**Lab1: back-propagation**

Author: 311553007應耀德

1. **Introduction**

**Lab Objective:**

In this lab, you will need to understand and implement simple neural networks with forwarding pass and backpropagation using two hidden layers. Notice that you can only use NumPy and the python standard libraries, any other frameworks (ex : TensorFlow、PyTorch) are not allowed in this lab.

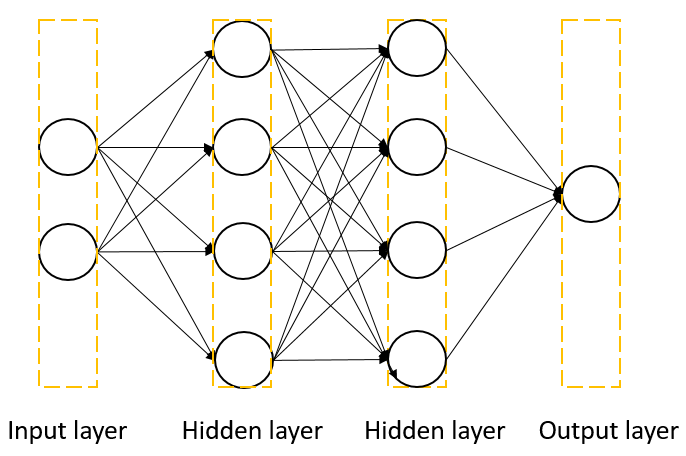


Figure 1. Two-layer neural network

**Requirements:**

1. Implement simple neural networks with two hidden layers.
2. You must use backpropagation in this neural network and can only use NumPy and other python standard libraries to implement.
3. Plot your comparison figure that show the predicted results and the ground-truth.

**Data:**

Use two types of data to test the neural network.

* Uniform distribution (blue points are labeled to 1, red points are labeled to 0)

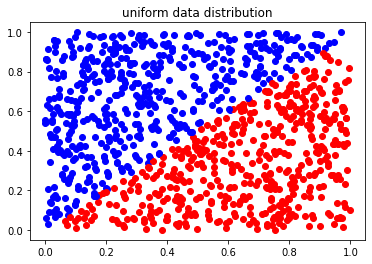


Figure 2. Data points generated from uniform distribution

* XOR (X shape) (blue points are labeled to 1, red points are labeled to 0)

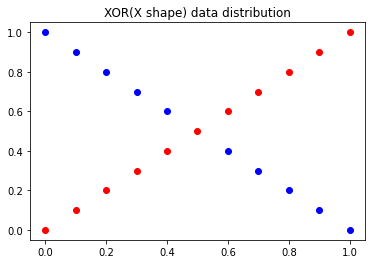


Figure 3. Data points generated from XOR

* Code for data generation

To make the data generation deterministic, specify the seed of the random generator and recreate the random generator for every generation.

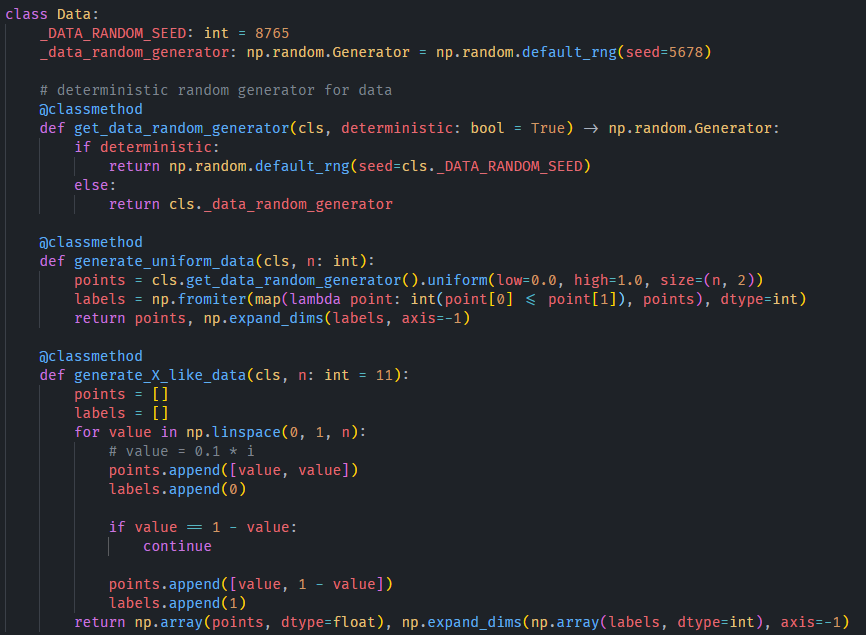


Figure 4. Code for data generation

**Data Setups:**

* Code for splitting data (output format is x\_train, x\_test, y\_train, y\_test)

一張含有 文字 的圖片

自動產生的描述

Figure 5. Code for splitting data

* Code for shuffling data (output format is x, y)

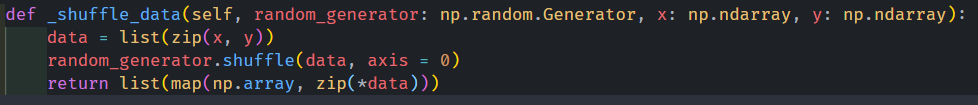


Figure 6. Code for shuffling data

**Data Preprocessing:**

* Uniform data

1. Generate 5000 data points for the training dataset.
   1. Split the training dataset into the training set and validation set by 0.8 and 0.2.
   2. Shuffle the training set before starting training for each epoch.

一張含有 文字 的圖片

自動產生的描述

Figure 7. Label distributions of the training and validation set

1. Generate 1000 data points for the testing dataset.

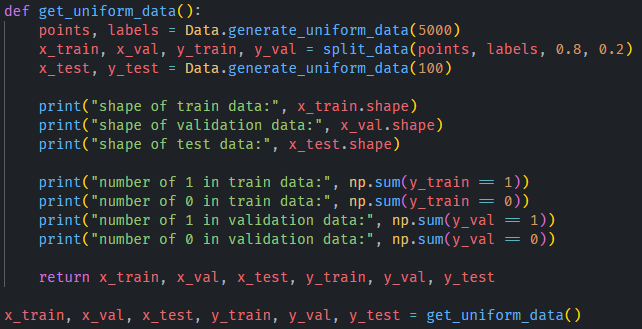


Figure 8. Code for preprocessing uniform data

* XOR data

1. Generate the same data for the training, validation, and testing dataset.
2. Shuffle the training set before starting training for each epoch.

