**Summer 2022 Deep Learning**

**Report of Lab #1**

Student name: 劉子齊 Jonathan

Student ID: 311605004

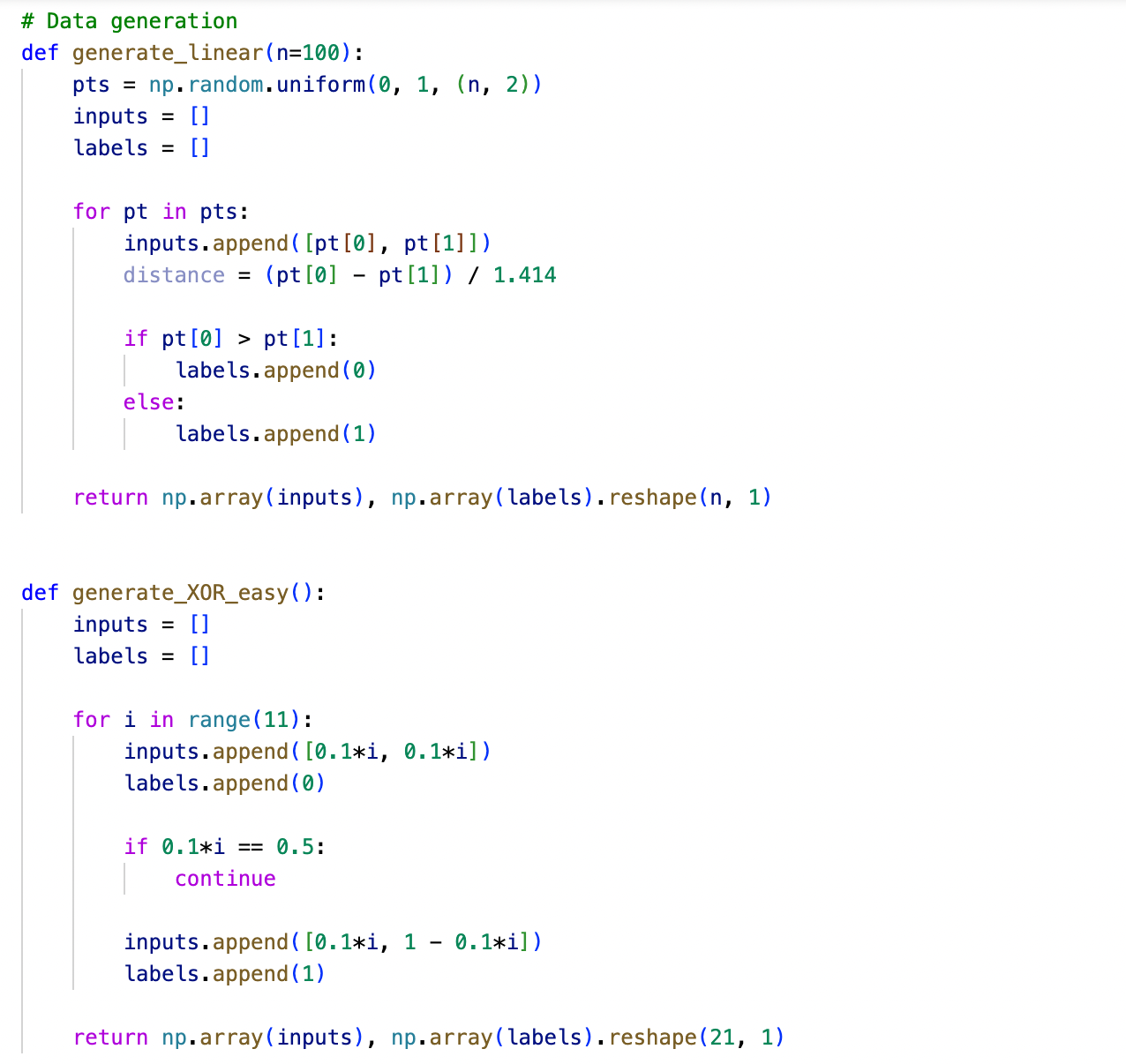
**Part 1: Introduction**

In this lab, my task is to implement a simple neural network without common frameworks, such as “TensorFlow” or “PyTorch”, but through “NumPy” and other standard libraries. Also, the simple neural network will be implemented with forward pass and backpropagation using two hidden layers, as the structure shown in the following figure.

Diagram

Description automatically generated

Besides, I would like to introduce the dataset I generated for this lab. As the instruction recommended, I generated the dataset through the given functions as below.



Graphical user interface

Description automatically generated with low confidence

To make sure the data generated reached the required distributions, I constructed the following “show\_data” function to show the distribution of the data I generated.

Text

Description automatically generated

For each experiment conducted, I will show the distribution of the data generated for the experiment in every experiment. The figure below was the data distribution of one of the experiments conducted, which is similar and obviously reached the requirements of data generation.

