PA 1: Simple Neural Network

20222421 GSAI SimJaeYoon

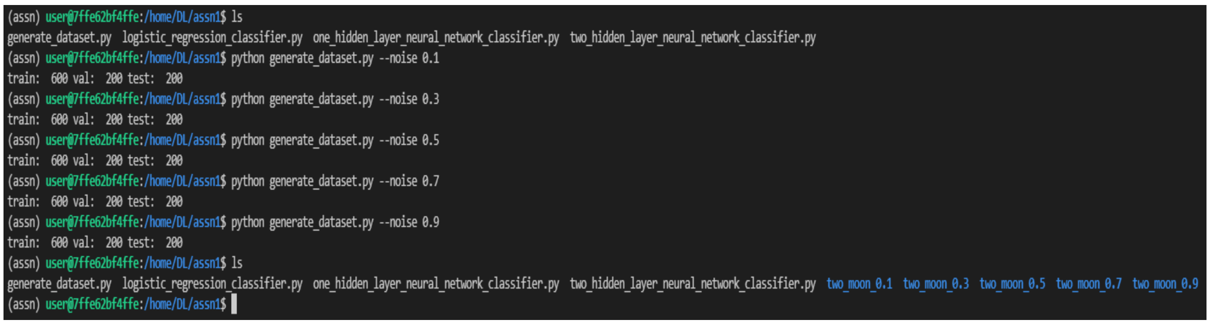
0. Overview

This project is about two class classification of "Moon dataset" using logistic regression classifier and one or two hidden layer neural network. The important thing is that we can use only numpy package which is python's library that supports easy processing of matrices or large multi-dimensional arrays in general. Although there so many useful deep learning frameworks like pytorch and tensorflow, this project is required to use only numpy package.

1. Generate Two Moon Datasets

Scikit-learn library is simple and efficient tools for predictive data analysis. It can be accessible to everybody and reusable in various contexts. We can use scikit-learn library for machine learning analysis and it provides lots of interesting mathematics tools. Also, it provides lots of data for prediction. Among them, we want to use "moon dataset" for binary classification by using make_moons function which makes two interleaving half circles.

1.1. To make moon dataset, use the attached moon dataset generator program (generator_dataset.py)

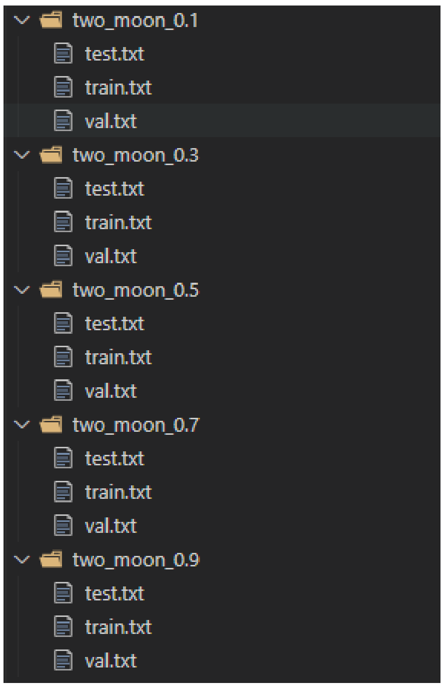


```
(assn) user@7ffe62bf4ffe:/home/DL/assn1$ ls
generate_dataset.py  logistic_regression_classifier.py  one_hidden_layer_neural_network_classifier.py  two_hidden_layer_neural_network_classifier.py
(assn) user@7ffe62bf4ffe:/home/DL/assn1$ python generate_dataset.py --noise 0.1
train:  600 val:  200 test:  200
(assn) user@7ffe62bf4ffe:/home/DL/assn1$ python generate_dataset.py --noise 0.3
train:  600 val:  200 test:  200
(assn) user@7ffe62bf4ffe:/home/DL/assn1$ python generate_dataset.py --noise 0.5
train:  600 val:  200 test:  200
(assn) user@7ffe62bf4ffe:/home/DL/assn1$ python generate_dataset.py --noise 0.7
train:  600 val:  200 test:  200
(assn) user@7ffe62bf4ffe:/home/DL/assn1$ python generate_dataset.py --noise 0.9
train:  600 val:  200 test:  200
(assn) user@7ffe62bf4ffe:/home/DL/assn1$ ls
generate_dataset.py  logistic_regression_classifier.py  one_hidden_layer_neural_network_classifier.py  two_hidden_layer_neural_network_classifier.py  two_moon_0.1  two_moon_0.3  two_moon_0.5  two_moon_0.7  two_moon_0.9
(assn) user@7ffe62bf4ffe:/home/DL/assn1$
```

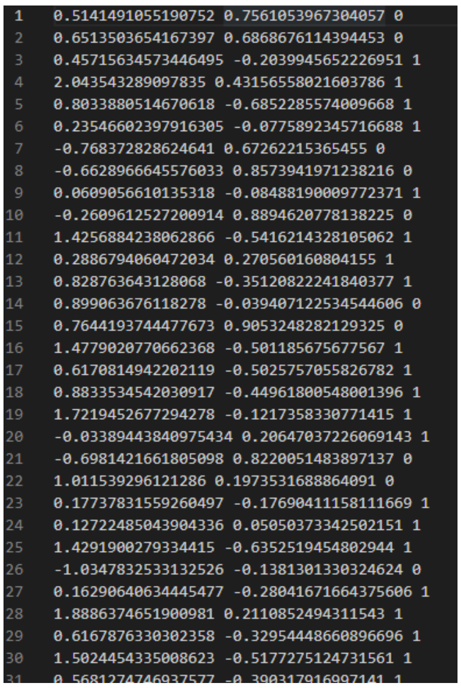
The scikit-learn library provides multiple datasets, of which the moon dataset was created using the given generator_dataset.py file.

1.2. Change the noise parameter from 0.1 to 0.9 by 0.2. The generated dataset will be stored as train.txt, val.txt, test.txt within the folder ./two_moon_noise.

Gaussian noise having a noise parameter value as a standard deviation may be added as data. So, we made a dataset by changing the noise parameter by 0.2. At this time, folders are created for each noise parameter, where train.txt, val.txt, and test.txt files are stored. And each txt file contains data with different labels of 0 and 1.