一張含有 文字 的圖片

自動產生的描述

Figure 9. Label distributions

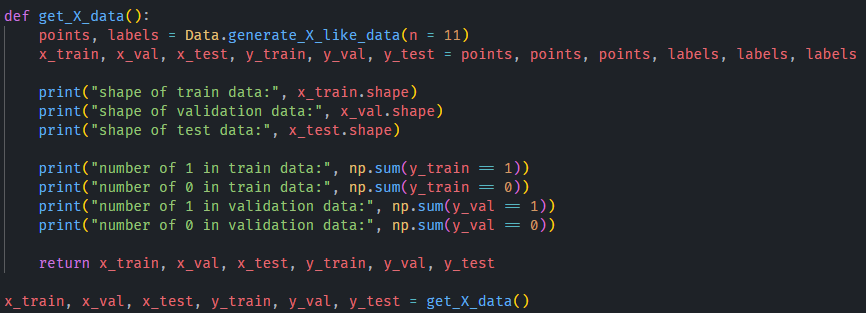


Figure 10. Code for preprocessing XOR data

**Flowchart for training a model:**

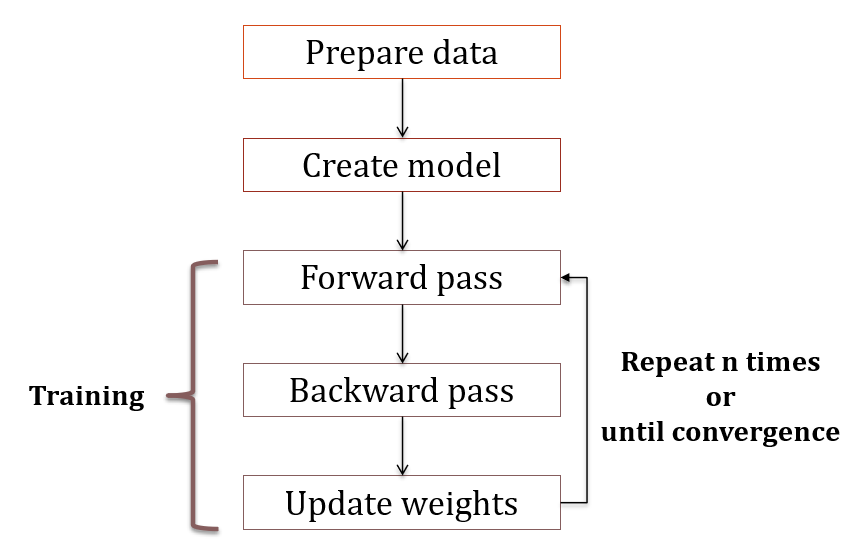


Figure 11. Flowchart for training a model

**Perceptron:**

A basic computation unit for neural network.

一張含有 文字 的圖片

自動產生的描述

Figure 12. Equation of linear transformation with bias

一張含有 文字, 時鐘, 黑色, 黑暗 的圖片

自動產生的描述

Figure 13. Details of calculating linear transformation

**Neural Network (Multilayer perceptron MLP):**

For each hidden layer and output layer, it contains at least one unit (perceptron).

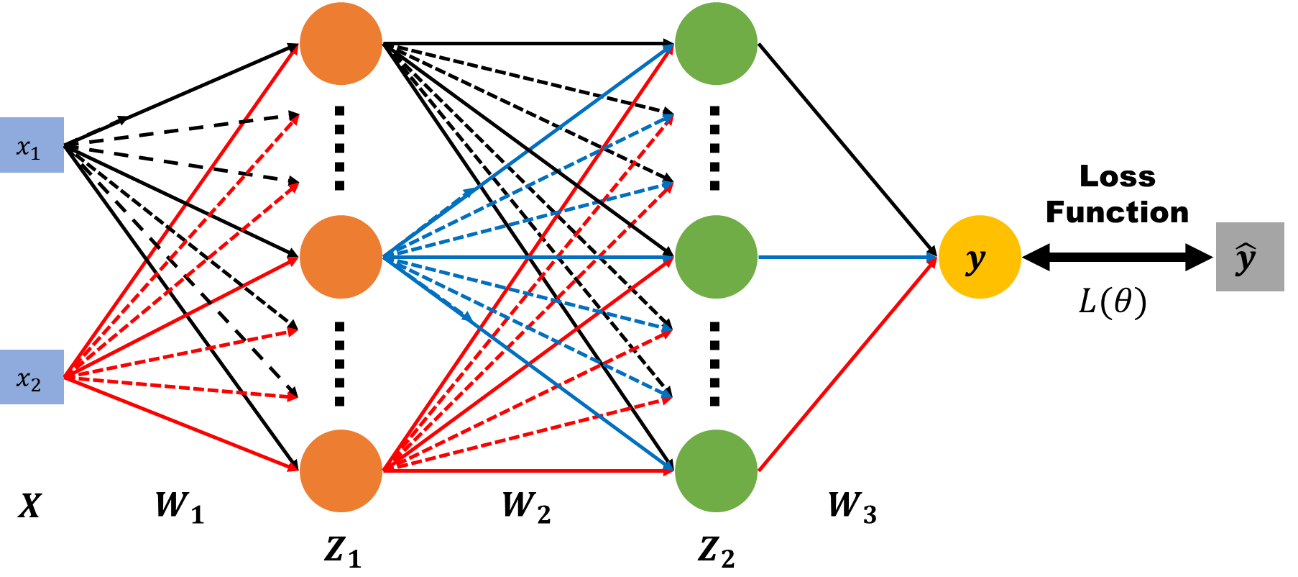
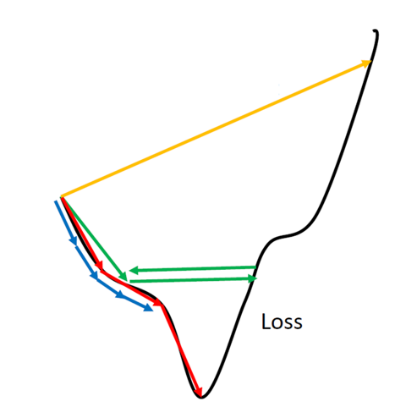


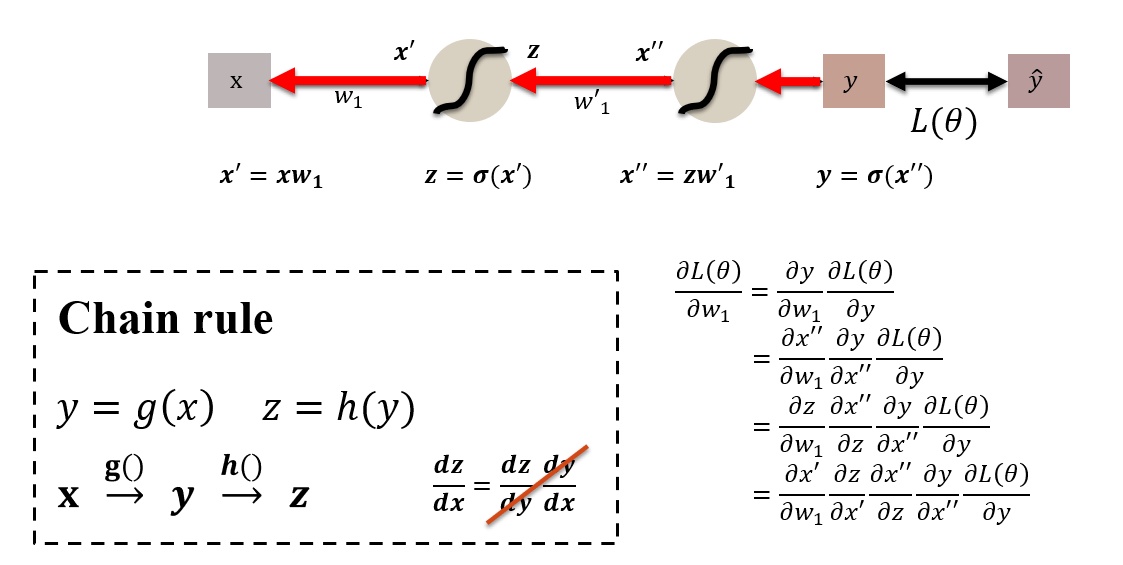
Figure 14. Architecture of Neural Network