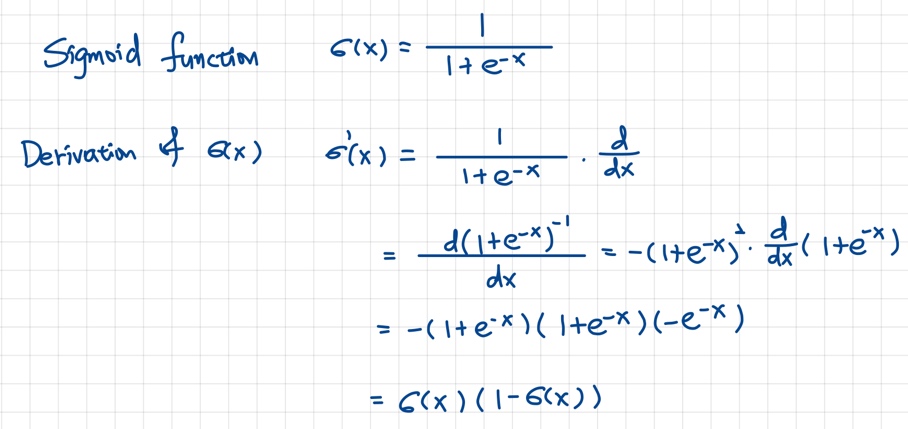
Chart, scatter chart

Description automatically generated

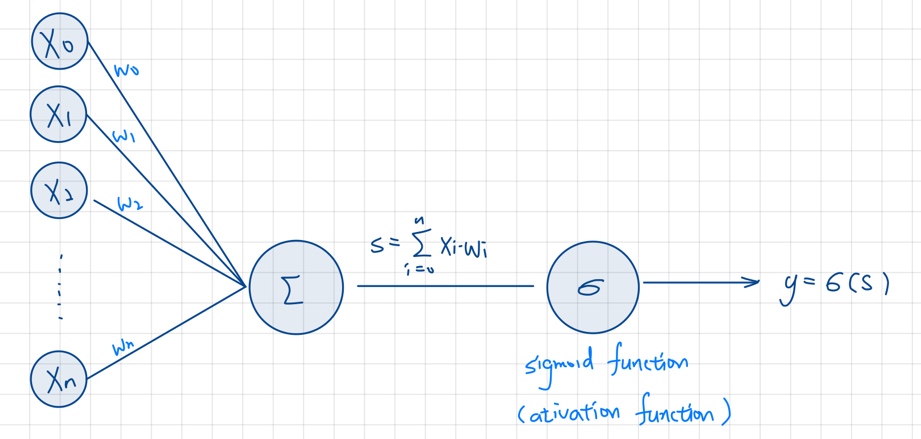
**Part 2: Experiment setups**

**Part 2-A: Sigmoid Functions**

In this lab, as the instruction recommended, I took Sigmoid function as the activation function, which is derived as the following mathematical notation.



The figure below shows the role of the activation function, which is the sigmoid function in this case, plays in the layer of the neural network.



The following figure is my implementation of the sigmoid function taking the TA’s hint as a reference. Besides, to take a deeper peek into sigmoid functions, I also drew it out through matplotlib, and the result is shown in the figure below.

Chart

Description automatically generated

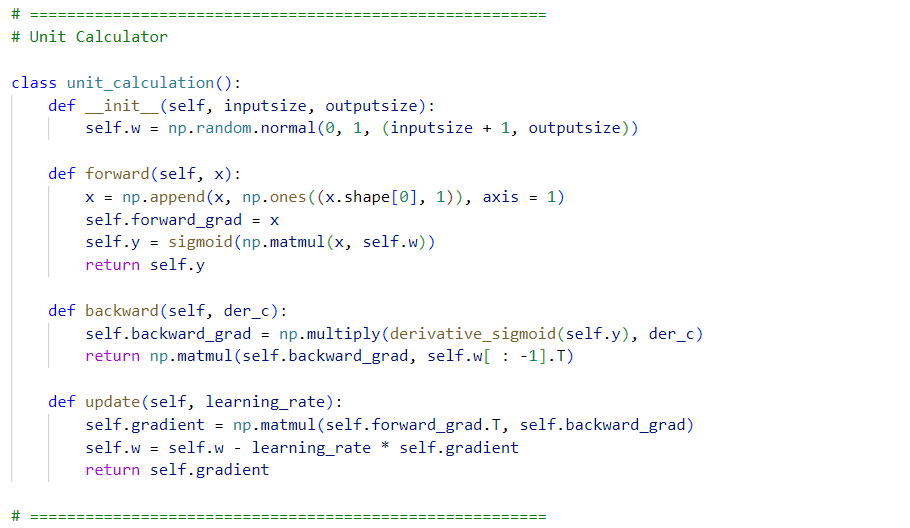
Graphical user interface, text, application

