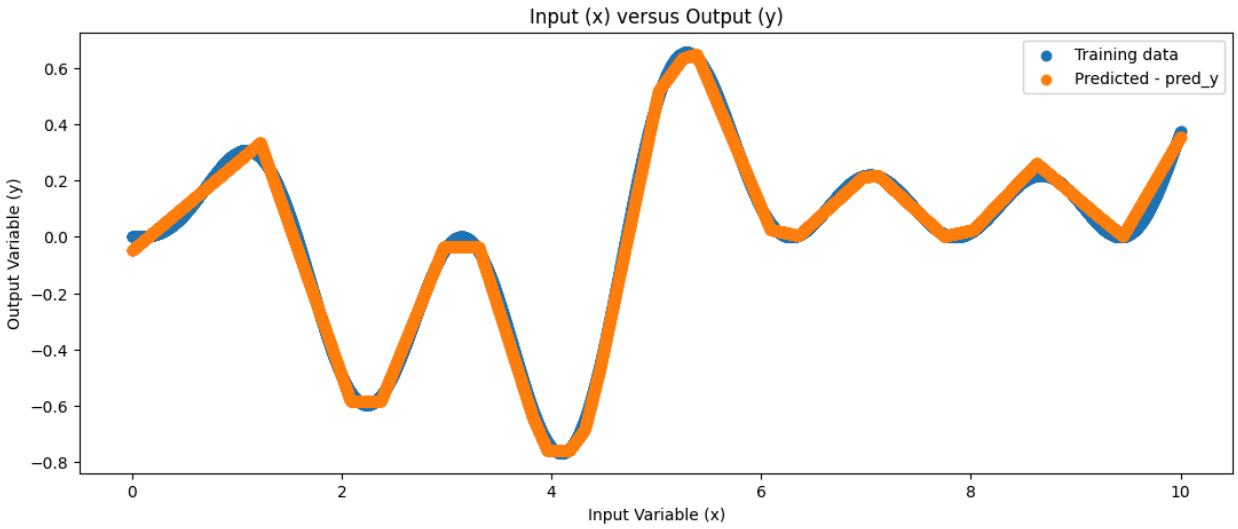


As seen, this was a very close result. However, the end still seemed to not be captured correctly, with it still just being a straight line, so as the 32 and even 128 or more neurons seemed to provide worse results, adding another hidden layer to increase complexity was tested. This was originally with the value of only 4 or 8 neurons however it was better but 16 seemed to work much better with this being the final model chosen.

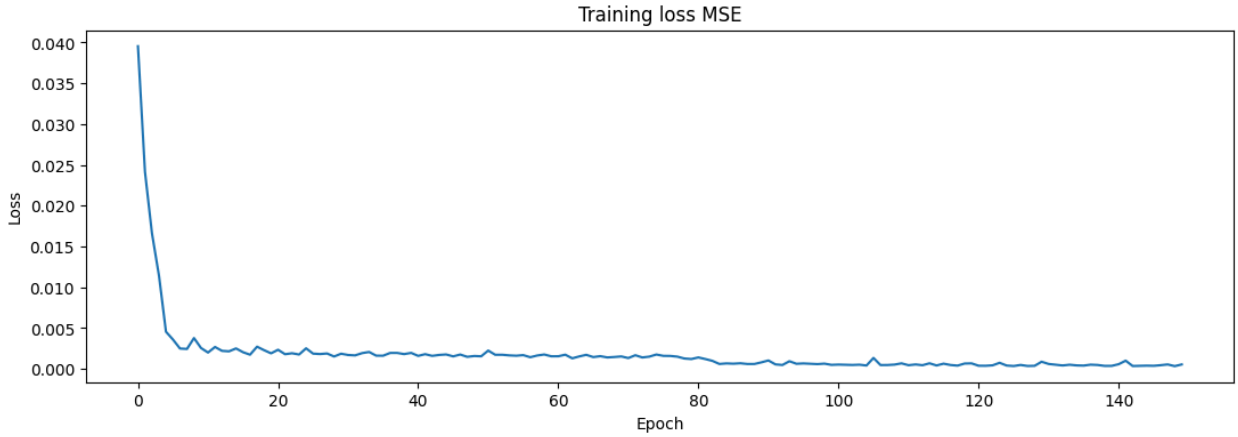
```
model.add(Dense(16, input_dim=1, activation='relu',
kernel_initializer='he_uniform'))
model.add(Dense(16, activation='relu', kernel_initializer='he_uniform'))
model.add(Dense(16, activation='relu', kernel_initializer='he_uniform'))
```

This is the results when using this as the final version

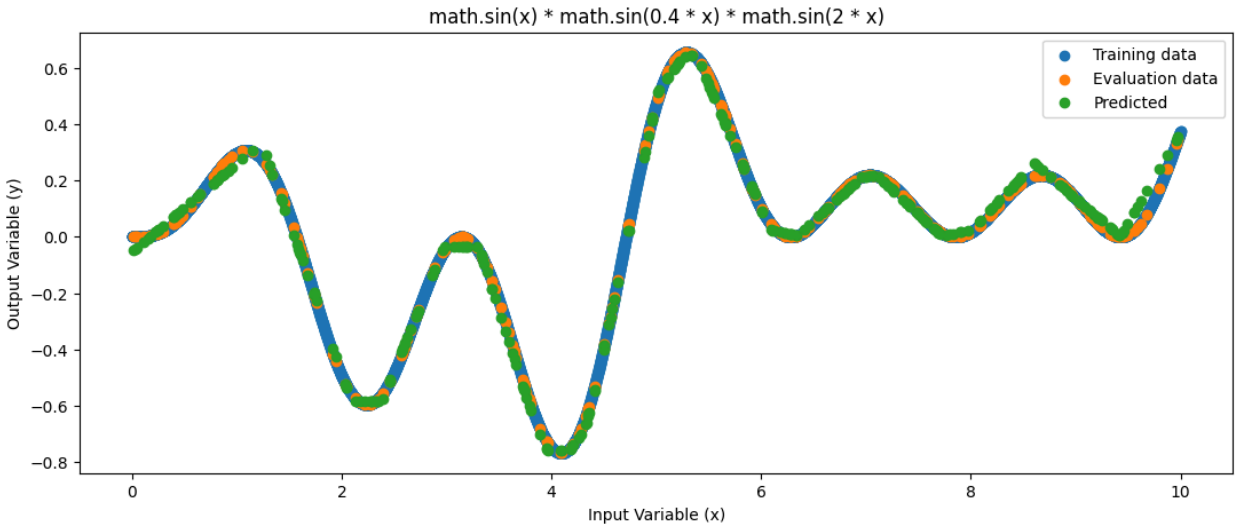
```
Training, please wait.........
Training finished...in  38.758201599121094  seconds.
```



Here is the epoch graph for the final version



As this was the final version, this was the version I used for the evaluation the graph from this can be seen below.



The average mean squared error from 10 runs of this version is, 0.0014376. With the lowest mean squared error when testing running the model being 0.000197.

Question 1

This is a regression problem, and regression problems are different to classification problems because the goal is to predict a continuous output like a real number rather than predicting a class label or category for every input.

Question 2

Sigmoid can't be used with this problem because sigmoid only predicts between 0 and 1 while we are looking to use the range of 0 and 10. So, using a linear type activation function like the Rectified Linear Unit (ReLU) activation function is an alternative that would work.

Question 3

The output layer can not have an activation function because the output layer node needs to be linear output. Linear output is necessary with regression tasks because the linear function between the input and output is important when predicting linear tasks.