(Sigmoid)

**Backpropagation:**

一張含有 文字 的圖片

自動產生的描述一張含有 文字 的圖片

自動產生的描述

**Activation functions:**

* Sigmoid

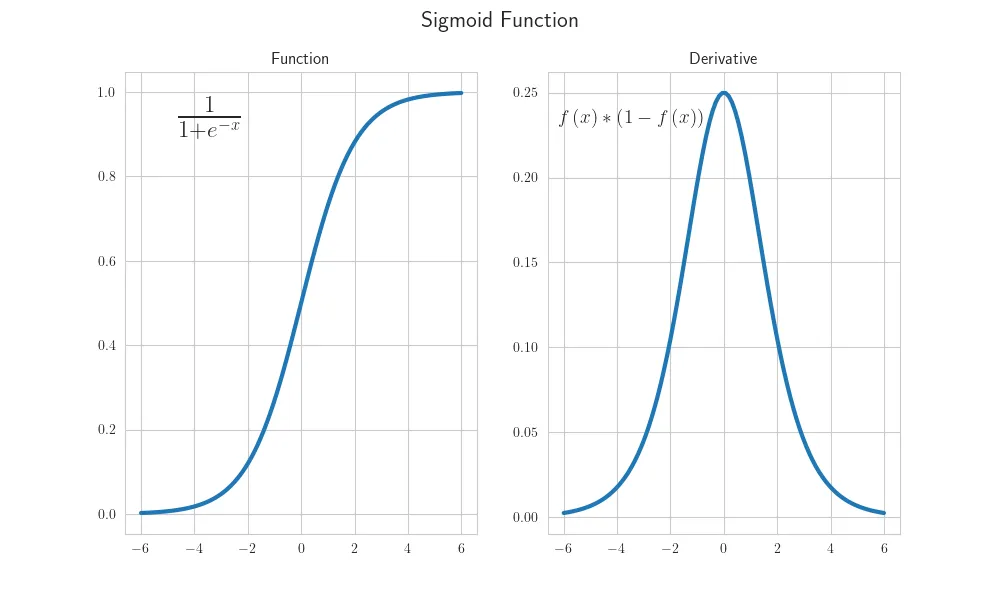
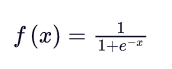
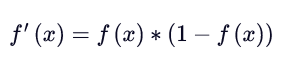


Figure 15. Visualization of Sigmoid function

* + The function of the forward pass



* + The function of the backward pass (derivative function)



* Tanh

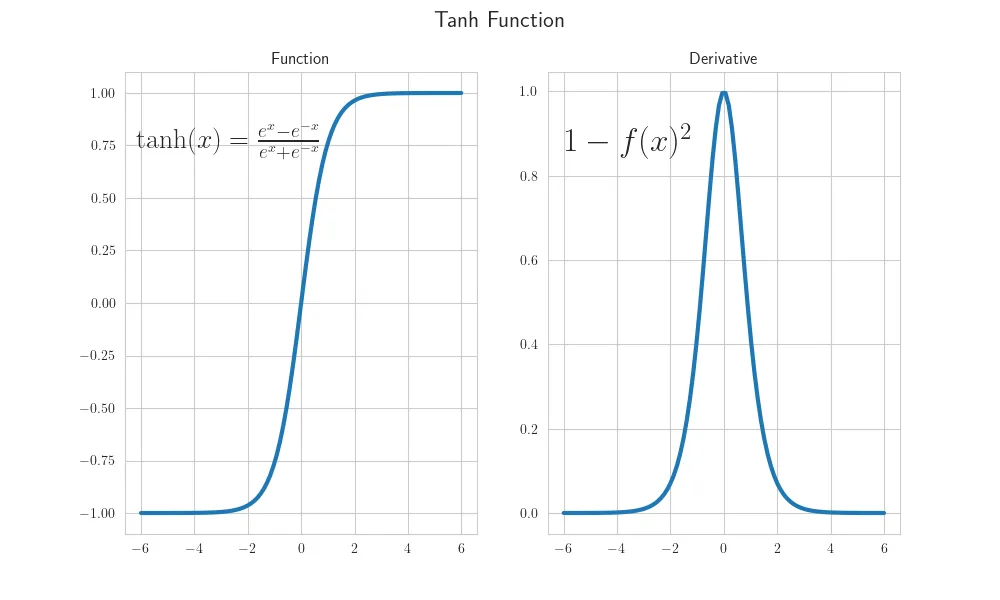


Figure 16. Visualization of Tanh function

* + The function of the forward pass

一張含有 文字 的圖片

自動產生的描述

* + The function of the backward pass (derivative function)



* ReLU

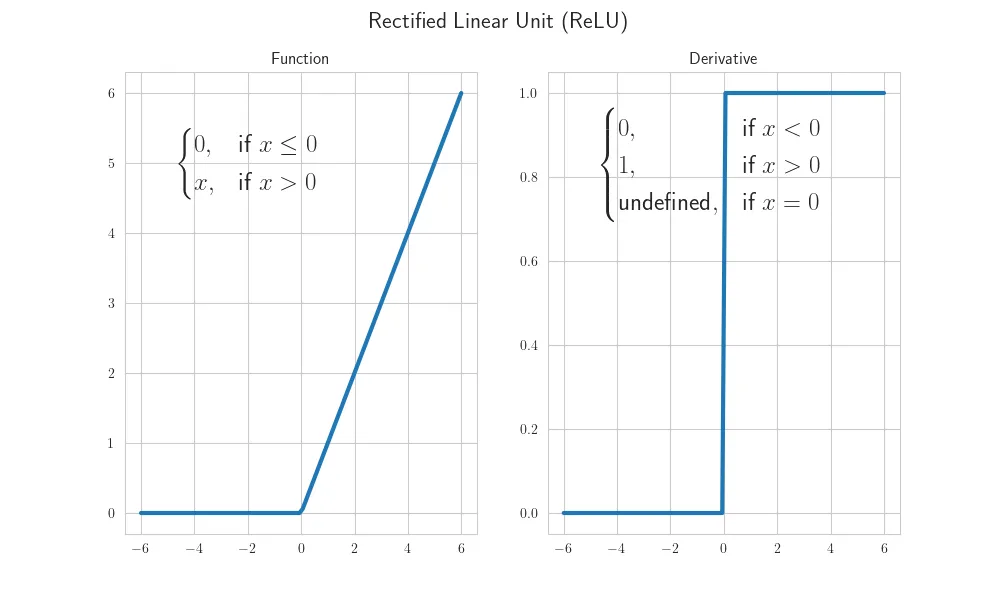
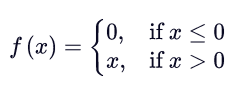


Figure 17. Visualization of ReLU function

* + The function of the forward pass





* + The function of the backward pass (derivative function)  
    Note: In this lab, if x = 0, I define the value 0 instead of undefined.