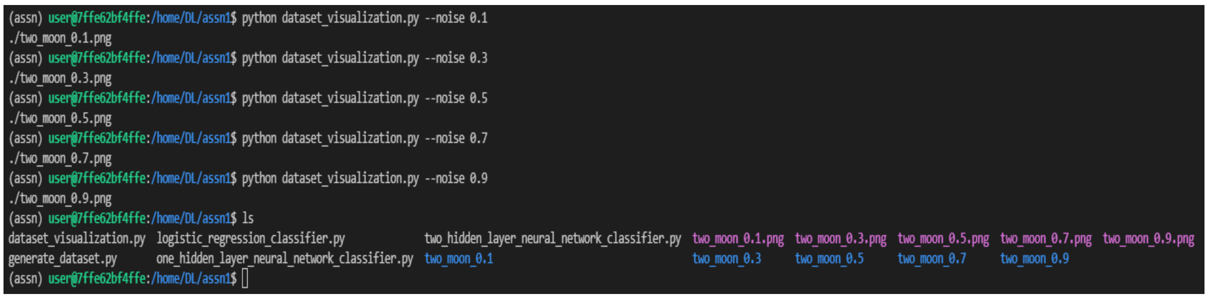


```
two_moon_0.1
    test.txt
    train.txt
    val.txt
two_moon_0.3
    test.txt
    train.txt
    val.txt
two_moon_0.5
    test.txt
    train.txt
    val.txt
two_moon_0.7
    test.txt
    train.txt
    val.txt
two_moon_0.9
    test.txt
    train.txt
    val.txt
```



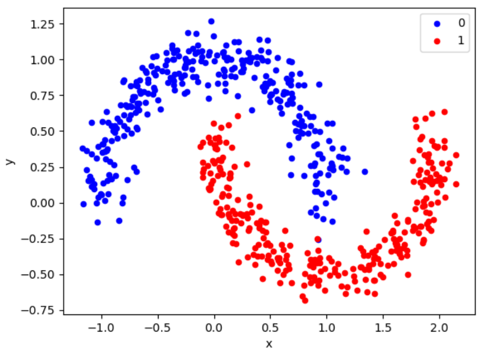
```
0.5141491055190752 0.7561053967304057 0
0.6513503654167397 0.6868676114394453 0
0.45715634573446495 -0.2039945652226951 1
2.043543289097835 0.43156558021603786 1
0.8033880514670618 -0.6852285574009668 1
0.23546602397916305 -0.0775892345716688 1
-0.768372828624641 0.67262215365455 0
-0.6628966645576033 0.8573941971238216 0
0.0609056610135318 -0.08488190009772371 1
-0.2609612527200914 0.8894620778138225 0
1.4256884238062866 -0.5416214328105062 1
0.2886794060472034 0.270560160804155 1
0.828763643128068 -0.35120822241840377 1
0.899063676118278 -0.039407122534544606 0
0.7644193744477673 0.9053248282129325 0
1.4779020770662368 -0.501185675677567 1
0.6170814942202119 -0.5025757055826782 1
0.8833534542030917 -0.44961800548001396 1
1.7219452677294278 -0.1217358330771415 1
-0.03389443840975434 0.20647037226069143 1
-0.6981421661805098 0.8220051483897137 0
1.011539296121286 0.1973531688864091 0
0.17737831559260497 -0.17690411158111669 1
0.12722485043904336 0.05050373342502151 1
1.4291900279334415 -0.6352519454802944 1
-1.0347832533132526 -0.1381301330324624 0
0.16290640634445477 -0.28041671664375606 1
1.8886374651900981 0.2110852494311543 1
0.6167876330302358 -0.32954448660896696 1
1.5024454335008623 -0.5177275124731561 1
0.5681274746937577 -0.390317916997141 1
```

In order to visually see how the distribution of data is made, a file called data_visualization.py was created to create a visualization file as follows.

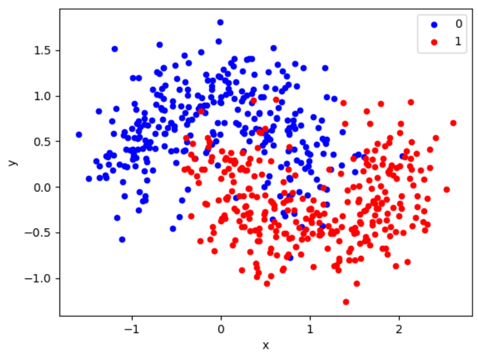


```
(assn) user@7ffe62bf4ffe:/home/DL/assn1$ python dataset_visualization.py --noise 0.1
./two_moon_0.1.png
(assn) user@7ffe62bf4ffe:/home/DL/assn1$ python dataset_visualization.py --noise 0.3
./two_moon_0.3.png
(assn) user@7ffe62bf4ffe:/home/DL/assn1$ python dataset_visualization.py --noise 0.5
./two_moon_0.5.png
(assn) user@7ffe62bf4ffe:/home/DL/assn1$ python dataset_visualization.py --noise 0.7
./two_moon_0.7.png
(assn) user@7ffe62bf4ffe:/home/DL/assn1$ python dataset_visualization.py --noise 0.9
./two_moon_0.9.png
(assn) user@7ffe62bf4ffe:/home/DL/assn1$ ls
dataset_visualization.py  logistic_regression_classifier.py            two_hidden_layer_neural_network_classifier.py  two_moon_0.1.png  two_moon_0.3.png  two_moon_0.5.png  two_moon_0.7.png  two_moon_0.9.png
generate_dataset.py       one_hidden_layer_neural_network_classifier.py  two_moon_0.1                                   two_moon_0.3      two_moon_0.5      two_moon_0.7      two_moon_0.9
(assn) user@7ffe62bf4ffe:/home/DL/assn1$ 
```

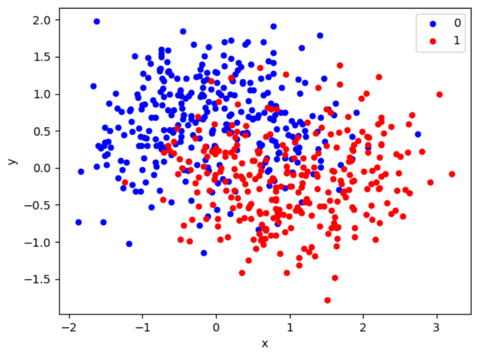
Looking at the distribution of data, as the value of noise increases, it is more difficult to distinguish because the data belonging to different labels overlap.



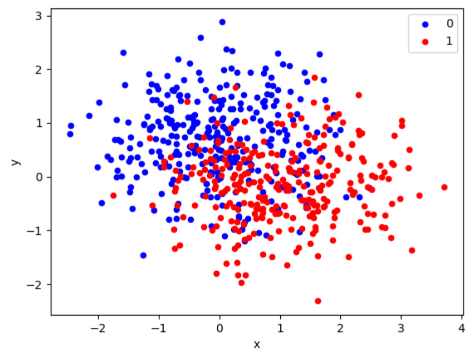
Noise Parameter = 0.1



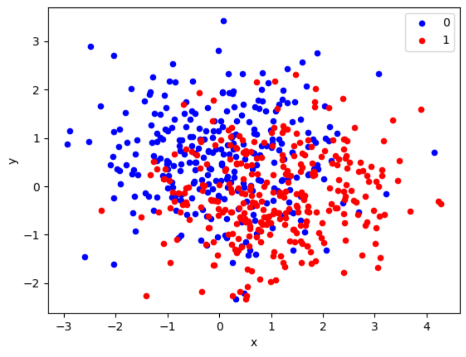
Noise Parameter = 0.3



Noise Parameter = 0.5



Noise Parameter = 0.7



Noise Parameter = 0.9

2. Logistic Regression Classifier

Logistic regression is a classification technique borrowed by machine learning from the field of statics. Logistic regression is a statistical method for analyzing a dataset in which there are one or more independent variables that determine an outcome. The intention behind using logistic regression is to find the best fitting model to describe the relationship between the dependent and the independent variable. This project is required to build logistic regression classifier and evaluate the classifier's accuracy for each dataset which was made.