Description automatically generated

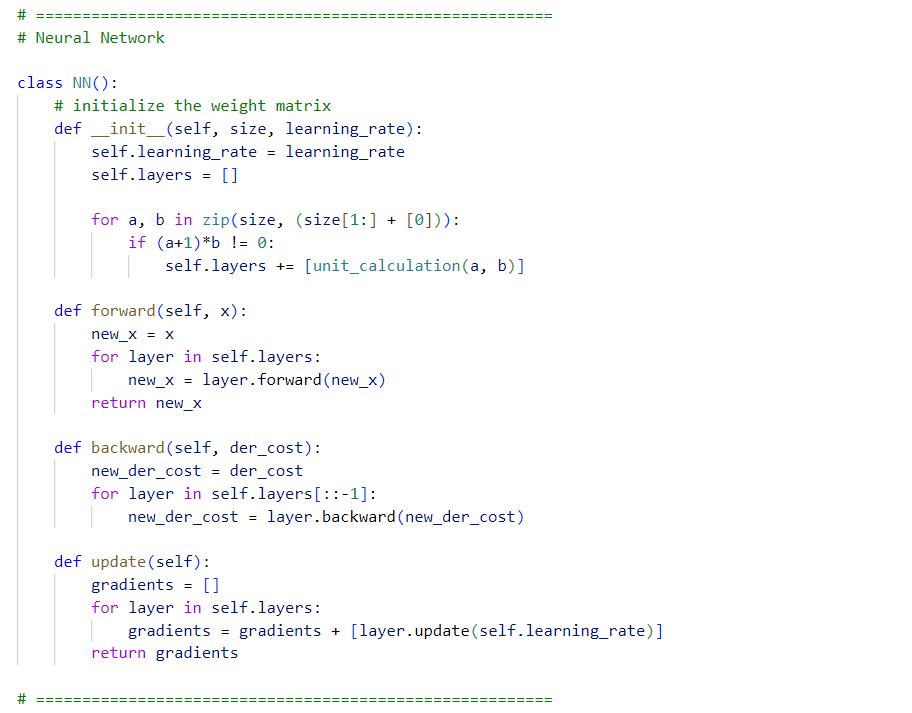
**Part 2-B: Neural Network**

For the main part of this lab, which is the neural network, I divided it into two parts, the unit calculation part, and the neural network part.

In the unit calculation part, this is where the input vector “x” gets the output scalar “y”. To multiply the output scalar “y”, I extend the unit calculator which will be the neural network. The way I implemented the unit calculator is as the following capture of the “unit\_calculator” function.



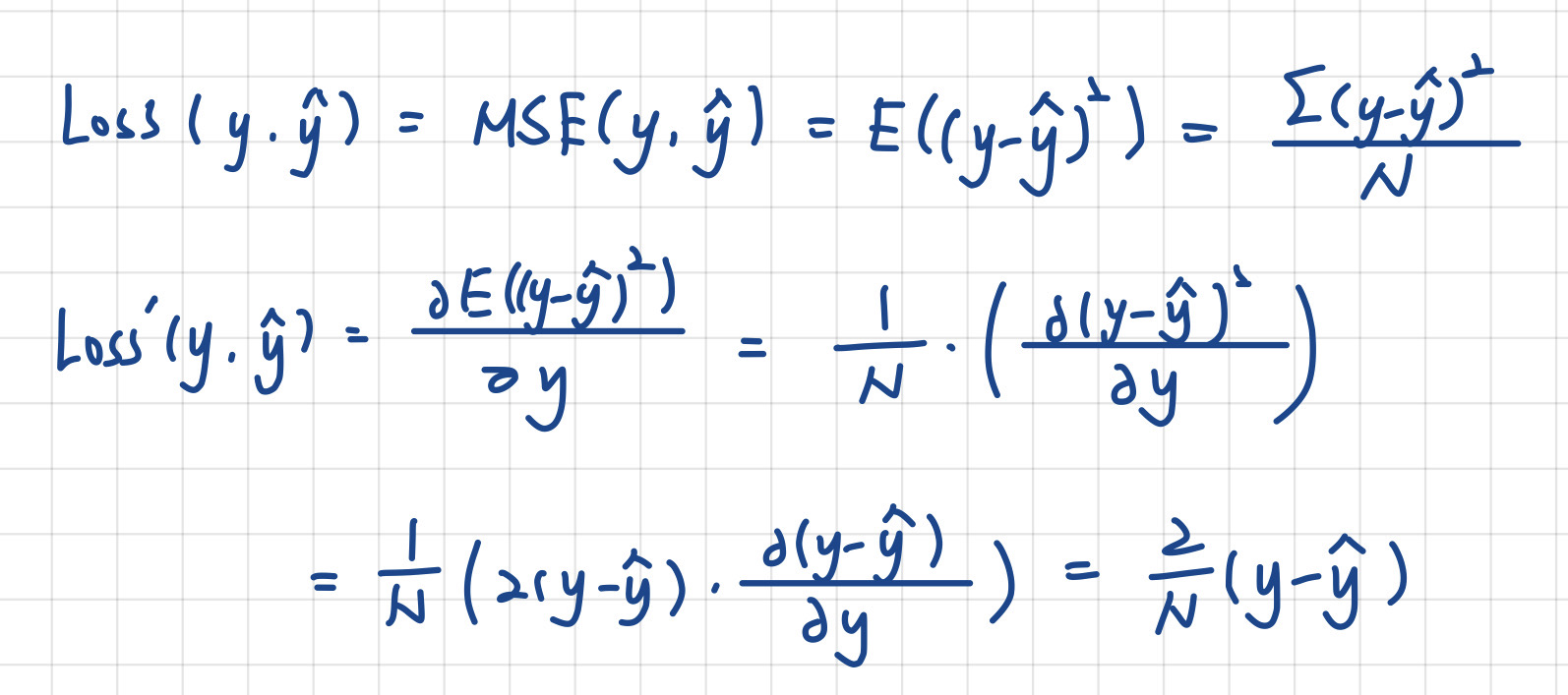
With the unit calculator function above, next we’ll move onto the main part, which is the neural network extended through the unit calculator. As shown in the following figure, this is how I implemented the neural network. Like I mentioned earlier, a unit calculator can only output a scalar, I put N units in the layers to output N scalars.