一張含有 文字 的圖片

自動產生的描述

* Leaky ReLU

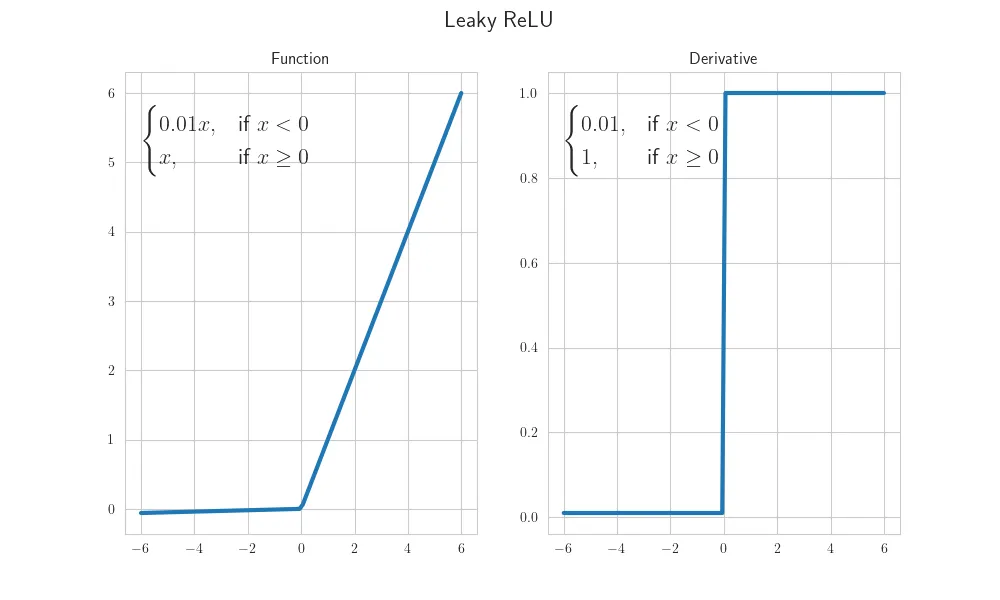


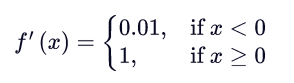
Figure 18. Visualization of Leaky ReLU function

* + The function of the forward pass

一張含有 文字 的圖片

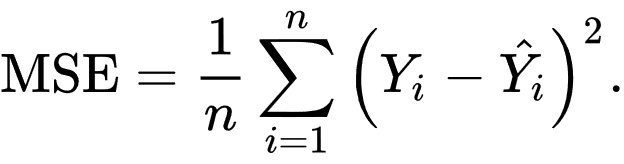
自動產生的描述

* + The function of the backward pass (derivative function)



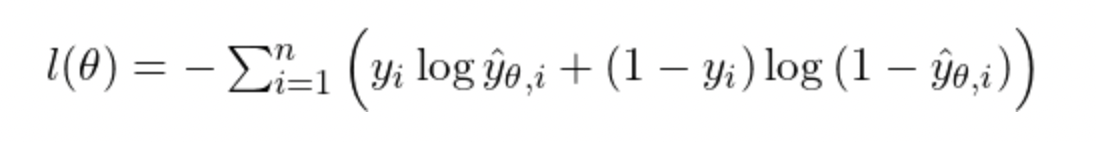
**Loss functions:**

* Mean Squared Error



Where

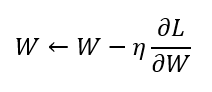
* Negative Log-Likelihood for Bernoulli Distribution



Where

**Optimizers:**

* Stochastic Gradient Descent (SGD)



Where

* Momentum

一張含有 文字 的圖片

自動產生的描述

Where

1. **Experiment setups**

**Shared settings:**

* **Optimizer – Stochastic Gradient Descent (SGD)**

一張含有 文字 的圖片

自動產生的描述

Figure 19. Implementation of stochastic gradient descent (SGD)

* **Loss function – Mean Squared Error**

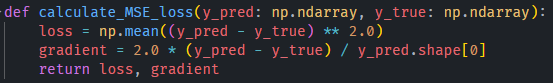


Figure 20. Implementation of mean squared error

* **Output unit – Sigmoid**

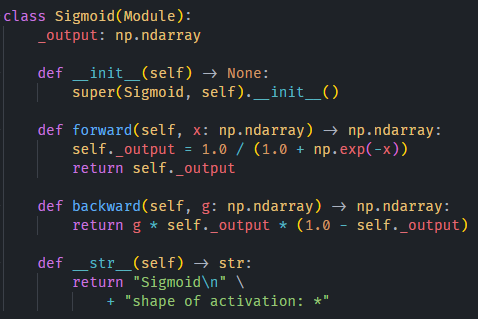


Figure 21. Implementation of sigmoid function

* **Dense layer (layers in neural network)**
  + forward: forward pass function
  + backward: backward pass function, calculate the gradients
  + update: step function, update parameters by optimizer

一張含有 文字 的圖片

自動產生的描述

一張含有 文字 的圖片

自動產生的描述

Figure 22. Implementation of dense layer

* **Backpropagation**

一張含有 文字 的圖片

自動產生的描述

Figure 23. Implementation of training process

1. Calculate the gradient of loss function, *g*
2. Propagate the gradient *g* to neural network (dense layers and activation functions)

一張含有 文字 的圖片

自動產生的描述

Figure 24. Propagate the gradient *g* to neural network

1. Calculate the gradients

一張含有 文字 的圖片

自動產生的描述

Figure 25. Calculate the gradients of dense layer

一張含有 文字 的圖片

自動產生的描述

Figure 26. Calculate the gradients of sigmoid function

1. Update parameters



Figure 27. Propagate the optimizer to neural network

一張含有 文字 的圖片

自動產生的描述

Figure 28. Update the parameters of dense layer

一張含有 文字 的圖片

自動產生的描述

Figure 29. Implementation of step function of SGD optimizer

**Uniform data:**

* **Neural network**
  + Input size: (\*, 2)
  + Output size: (\*, 1)
  + Initializer: normal distribution
  + Learning rate: 1.5
  + Hidden layers: (2 x 3) with bias, (3 x 2) with bias
  + Hidden units: ReLU
  + Output layer: with bias