2.1. Implement logistic regression classifier and evaluate the classifier's accuracy for each dataset.

First of all, we created a data loader to use the moon data set that I had previously created.

```
# Data loader from moon datasets
class Data_Loader:
    def __init__(self, noise_parameter=0.3):
        self.noise = noise_parameter

    def read_data(self):
        load_position = './' +'two_moon_' + str(self.noise) + '/'
        train_data = np.genfromtxt(load_position + 'train.txt', dtype = float)
        validation_data = np.genfromtxt(load_position + 'val.txt', dtype = float)
        test_data = np.genfromtxt(load_position + 'test.txt', dtype = float)
        return train_data, validation_data, test_data
```

After that, we created a class called logical regression classifier and defined the activation function and various functions inside to implement the training method.

```
# Logistic regression classifier
class Logistic_Regression_Classifier:
    def __init__(self, num_samples, num_features, learning_rate, epoch):
        self.weight = None
        self.bias = None
        self.num_samples = num_samples # 600 or 200
        self.num_features = num_features # 3
        self.lr = learning_rate # 0.1
        self.epoch = epoch # 1000

    # Activation function(Sigmoid)
    def activation_function_sigmoid(self, Z):
        A = 1 / (1 + np.exp(-Z))
        return A

    # Activation function(ReLU)
    def activation_function_relu(self, Z):
        A = np.maximum(0, Z)
        return A

    # Loss function
    def loss_function_cross_entropy(self, predict_y, real_y):
        L = -(1 / self.num_samples) * np.sum(real_y * np.log(predict_y) + (1 - real_y) * np.log(1-
predict_y))
        return L

    # Parameter(w,b) initialization
    def parameter_initialization(self):
        self.weight = np.random.randn(self.num_features, 1) # (2, 1)
        self.bias = np.random.randn(1, 1) # (1, 1)
```

The following is the process of training through forward and backpropagation.

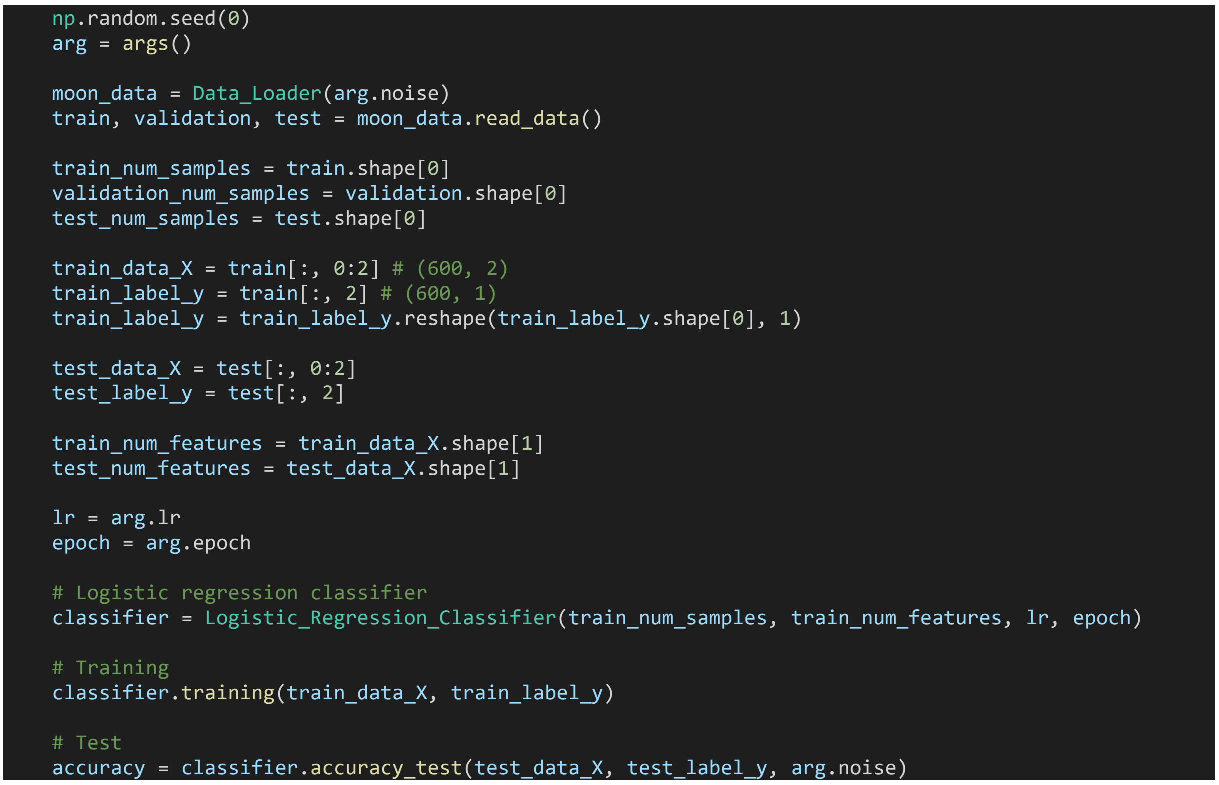
```
# Training : forward propagation
     def forward_propagation_training(self, X):
        Z = np.dot(X, self.weight) + self.bias
        A = self.activation_function_sigmoid(Z)
        return A

    # Training : back propagation
    def back_propagation_training(self, X, predict_y, real_y):
        dA = (-1) * np.divide(real_y, predict_y) + np.divide(1 - real_y, 1 - predict_y)
        dZ = predict_y - real_y
        dw = (1 / self.num_samples) * np.dot(X.T, dZ)
        db = (1 / self.num_samples) * np.sum(dZ)

        # Parameter update
        self.weight = self.weight - self.lr * dw
        self.bias = self.bias - self.lr * db

    # Training : forward propagation + back propagation
    def training(self, X, real_y):
        # Parameter intialization
        self.parameter_initialization()

        for i in range(self.epoch):
            predict_y = self.forward_propagation_training(X)
            self.back_propagation_training(X, predict_y, real_y)
```

The following is the process of measuring the actual accuracy using test data set.

```
    # Test : foward propagation and classification
    def forward_propagation_test(self, X):
        Z = np.dot(X, self.weight) + self.bias
        A = self.activation_function_sigmoid(Z)

        for i in range(A.shape[0]):
            if A[i][0] > 0.5:
                A[i][0] = 1
            else:
                A[i][0] = 0

        return A

    # Test : accuracy
    def accuracy_test(self, X, real_y, noise):
        predict_y = self.forward_propagation_test(X)
        num_test_samples = len(predict_y)

        correct = 0
        for i in range(num_test_samples):
            if(predict_y[i] == real_y[i]):
                correct = correct + 1
        accuracy = correct / num_test_samples * 100

        print('Accuracy of logistic regression classifier with ' + str(noise) + ' noise parameter is '
+ str(accuracy) + '%' + '!')
```