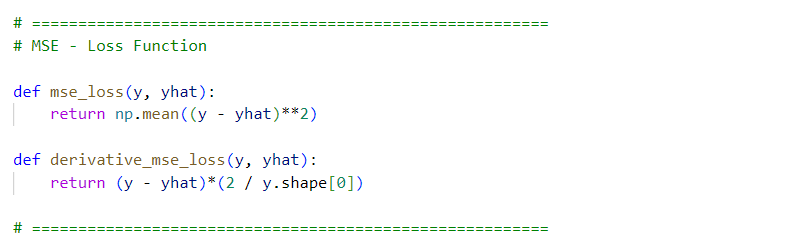
**Part 2-C: Backpropagation**

As we all know, backpropagation is to update the weight matrix. In the beginning, I initialized all the weight parameters in the network randomly. The goal here in backpropagation is the minimize the cost from the loss function below.

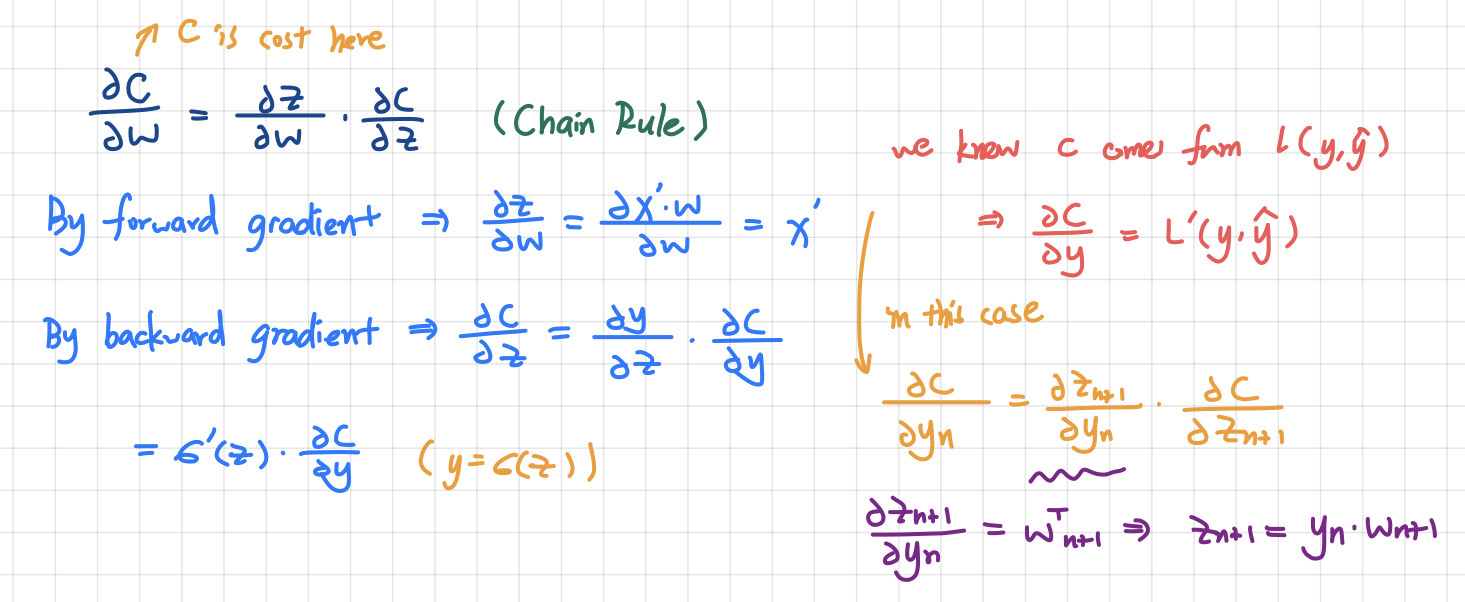
For the loss function, I chose to use the Mean Square Error, which was mentioned in class, as my loss function. In the mathematical perspective, I implemented it as below.