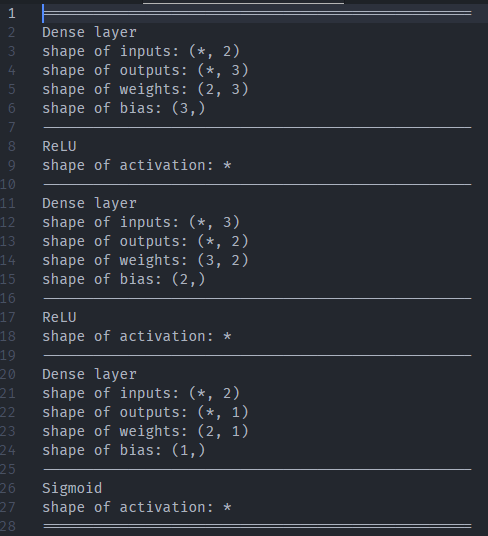


Figure 30. Architecture of neural network for uniform data

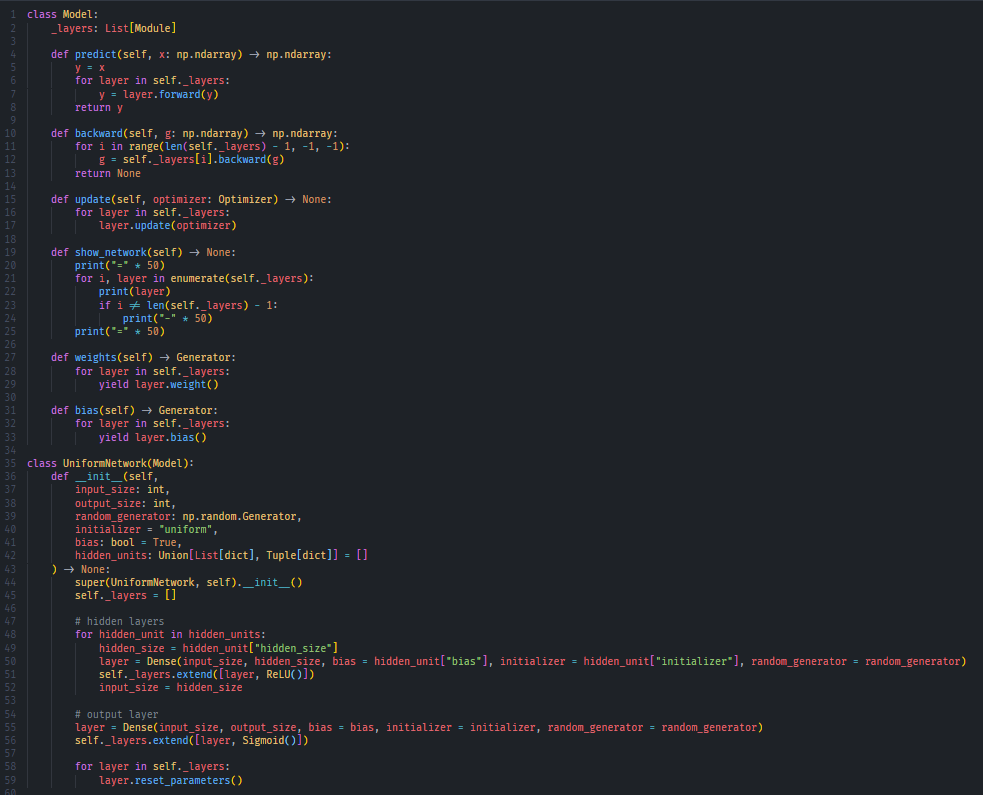


Figure 31. Implementation of neural network for uniform data

**XOR data:**

* **Neural network**
  + Input size: (\*, 2)
  + Output size: (\*, 1)
  + Initializer: normal distribution
  + Learning rate: 1.0
  + Hidden layers: (2 x 3) with bias, (3 x 2) with bias
  + Hidden units: ReLU
  + Output layer: with bias

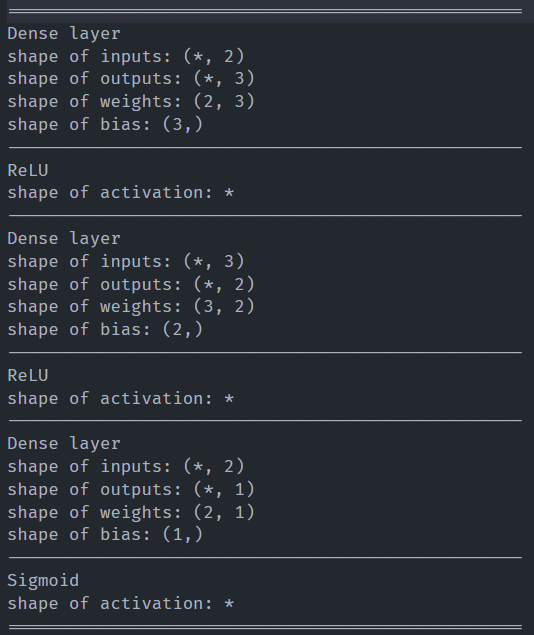


Figure 32. Architecture of neural network for XOR data

一張含有 文字 的圖片

自動產生的描述

Figure 33. Implementation of neural network for XOR data

1. **Results of your testing**

**Uniform data:**

* + **Learning curve (train loss, validation loss, epoch)**

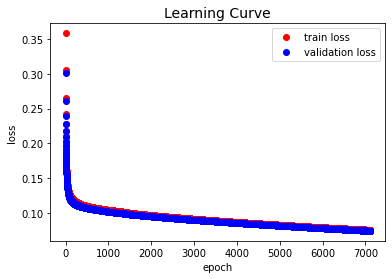


Figure 34. Learning curve of neural network for uniform data

* + **Accuracy (train accuracy, validation accuracy, epoch)**

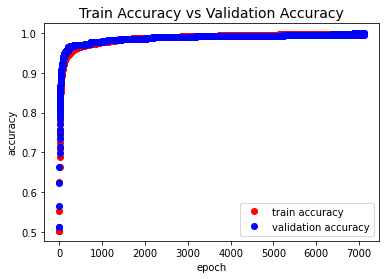


Figure 35. Accuracy comparison between training and validation

* + **Predict result:** accuracy 99.4%

一張含有 文字, 藍色, 靠近 的圖片

自動產生的描述

Figure 36. Predict results of neural network for uniform data

**XOR data:**

* + **Learning curve (train loss, validation loss, epoch)**

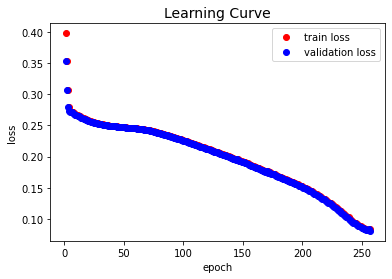


Figure 37. Learning curve of neural network for XOR data

* + **Accuracy (train accuracy, validation accuracy, epoch)**

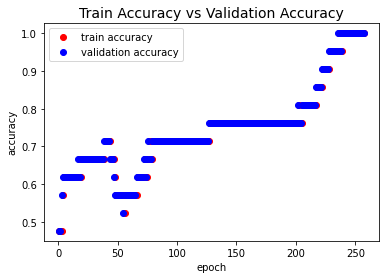


Figure 38. Accuracy comparison between training and validation

* + **Predict result:** accuracy 100%

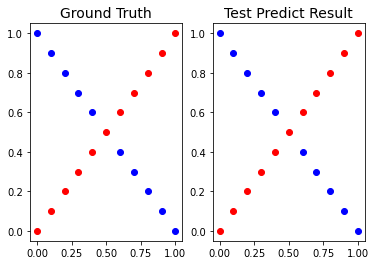
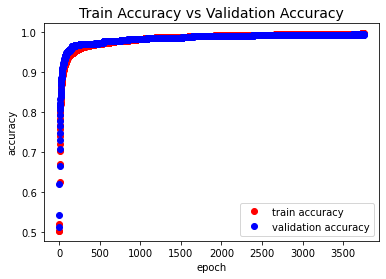