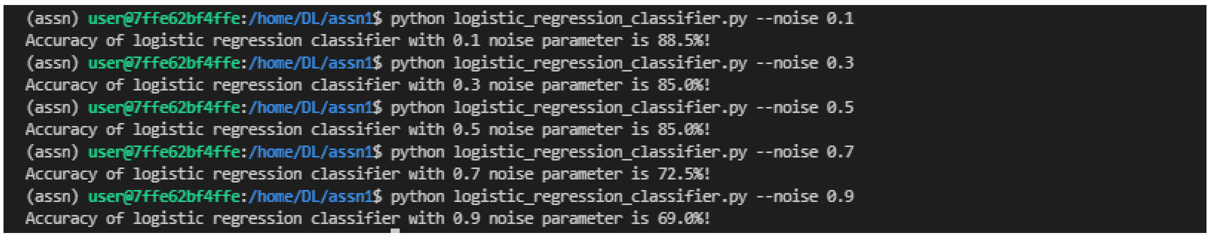
The following shows the entire process of performing the regression task. In the process of doing this task, the hyperparameter and the like were tuned using validation data set, and the same was done in the same way.

```
def args():
    parser = argparse.ArgumentParser()
    parser.add_argument('--noise', type=float, default=0.3)
    parser.add_argument('--lr', type=float, default=0.1)
    parser.add_argument('--epoch', type=float, default=1000)
    return parser.parse_args()

if __name__ == "__main__":
```



```python
    np.random.seed(0)
    arg = args()

    moon_data = Data_Loader(arg.noise)
    train, validation, test = moon_data.read_data()

    train_num_samples = train.shape[0]
    validation_num_samples = validation.shape[0]
    test_num_samples = test.shape[0]

    train_data_X = train[:, 0:2] # (600, 2)
    train_label_y = train[:, 2] # (600, 1)
    train_label_y = train_label_y.reshape(train_label_y.shape[0], 1)

    test_data_X = test[:, 0:2]
    test_label_y = test[:, 2]

    train_num_features = train_data_X.shape[1]
    test_num_features = test_data_X.shape[1]

    lr = arg.lr
    epoch = arg.epoch

    # Logistic regression classifier
    classifier = Logistic_Regression_Classifier(train_num_samples, train_num_features, lr, epoch)

    # Training
    classifier.training(train_data_X, train_label_y)

    # Test
    accuracy = classifier.accuracy_test(test_data_X, test_label_y, arg.noise)
```

Therefore, I summarized the accuracy obtained through the logistic regression task as follows.



```
(assn) user@7ffe62bf4ffe:/home/DL/assn1$ python logistic_regression_classifier.py --noise 0.1
Accuracy of logistic regression classifier with 0.1 noise parameter is 88.5%!
(assn) user@7ffe62bf4ffe:/home/DL/assn1$ python logistic_regression_classifier.py --noise 0.3
Accuracy of logistic regression classifier with 0.3 noise parameter is 85.0%!
(assn) user@7ffe62bf4ffe:/home/DL/assn1$ python logistic_regression_classifier.py --noise 0.5
Accuracy of logistic regression classifier with 0.5 noise parameter is 85.0%!
(assn) user@7ffe62bf4ffe:/home/DL/assn1$ python logistic_regression_classifier.py --noise 0.7
Accuracy of logistic regression classifier with 0.7 noise parameter is 72.5%!
(assn) user@7ffe62bf4ffe:/home/DL/assn1$ python logistic_regression_classifier.py --noise 0.9
Accuracy of logistic regression classifier with 0.9 noise parameter is 69.0%!
```

Noise Parameter	**0.1**	**0.3**	**0.5**	**0.7**	**0.9**
Accuracy	88.5	85.0	85.0	72.5	69.0

2.2. Analyze the classifier's accuracy with respect to the noise value.

It can be seen above that the distribution of data varies slightly depending on the noise parameter value. By changing the noise value of the same logistic regression classifier, we trained on train dataset and checked how accurately the classification task was conducted for the test dataset. When visualizing the distribution of data, it can be seen that the higher the noise value, the more difficult it is to classify. Therefore, it was expected that the accuracy of the classifier and the noise value would be inversely proportional. When checking the results, it was confirmed that the performance of the logistic regression classifier gradually decreased as the noise value increased. This seems to have difficulty in accurately solving the irregularity of the noise value because the logistic regression classifier simply performs a simple calculation process in the forward propagation process.

3. One Hidden Layer Neural Network Classifier

In the existing logistic regression classifier, only input and output layer existed. This time, we can design a classifier by adding one hidden layer and several hidden units between the input layer and the output layer.

3.1. Implement one hidden layer NN classifier with different number of hidden units(=1, 3, 5, 7, 9) and evaluate the one hidden layer NN classifier's accuracy for each dataset.

The overall structure is the same as the task above, but weight, bias, and intermediate calculation processes have been added because the hidden layer must be added between the input layer and the output layer.