Through the derived function, I implemented by the way in the figure below.



With the “mse\_loss” and the “derivative\_mse\_loss” functions defined, our goal is to minimize the cost. We already know that we update the network weight through gradient descent. We can see the mathematical computation in the figure below.

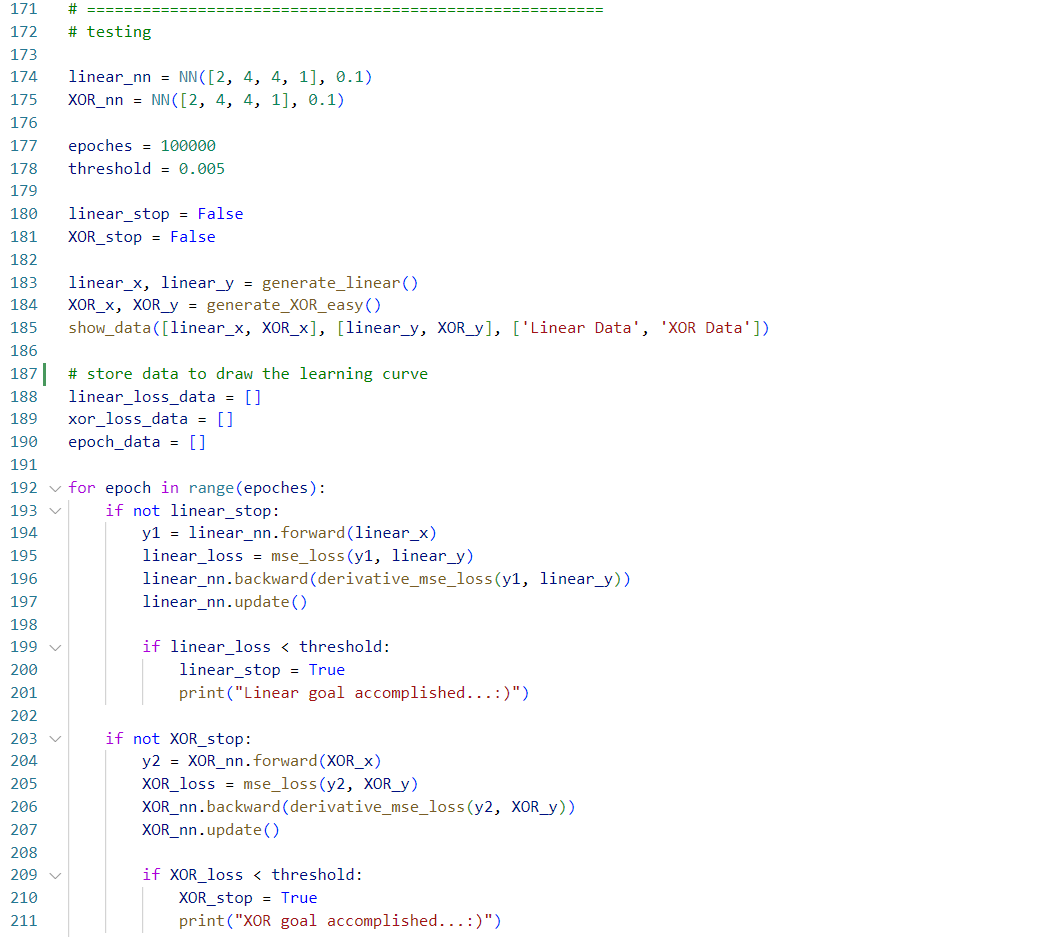


Through the mathematical calculation above, we can see the output y of the current layer is the input for the next layer. In the final “purple” formula, we get that we compute the output layer first, then send the parameters to the previous layer. Hence, we can compute  in every layer.