Figure 39. Predict results of neural network for XOR data

1. **Discussion**
   * **Try different learning rates:** useuniform data for this experiment

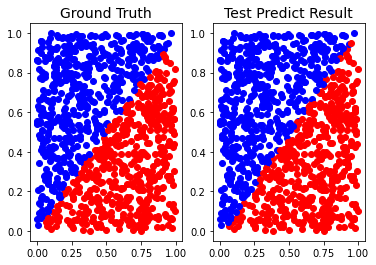
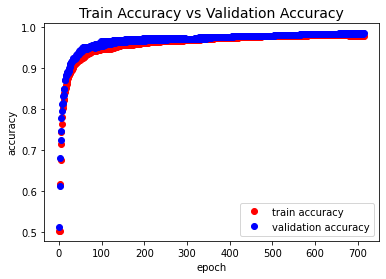
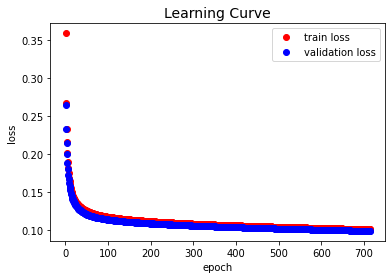
**Learning rate = 2.0:** the epochs is totally decreased by 50%; the accuracy is 99.1%

一張含有 文字 的圖片

自動產生的描述一張含有 文字, 藍色, 靠近 的圖片

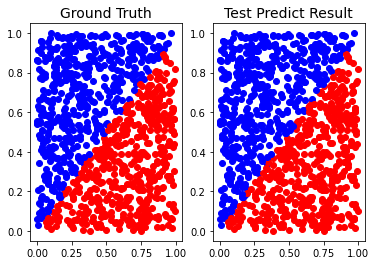
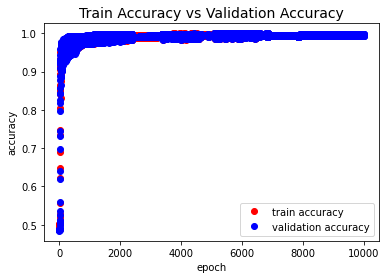
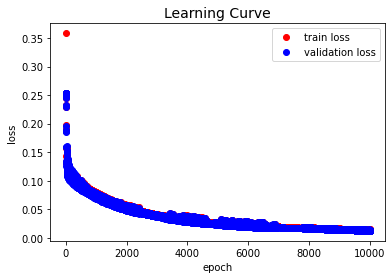
自動產生的描述

**Learning rate = 3.0:** the epochs is totally decreased by 20%; the accuracy is 97.9%



**Learning rate = 15.0:** the epochs is increased; but the accuracy is 99.4%

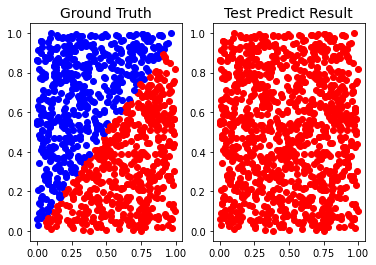
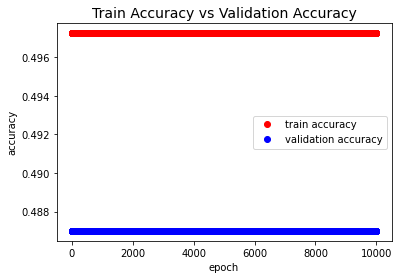
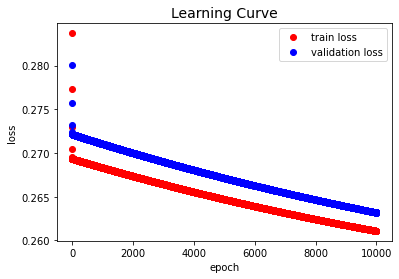
Compared with other learning rates, we can see the learning curve drops down dramatically and the accuracy goes up. The model still has good performance, but there are more and more glitches during the training and validation.



* + **Try different numbers of hidden units:** useuniform data for this experiment

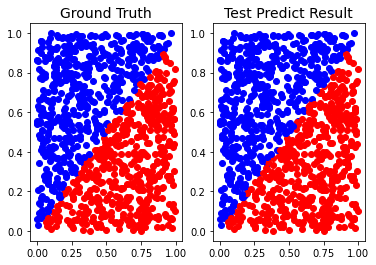
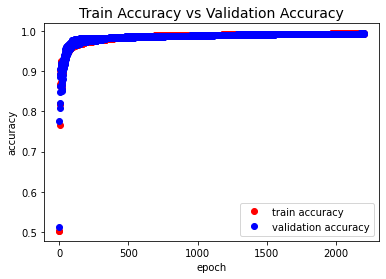
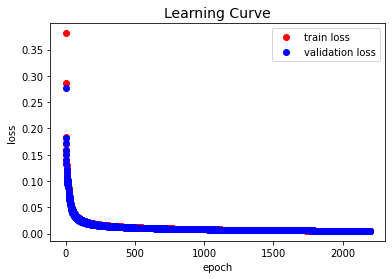
**If we change the size of the last hidden layer to 1, 3, 4:**

The model cannot predict correctly.



**If we change the size of the last hidden layer to 5, 6, 7… (above 5):**

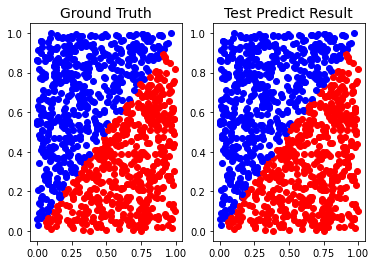
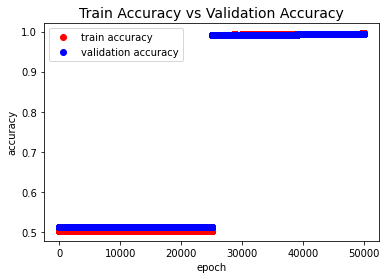
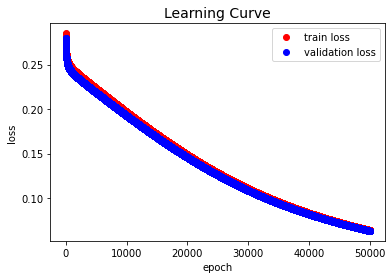
The model can predict normally and there is no overfitting. But the model is for ideal only (not realistic) because the data is from uniform distribution, and the decision line of labels is diagonal.



**If we remove one hidden layer and change the size of another hidden layer to 1:**

The model can also predict normally! That means we can use only one line to fit the curve.