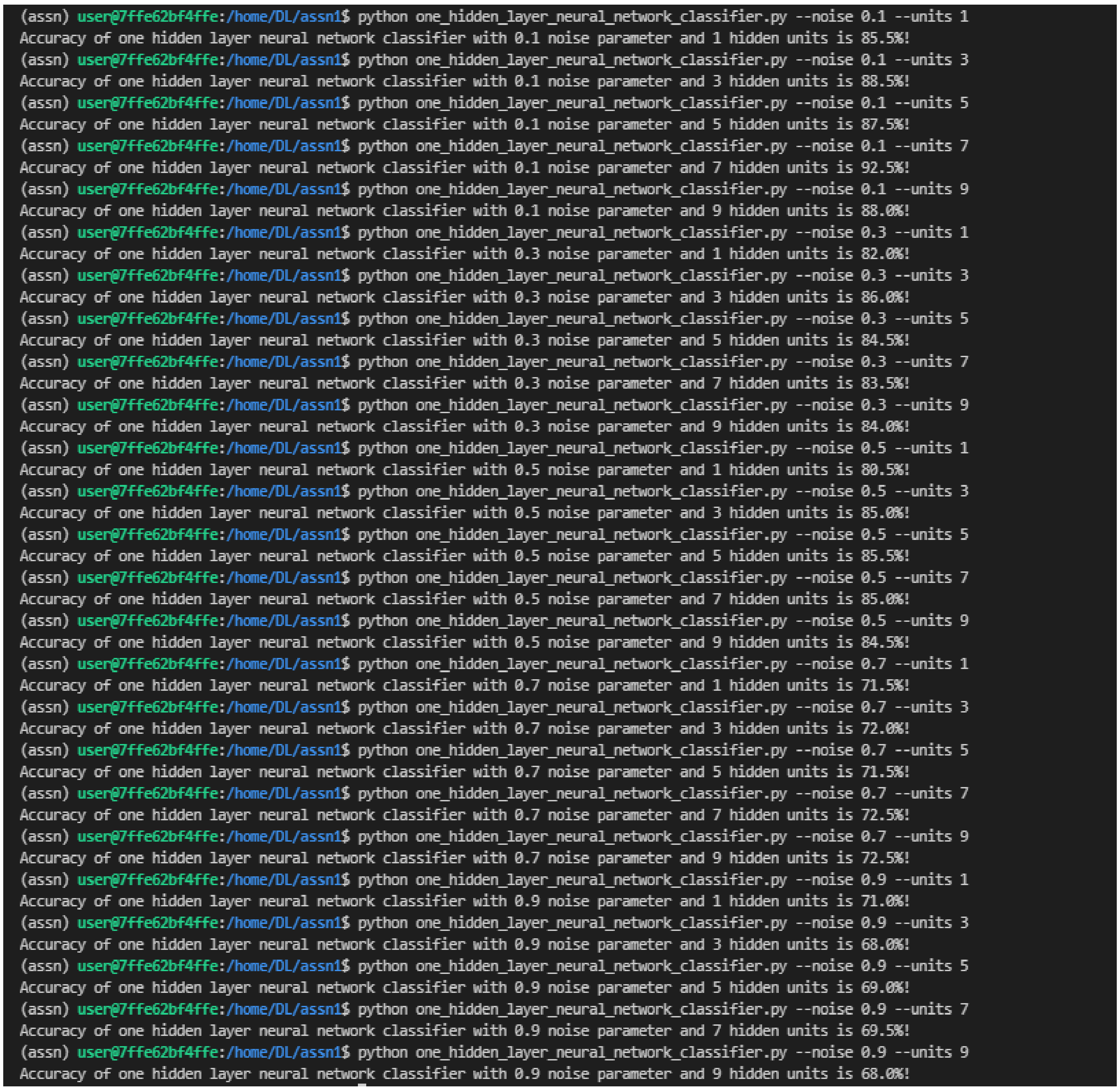
```
# Parameter(w,b) initialization
    def parameter_initialization(self):
        self.weight1 = np.random.randn(self.num_features, self.num_units) # (2, 1/3/5/7/9)
        self.bias1 = np.random.randn(1, 1) # (1, 1)
        self.weight2 = np.random.randn(self.num_units, 1) # (1/3/5/7/9, 1)
        self.bias2 = np.random.randn(1, 1) # (1, 1)
```

The training process has also added a calculation process for hidden layer.

```
    # Training : forward propagation
    def forward_propagation_training(self, X):
        Z1 = np.dot(X, self.weight1) + self.bias1
        A1 = self.activation_function_relu(Z1)
        Z2 = np.dot(A1, self.weight2) + self.bias2
        A2 = self.activation_function_sigmoid(Z2)

        self.Z1 = Z1
        self.A1 = A1
        self.Z2 = Z2
        self.A2 = A2

        return A2

    # Training : back propagation
    def back_propagation_training(self, X, predict_y, real_y):
        dA2 = (-1) * np.divide(real_y, predict_y) + np.divide(1 - real_y, 1 - predict_y)
        dZ2 = dA2 * 1 / (1 + np.exp(-self.Z2)) * (1 - 1 / (1 + np.exp(-self.Z2)))
        dw2 = (1 / self.num_samples) * np.dot(self.A1.T, dZ2)
        db2 = (1 / self.num_samples) * np.sum(dZ2)

        dA1 = np.dot(dZ2, self.weight2.T)
        dZ1 = dA1 * 1 / (1 + np.exp(-self.Z1)) * (1 - 1 / (1 + np.exp(-self.Z1)))
        dw1 = (1 / self.num_samples) * np.dot(X.T, dZ1)
        db1 = (1 / self.num_samples) * np.sum(dZ1)

        # Parameter update
        self.weight1 = self.weight1 - self.lr * dw1
        self.bias1 = self.bias1 - self.lr * db1
        self.weight2 = self.weight2 - self.lr * dw2
        self.bias2 = self.bias2 - self.lr * db2

    # Training : forward propagation + back propagation
    def training(self, X, real_y):
        # Parameter intialization
        self.parameter_initialization()

        for i in range(self.epoch):
            predict_y = self.forward_propagation_training(X)
            self.back_propagation_training(X, predict_y, real_y)
```