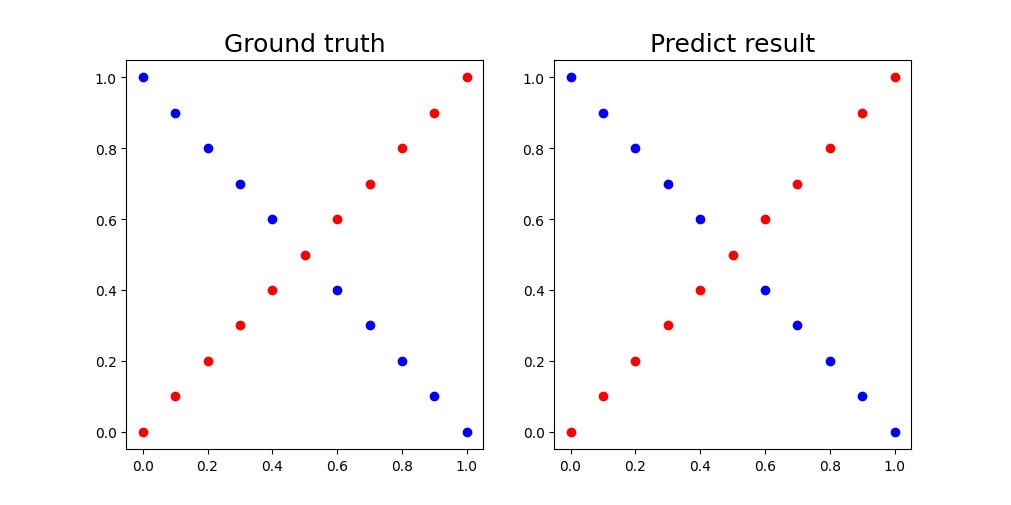
**Part 3: Results of my testing**

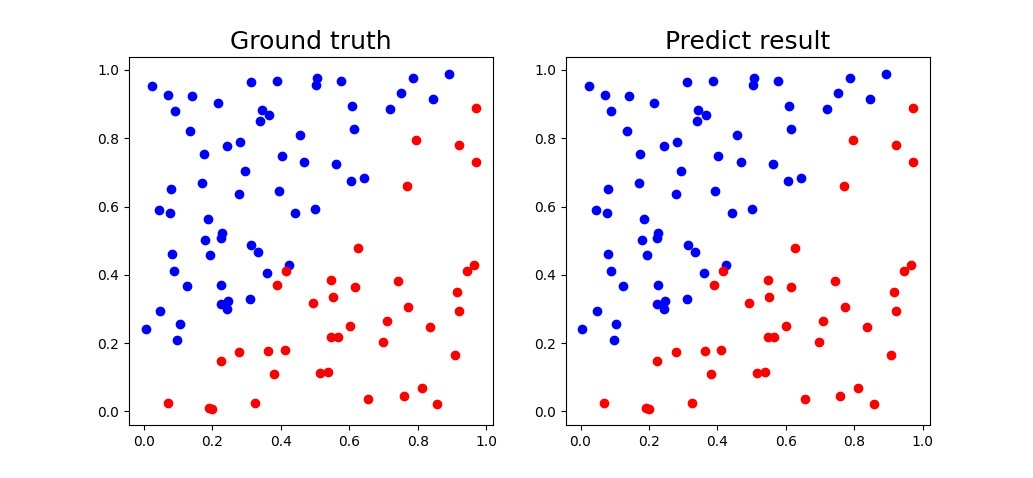
**Part 3-0: The way I run the NN for testing**

The following figure is how I implemented the NNs and run the testing, I will show how I drew the comparison figures and show the results in the following sections.

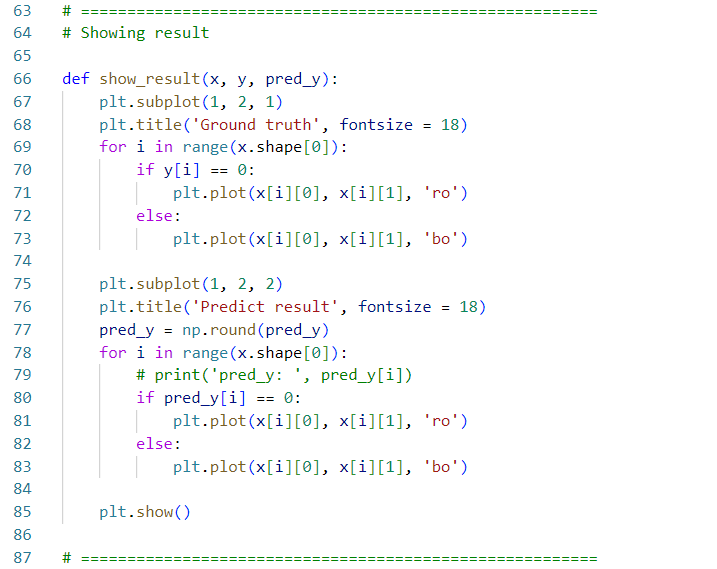


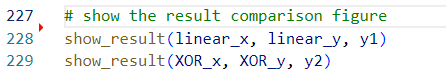
**Part 3-A: Screenshot and comparison figure**