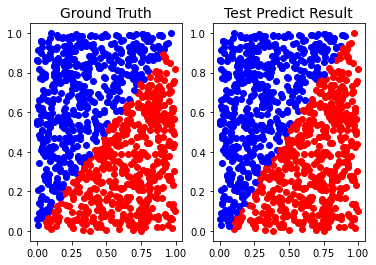
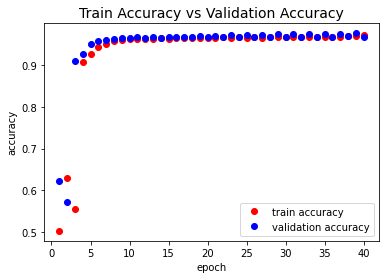
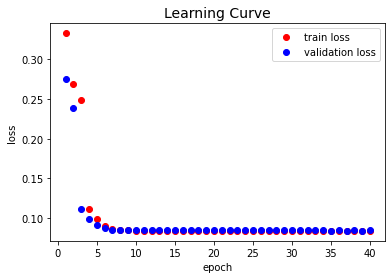
(In fact, applying deep learning is redundant. Perceptron can do it well.)



* + **Try without activation functions:**

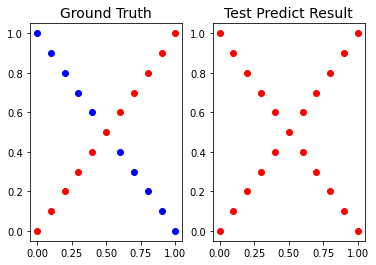
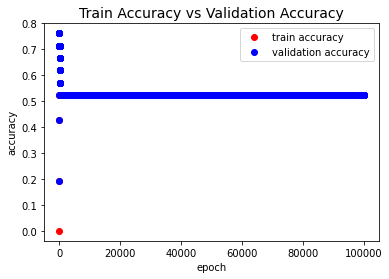
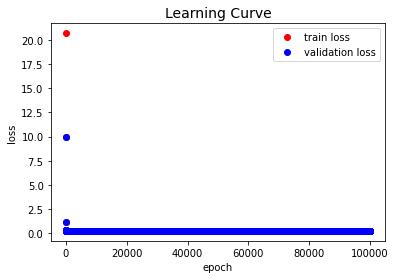
**Uniform data:**

The linear equation can easily fit the curve. The model doesn’t need a nonlinear equation.



**But what if the data is XOR?**

Gradient vanishing! The model cannot only use a linear equation to fit the curve.



1. **Extra**

**Momentum optimizer**

一張含有 文字 的圖片

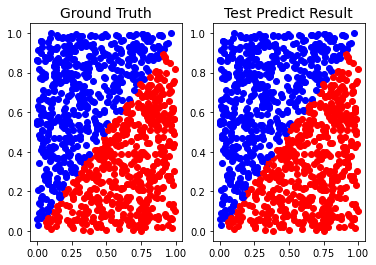
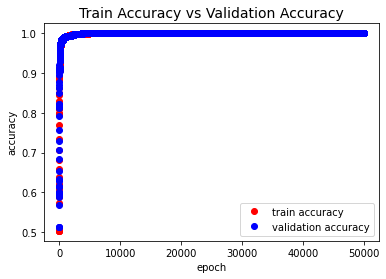
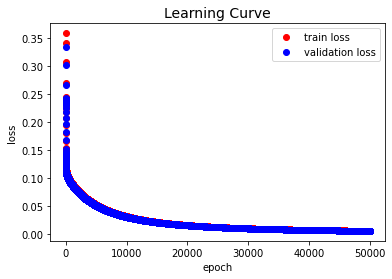
自動產生的描述

* + Uniform data: use the same settings in experiment setups

Learning rate: 0.5

Momentum: 0.9

Accuracy: 100% (very smooth but slow convergence)

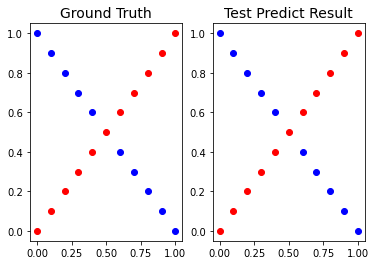
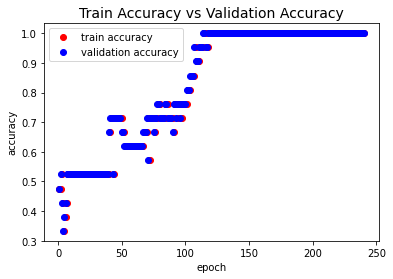
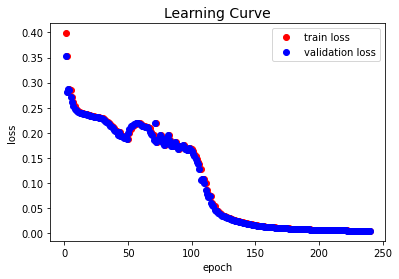


* + XOR data: use the same settings in experiment setups

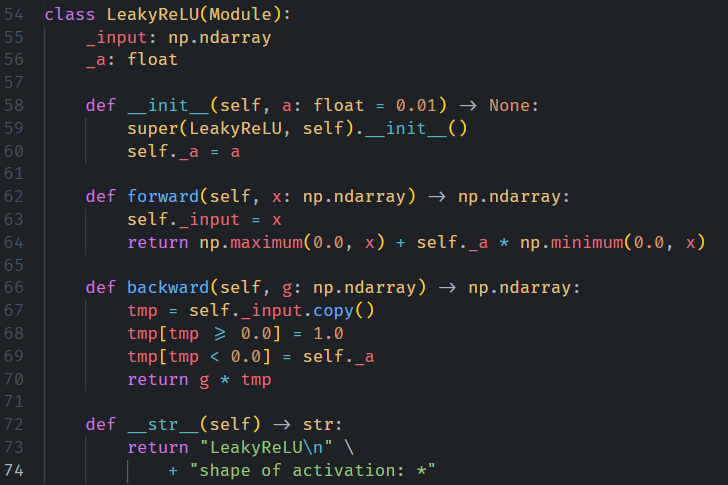
Learning rate: 1.0

Momentum: 0.9

Accuracy: 100%

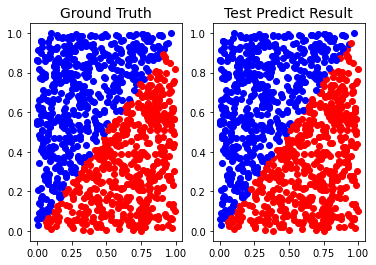
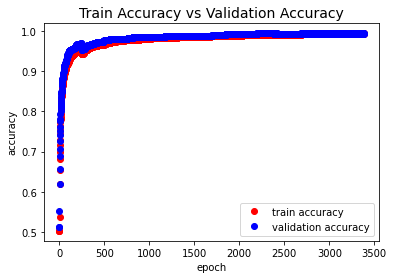
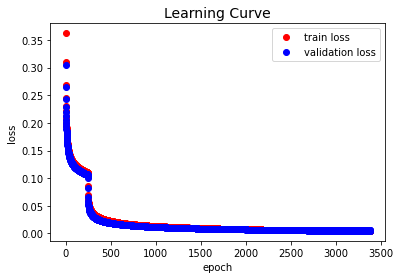