The test process is the same.

```
    # Test : foward propagation and classification
    def forward_propagation_test(self, X):
        Z1 = np.dot(X, self.weight1) + self.bias1
        A1 = self.activation_function_sigmoid(Z1)
        Z2 = np.dot(A1, self.weight2) + self.bias2
        A2 = self.activation_function_sigmoid(Z2)

        for i in range(A2.shape[0]):
            if A2[i][0] > 0.5:
                A2[i][0] = 1
            else:
                A2[i][0] = 0

        return A2

    # Test : accuracy
    def accuracy_test(self, X, real_y, noise, num_units):
        predict_y = self.forward_propagation_test(X)
        num_test_samples = len(predict_y)

        correct = 0
        for i in range(num_test_samples):
            if(predict_y[i] == real_y[i]):
                correct = correct + 1
        accuracy = correct / num_test_samples * 100

        print('Accuracy of one hidden layer neural network classifier with ' + str(noise) + ' noise 
parameter and ' + str(num_units) + ' hidden units is ' + str(accuracy) + '%' + '!')
```



```
(assn) user@7ffe62bf4ffe:/home/DL/assn1$ python one_hidden_layer_neural_network_classifier.py --noise 0.1 --units 1
Accuracy of one hidden layer neural network classifier with 0.1 noise parameter and 1 hidden units is 85.5%!
(assn) user@7ffe62bf4ffe:/home/DL/assn1$ python one_hidden_layer_neural_network_classifier.py --noise 0.1 --units 3
Accuracy of one hidden layer neural network classifier with 0.1 noise parameter and 3 hidden units is 88.5%!
(assn) user@7ffe62bf4ffe:/home/DL/assn1$ python one_hidden_layer_neural_network_classifier.py --noise 0.1 --units 5
Accuracy of one hidden layer neural network classifier with 0.1 noise parameter and 5 hidden units is 87.5%!
(assn) user@7ffe62bf4ffe:/home/DL/assn1$ python one_hidden_layer_neural_network_classifier.py --noise 0.1 --units 7
Accuracy of one hidden layer neural network classifier with 0.1 noise parameter and 7 hidden units is 92.5%!
(assn) user@7ffe62bf4ffe:/home/DL/assn1$ python one_hidden_layer_neural_network_classifier.py --noise 0.1 --units 9
Accuracy of one hidden layer neural network classifier with 0.1 noise parameter and 9 hidden units is 88.0%!
(assn) user@7ffe62bf4ffe:/home/DL/assn1$ python one_hidden_layer_neural_network_classifier.py --noise 0.3 --units 1
Accuracy of one hidden layer neural network classifier with 0.3 noise parameter and 1 hidden units is 82.0%!
(assn) user@7ffe62bf4ffe:/home/DL/assn1$ python one_hidden_layer_neural_network_classifier.py --noise 0.3 --units 3
Accuracy of one hidden layer neural network classifier with 0.3 noise parameter and 3 hidden units is 86.0%!
(assn) user@7ffe62bf4ffe:/home/DL/assn1$ python one_hidden_layer_neural_network_classifier.py --noise 0.3 --units 5
Accuracy of one hidden layer neural network classifier with 0.3 noise parameter and 5 hidden units is 84.5%!
(assn) user@7ffe62bf4ffe:/home/DL/assn1$ python one_hidden_layer_neural_network_classifier.py --noise 0.3 --units 7
Accuracy of one hidden layer neural network classifier with 0.3 noise parameter and 7 hidden units is 83.5%!
(assn) user@7ffe62bf4ffe:/home/DL/assn1$ python one_hidden_layer_neural_network_classifier.py --noise 0.3 --units 9
Accuracy of one hidden layer neural network classifier with 0.3 noise parameter and 9 hidden units is 84.0%!
(assn) user@7ffe62bf4ffe:/home/DL/assn1$ python one_hidden_layer_neural_network_classifier.py --noise 0.5 --units 1
Accuracy of one hidden layer neural network classifier with 0.5 noise parameter and 1 hidden units is 80.5%!
(assn) user@7ffe62bf4ffe:/home/DL/assn1$ python one_hidden_layer_neural_network_classifier.py --noise 0.5 --units 3
Accuracy of one hidden layer neural network classifier with 0.5 noise parameter and 3 hidden units is 85.0%!
(assn) user@7ffe62bf4ffe:/home/DL/assn1$ python one_hidden_layer_neural_network_classifier.py --noise 0.5 --units 5
Accuracy of one hidden layer neural network classifier with 0.5 noise parameter and 5 hidden units is 85.5%!
(assn) user@7ffe62bf4ffe:/home/DL/assn1$ python one_hidden_layer_neural_network_classifier.py --noise 0.5 --units 7
Accuracy of one hidden layer neural network classifier with 0.5 noise parameter and 7 hidden units is 85.0%!
(assn) user@7ffe62bf4ffe:/home/DL/assn1$ python one_hidden_layer_neural_network_classifier.py --noise 0.5 --units 9
Accuracy of one hidden layer neural network classifier with 0.5 noise parameter and 9 hidden units is 84.5%!
(assn) user@7ffe62bf4ffe:/home/DL/assn1$ python one_hidden_layer_neural_network_classifier.py --noise 0.7 --units 1
Accuracy of one hidden layer neural network classifier with 0.7 noise parameter and 1 hidden units is 71.5%!
(assn) user@7ffe62bf4ffe:/home/DL/assn1$ python one_hidden_layer_neural_network_classifier.py --noise 0.7 --units 3
Accuracy of one hidden layer neural network classifier with 0.7 noise parameter and 3 hidden units is 72.0%!
(assn) user@7ffe62bf4ffe:/home/DL/assn1$ python one_hidden_layer_neural_network_classifier.py --noise 0.7 --units 5
Accuracy of one hidden layer neural network classifier with 0.7 noise parameter and 5 hidden units is 71.5%!
(assn) user@7ffe62bf4ffe:/home/DL/assn1$ python one_hidden_layer_neural_network_classifier.py --noise 0.7 --units 7
Accuracy of one hidden layer neural network classifier with 0.7 noise parameter and 7 hidden units is 72.5%!
(assn) user@7ffe62bf4ffe:/home/DL/assn1$ python one_hidden_layer_neural_network_classifier.py --noise 0.7 --units 9
Accuracy of one hidden layer neural network classifier with 0.7 noise parameter and 9 hidden units is 72.5%!
(assn) user@7ffe62bf4ffe:/home/DL/assn1$ python one_hidden_layer_neural_network_classifier.py --noise 0.9 --units 1
Accuracy of one hidden layer neural network classifier with 0.9 noise parameter and 1 hidden units is 71.0%!
(assn) user@7ffe62bf4ffe:/home/DL/assn1$ python one_hidden_layer_neural_network_classifier.py --noise 0.9 --units 3
Accuracy of one hidden layer neural network classifier with 0.9 noise parameter and 3 hidden units is 68.0%!
(assn) user@7ffe62bf4ffe:/home/DL/assn1$ python one_hidden_layer_neural_network_classifier.py --noise 0.9 --units 5
Accuracy of one hidden layer neural network classifier with 0.9 noise parameter and 5 hidden units is 69.0%!
(assn) user@7ffe62bf4ffe:/home/DL/assn1$ python one_hidden_layer_neural_network_classifier.py --noise 0.9 --units 7
Accuracy of one hidden layer neural network classifier with 0.9 noise parameter and 7 hidden units is 69.5%!
(assn) user@7ffe62bf4ffe:/home/DL/assn1$ python one_hidden_layer_neural_network_classifier.py --noise 0.9 --units 9
Accuracy of one hidden layer neural network classifier with 0.9 noise parameter and 9 hidden units is 68.0%!
```

This is the result of measuring the accuracy while changing the number of hidden layer units for each noise.

Noise Parameter / # Hidden Units	**0.1**	**0.3**	**0.5**	**0.7**	**0.9**
1	85.5	82.0	80.5	71.5	71.0
3	88.5	86.0	85.0	72.0	68.0
5	87.5	84.5	85.5	71.5	69.0
7	92.5	83.5	85.0	72.5	69.5
9	88.0	84.0	84.5	72.5	68.0

Accuracy(%)

3.2. Analyze the classifier’s accuracy with different number of hidden units for each dataset.

Looking at the classifier in which the hidden layer is added between the input layer and the output layer, it may be seen that performance is measured differently depending on the number of units in the hidden layer. When adding one hidden layer, the difference according to the number of units does not appear to be significant, but it is true that the number of hidden units generally affects the result. In the intermediate step, the hidden units are calculated and the final result is calculated again using the hidden units, but the number of hidden units is determined by the hyper parameter. The performance can be improved only when the number of hidden units is well set. Expressive power is sufficient when using a small number of hidden units, but in fact, this classification task is simple and I don't think the number of hidden units has had a great significant impact.

3.3. Analyze the classifier’s accuracy with respect to the noise value.

I think the difference in performance according to the noise value is the same as the previous logistic regression classifier. The difference, however, is that the addition of a hidden layer has led to better performance. However, if the noise is small, the existing data set is visually easy to separate, so even if we add one hidden layer, I think the smaller the noise, the better the classification is. If the noise increases, I think it is worse in terms of performance because we want to perform training process even the data set mixed with noise.

4. Two Hidden Layer Neural Network Classifier

This time, we can design a classifier by adding one more hidden layer between the input layer and the output layer.

4.1. Find the optimal two hidden layer NN classifiers for each dataset by changing the number of hidden units over hidden layer 1 and 2.