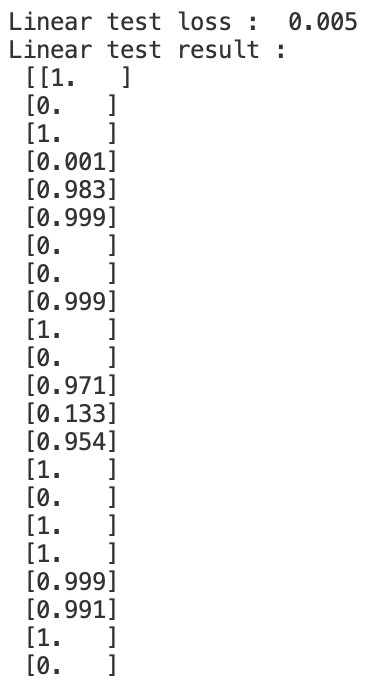
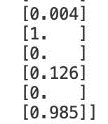
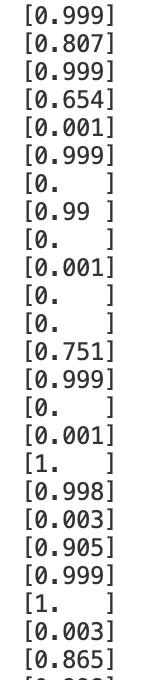
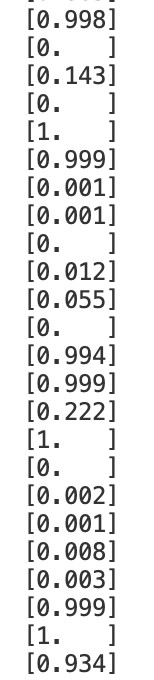
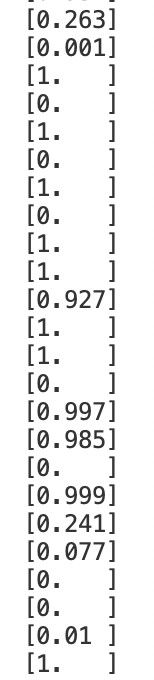
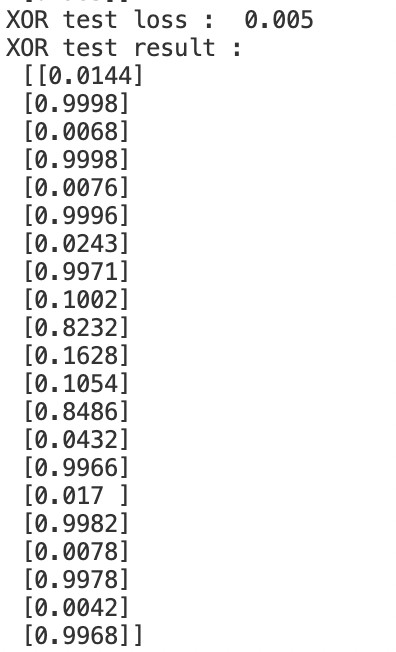


The following function in the figure is how I get the comparison plots.

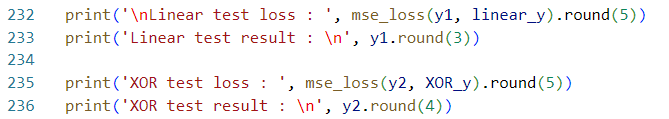




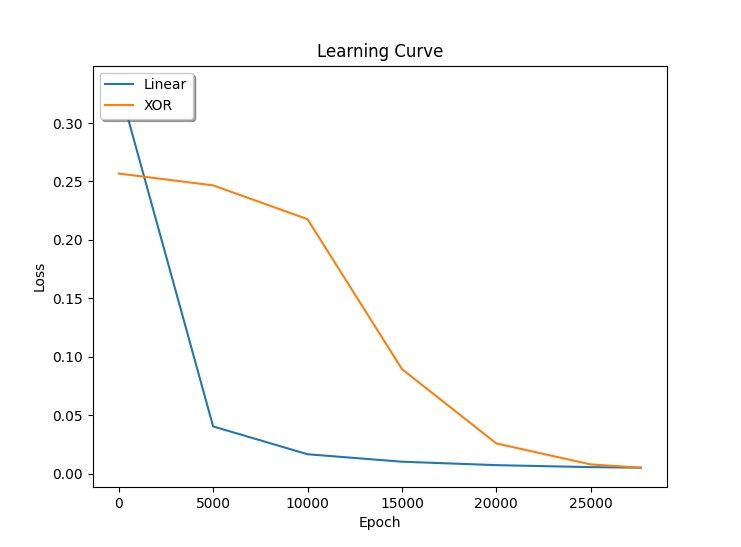
**Part 3-B: Show the accuracy of your prediction**

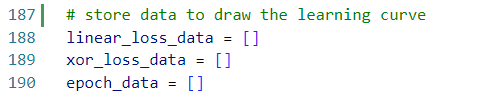
The following figure is the way I obtain and print the result.

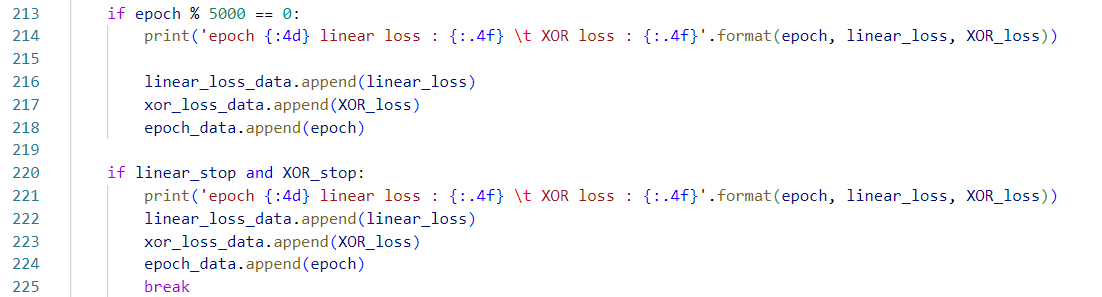


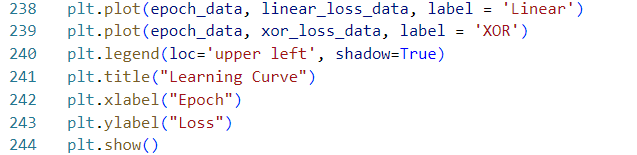
**Part 3-C: Learning curve (loss, epoch curve)**



The following figure is the way I obtain and plot the learning curve.