**Leaky ReLU activation function as hidden units**



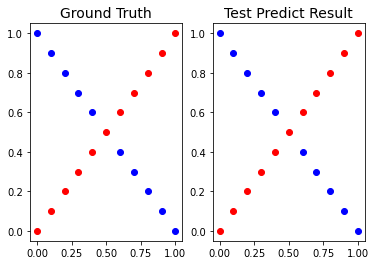
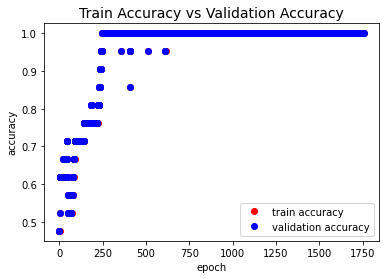
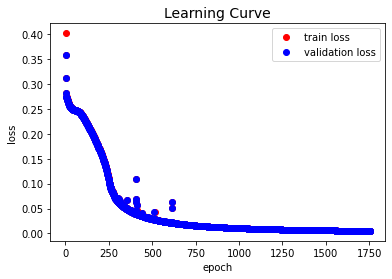
* + Uniform data: use the same settings in experiment setups

Accuracy: 99% (fast convergence)

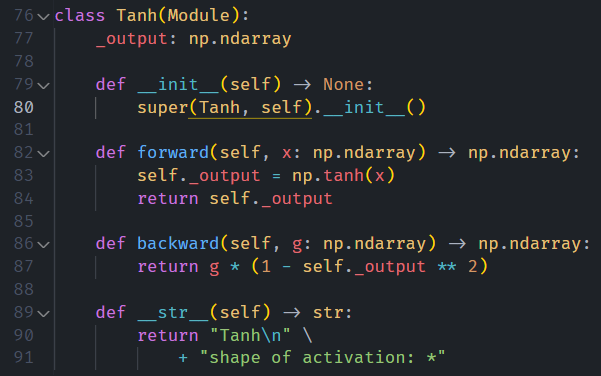


* + XOR data: use the same settings in experiment setups

Accuracy: 100% (fast convergence)



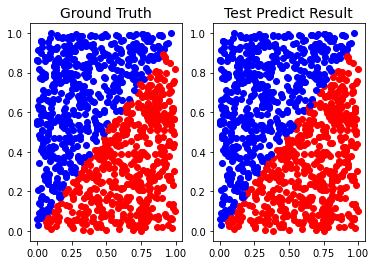
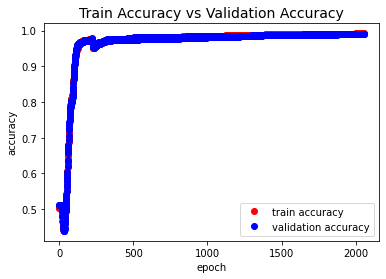
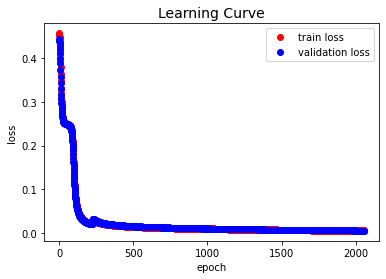
**Tanh activation function as hidden units**



* + Uniform data: use the same settings in experiment setups

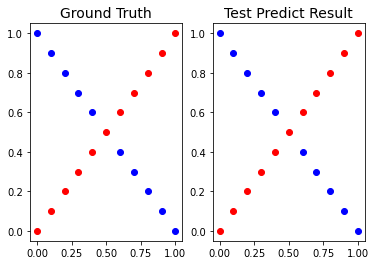
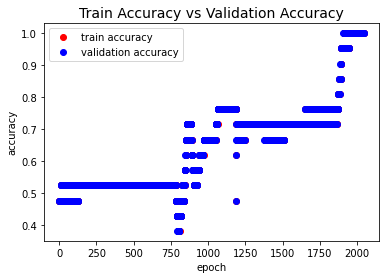
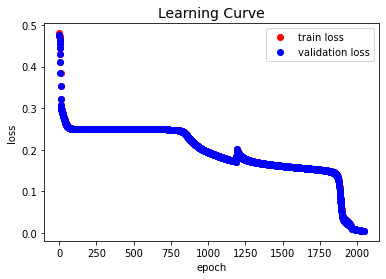
The generalization error between training and validation is almost disappeared.

Accuracy: 99.1%



* + XOR data: use the same settings in experiment setups

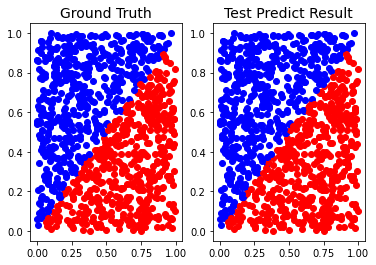
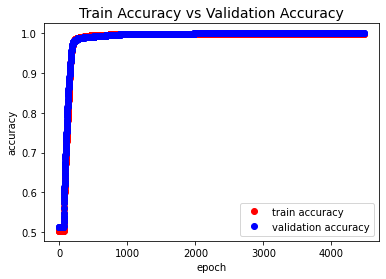
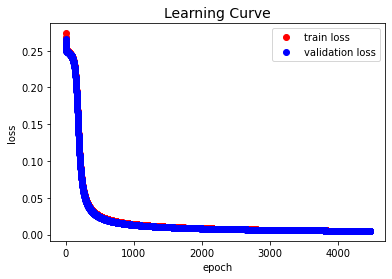
Accuracy: 100%



**Sigmoid activation function as hidden units**

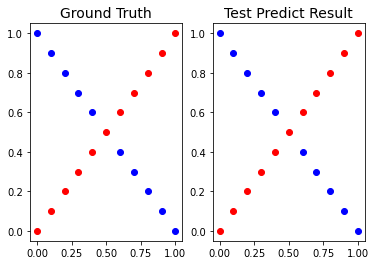
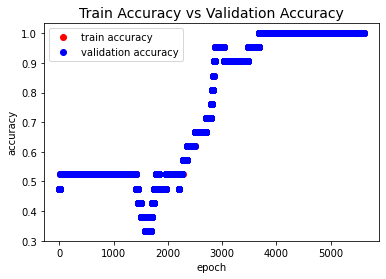
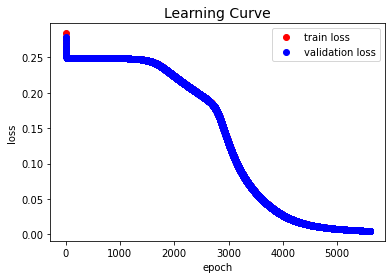
* + Uniform data: use the same settings in experiment setups

Accuracy: 99%



* + XOR data: use the same settings in experiment setups

Accuracy: 100%



1. **Reference**
   1. Activation functions: <https://ml-explained.com/blog/activation-functions-explained>
   2. PyTorch document: <https://pytorch.org/docs/stable/index.html>
   3. Optimizers: <https://medium.com/%E9%9B%9E%E9%9B%9E%E8%88%87%E5%85%94%E5%85%94%E7%9A%84%E5%B7%A5%E7%A8%8B%E4%B8%96%E7%95%8C/%E6%A9%9F%E5%99%A8%E5%AD%B8%E7%BF%92ml-note-sgd-momentum-adagrad-adam-optimizer-f20568c968db>
   4. Lab 1 Word & PowerPoint