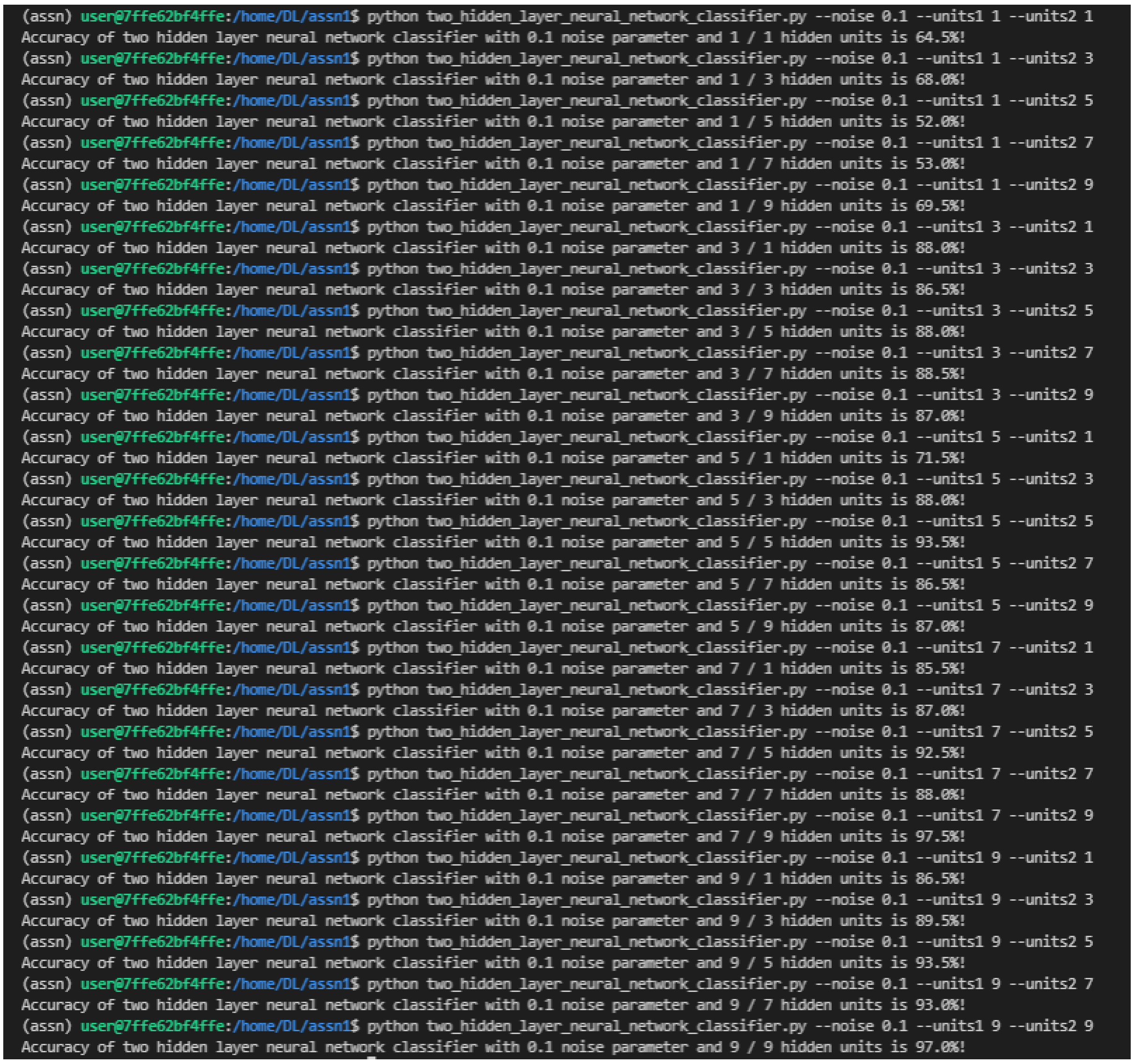
Since one more hidden layer has been added, some parameters are added than a classifier with one hidden layer and also additional calculation is required.

```
# Parameter(w,b) initialization
    def parameter_initialization(self):
        self.weight1 = np.random.randn(self.num_features, self.num_units1) # (2, 1/3/5/7/9)
        self.bias1 = np.random.randn(1, 1) # (1, 1)
        self.weight2 = np.random.randn(self.num_units1, self.num_units2) # (1/3/5/7/9, 1/3/5/7/9)
        self.bias2 = np.random.randn(1, 1) # (1, 1)
        self.weight3 = np.random.randn(self.num_units2, 1) # (1/3/5/7/9, 1)
        self.bias3 = np.random.randn(1, 1) # (1, 1)
```