**Part 4: Discussion**

**Part 4-A: Try different learning rates**

In this section, I’ll try on different learning rates on the same dataset, and I’ll take record of the epochs needed for the neural networks to reach the target loss, which is 0.005, on two kinds of datasets. The following chart is the result of the experiments using different learning rates.

|  |  |  |
| --- | --- | --- |
| Learning Rate | Linear Data | XOR Data |
| 0.001 | 3,172,986 epochs | 2,896,658 epochs |
| 0.01 | 679,327 epochs | 359,669 epochs |
| 0.1 | 36,717 epochs | 30,658 epochs |
| 0.5 | 10,442 epochs | 5,910 epochs |
| 0.999 | 1,376 epochs | 3,394 epochs |

Through the experiments above, we can easily find the epochs required for the neural networks to reach the loss goal increases as we lower the learning rate, which proves that higher learning rate requires less training time in the same background conditions.

**Part 4-B: Try different numbers of hidden units**

In this part, I’ll also test the results of different numbers of hidden units in the same background condition, and see how the different neural networks preform in reaching the requiring loss, which is 0.005. The following chart is the result of the experiments using different hidden units.

Note | The learning rates in this part are all set to 0.1

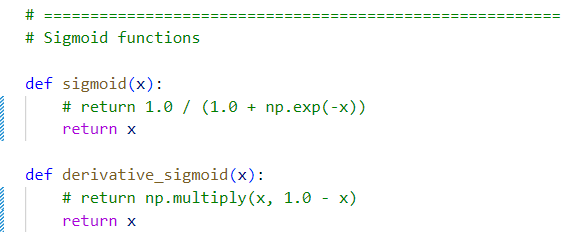
|  |  |  |
| --- | --- | --- |
| Hidden Units | Linear Data | XOR Data |
| 2 | 144,633 epochs | 1,000,000+ epochs |
| 4 | 36,717 epochs | 30,658 epochs |
| 8 | 24,257 epochs | 28,231 epochs |
| 16 | 14,090 epochs | 18,868 epochs |
| 32 | 12,682 epochs | 30,570 epochs |

From the result above, we can find the neural network starts to perform a bit strange when there are 2 or 32 hidden units in the hidden layers. In the case of this lab, the complexity of the dataset is actually not too high. Hence, we can find the layer with 16 hidden units performs the best among the experiments conducted. When the number of hidden units reached 32, the neural network started to act not so good, or sometimes even worse, on the datasets. In my opinion, 16 hidden units in a layer will be the best number in this lab.

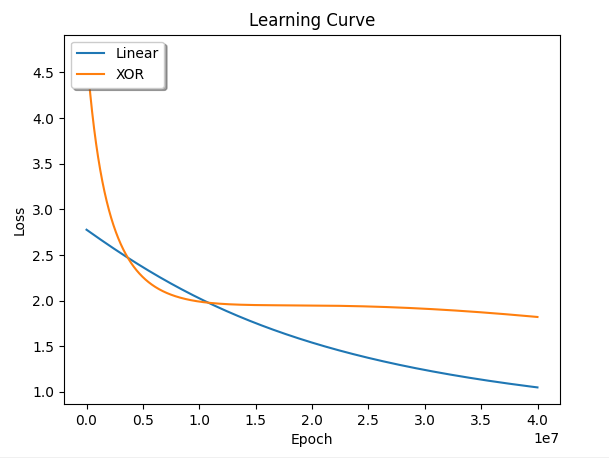
**Part 4-C: Try without activation functions**

If we took the activation function away, the weight and the bias will simple do linear transformation when updating. As mentioned in class, linear equation might be easier to solve, but it might also loose the capacity to solve problems with higher complexity. If the neural network got no activation functions in it, then it will become just a linear regression model. The main work activation functions do is to make the input capable to learn and perform tasks with higher complexity.

For this part, I changed the sigmoid functions by outputting the input directly, which is shown in the figure below.



After taking the activation functions away, I found the original learning rate make the model’s loss overflow. Hence, I started to lower the learning rate to avoid gradient exploration. After testing nine learning rates, without exploding during the process, I found myself able to get the proper result not until I lower to learning rate to “0.000000001”. However, the model still can reach the expected result, which if the loss with 0.005, even with 4 \* 107 epochs. The following figure shows the learning curve of the model.



Through this experiment, we can find the importance of activation functions if we want to work on more complicated cases. Without activation functions, it is nearly impossible for us to be able to perform complex tasks, which might cost us plenty of time without getting the expecting result.