Training and test processes also require additional operations, where it is noteworthy that performance loss could be prevented by using sigmoid only at the end of activation function.

```
# Training : back propagation
    def back_propagation_training(self, X, predict_y, real_y):
        dA3 = (-1) * np.divide(real_y, predict_y) + np.divide(1 - real_y, 1 - predict_y)
        dZ3 = dA3 * 1 / (1 + np.exp(-self.Z3)) * (1 - 1 / (1 + np.exp(-self.Z3)))
        dw3 = (1 / self.num_samples) * np.dot(self.A2.T, dZ3)
        db3 = (1 / self.num_samples) * np.sum(dZ3)

        dA2 = np.dot(dZ3, self.weight3.T)
        dZ2 = dA2 * 1 / (1 + np.exp(-self.Z2)) * (1 - 1 / (1 + np.exp(-self.Z2)))
        dw2 = (1 / self.num_samples) * np.dot(self.A1.T, dZ2)
        db2 = (1 / self.num_samples) * np.sum(dZ2)

        dA1 = np.dot(dZ2, self.weight2.T)
        dZ1 = dA1 * 1 / (1 + np.exp(-self.Z1)) * (1 - 1 / (1 + np.exp(-self.Z1)))
        dw1 = (1 / self.num_samples) * np.dot(X.T, dZ1)
        db1 = (1 / self.num_samples) * np.sum(dZ1)

        # Parameter update
        self.weight1 = self.weight1 - self.lr * dw1
        self.bias1 = self.bias1 - self.lr * db1
        self.weight2 = self.weight2 - self.lr * dw2
        self.bias2 = self.bias2 - self.lr * db2
        self.weight3 = self.weight3 - self.lr * dw3
        self.bias3 = self.bias3 - self.lr * db3

    # Training : forward propagation + back propagation
    def training(self, X, real_y):
        # Parameter intialization
        self.parameter_initialization()

        for i in range(self.epoch):
            predict_y = self.forward_propagation_training(X)
            self.back_propagation_training(X, predict_y, real_y)
```

```
# Test : foward propagation and classification
    def forward_propagation_test(self, X):
        Z1 = np.dot(X, self.weight1) + self.bias1
        A1 = self.activation_function_relu(Z1)
        Z2 = np.dot(A1, self.weight2) + self.bias2
        A2 = self.activation_function_relu(Z2)
        Z3 = np.dot(A2, self.weight3) + self.bias3
        A3 = self.activation_function_sigmoid(Z3)

        for i in range(A3.shape[0]):
            if A3[i][0] > 0.5:
                A3[i][0] = 1
            else:
                A3[i][0] = 0

        return A3

    # Test : accuracy
    def accuracy_test(self, X, real_y, noise, num_units1, num_units2):
        predict_y = self.forward_propagation_test(X)
        num_test_samples = len(predict_y)

        correct = 0
        for i in range(num_test_samples):
            if(predict_y[i] == real_y[i]):
                correct = correct + 1
        accuracy = correct / num_test_samples * 100

        print('Accuracy of two hidden layer neural network classifier with ' + str(noise) + ' noise
parameter and ' + str(num_units1) + ' / ' + str(num_units2) + ' hidden units is ' + str(accuracy) +
'%' + '!')
```



```
(assn) user@7ffe62bf4ffe:/home/DL/assn1$ python two_hidden_layer_neural_network_classifier.py --noise 0.1 --units1 1 --units2 1
Accuracy of two hidden layer neural network classifier with 0.1 noise parameter and 1 / 1 hidden units is 64.5%!
(assn) user@7ffe62bf4ffe:/home/DL/assn1$ python two_hidden_layer_neural_network_classifier.py --noise 0.1 --units1 1 --units2 3
Accuracy of two hidden layer neural network classifier with 0.1 noise parameter and 1 / 3 hidden units is 68.0%!
(assn) user@7ffe62bf4ffe:/home/DL/assn1$ python two_hidden_layer_neural_network_classifier.py --noise 0.1 --units1 1 --units2 5
Accuracy of two hidden layer neural network classifier with 0.1 noise parameter and 1 / 5 hidden units is 52.0%!
(assn) user@7ffe62bf4ffe:/home/DL/assn1$ python two_hidden_layer_neural_network_classifier.py --noise 0.1 --units1 1 --units2 7
Accuracy of two hidden layer neural network classifier with 0.1 noise parameter and 1 / 7 hidden units is 53.0%!
(assn) user@7ffe62bf4ffe:/home/DL/assn1$ python two_hidden_layer_neural_network_classifier.py --noise 0.1 --units1 1 --units2 9
Accuracy of two hidden layer neural network classifier with 0.1 noise parameter and 1 / 9 hidden units is 69.5%!
(assn) user@7ffe62bf4ffe:/home/DL/assn1$ python two_hidden_layer_neural_network_classifier.py --noise 0.1 --units1 3 --units2 1
Accuracy of two hidden layer neural network classifier with 0.1 noise parameter and 3 / 1 hidden units is 88.0%!
(assn) user@7ffe62bf4ffe:/home/DL/assn1$ python two_hidden_layer_neural_network_classifier.py --noise 0.1 --units1 3 --units2 3
Accuracy of two hidden layer neural network classifier with 0.1 noise parameter and 3 / 3 hidden units is 86.5%!
(assn) user@7ffe62bf4ffe:/home/DL/assn1$ python two_hidden_layer_neural_network_classifier.py --noise 0.1 --units1 3 --units2 5
Accuracy of two hidden layer neural network classifier with 0.1 noise parameter and 3 / 5 hidden units is 88.0%!
(assn) user@7ffe62bf4ffe:/home/DL/assn1$ python two_hidden_layer_neural_network_classifier.py --noise 0.1 --units1 3 --units2 7
Accuracy of two hidden layer neural network classifier with 0.1 noise parameter and 3 / 7 hidden units is 88.5%!
(assn) user@7ffe62bf4ffe:/home/DL/assn1$ python two_hidden_layer_neural_network_classifier.py --noise 0.1 --units1 3 --units2 9
Accuracy of two hidden layer neural network classifier with 0.1 noise parameter and 3 / 9 hidden units is 87.0%!
(assn) user@7ffe62bf4ffe:/home/DL/assn1$ python two_hidden_layer_neural_network_classifier.py --noise 0.1 --units1 5 --units2 1
Accuracy of two hidden layer neural network classifier with 0.1 noise parameter and 5 / 1 hidden units is 71.5%!
(assn) user@7ffe62bf4ffe:/home/DL/assn1$ python two_hidden_layer_neural_network_classifier.py --noise 0.1 --units1 5 --units2 3
Accuracy of two hidden layer neural network classifier with 0.1 noise parameter and 5 / 3 hidden units is 88.0%!
(assn) user@7ffe62bf4ffe:/home/DL/assn1$ python two_hidden_layer_neural_network_classifier.py --noise 0.1 --units1 5 --units2 5
Accuracy of two hidden layer neural network classifier with 0.1 noise parameter and 5 / 5 hidden units is 93.5%!
(assn) user@7ffe62bf4ffe:/home/DL/assn1$ python two_hidden_layer_neural_network_classifier.py --noise 0.1 --units1 5 --units2 7
Accuracy of two hidden layer neural network classifier with 0.1 noise parameter and 5 / 7 hidden units is 86.5%!
(assn) user@7ffe62bf4ffe:/home/DL/assn1$ python two_hidden_layer_neural_network_classifier.py --noise 0.1 --units1 5 --units2 9
Accuracy of two hidden layer neural network classifier with 0.1 noise parameter and 5 / 9 hidden units is 87.0%!
(assn) user@7ffe62bf4ffe:/home/DL/assn1$ python two_hidden_layer_neural_network_classifier.py --noise 0.1 --units1 7 --units2 1
Accuracy of two hidden layer neural network classifier with 0.1 noise parameter and 7 / 1 hidden units is 85.5%!
(assn) user@7ffe62bf4ffe:/home/DL/assn1$ python two_hidden_layer_neural_network_classifier.py --noise 0.1 --units1 7 --units2 3
Accuracy of two hidden layer neural network classifier with 0.1 noise parameter and 7 / 3 hidden units is 87.0%!
(assn) user@7ffe62bf4ffe:/home/DL/assn1$ python two_hidden_layer_neural_network_classifier.py --noise 0.1 --units1 7 --units2 5
Accuracy of two hidden layer neural network classifier with 0.1 noise parameter and 7 / 5 hidden units is 92.5%!
(assn) user@7ffe62bf4ffe:/home/DL/assn1$ python two_hidden_layer_neural_network_classifier.py --noise 0.1 --units1 7 --units2 7
Accuracy of two hidden layer neural network classifier with 0.1 noise parameter and 7 / 7 hidden units is 88.0%!
(assn) user@7ffe62bf4ffe:/home/DL/assn1$ python two_hidden_layer_neural_network_classifier.py --noise 0.1 --units1 7 --units2 9
Accuracy of two hidden layer neural network classifier with 0.1 noise parameter and 7 / 9 hidden units is 97.5%!
(assn) user@7ffe62bf4ffe:/home/DL/assn1$ python two_hidden_layer_neural_network_classifier.py --noise 0.1 --units1 9 --units2 1
Accuracy of two hidden layer neural network classifier with 0.1 noise parameter and 9 / 1 hidden units is 86.5%!
(assn) user@7ffe62bf4ffe:/home/DL/assn1$ python two_hidden_layer_neural_network_classifier.py --noise 0.1 --units1 9 --units2 3
Accuracy of two hidden layer neural network classifier with 0.1 noise parameter and 9 / 3 hidden units is 89.5%!
(assn) user@7ffe62bf4ffe:/home/DL/assn1$ python two_hidden_layer_neural_network_classifier.py --noise 0.1 --units1 9 --units2 5
Accuracy of two hidden layer neural network classifier with 0.1 noise parameter and 9 / 5 hidden units is 93.5%!
(assn) user@7ffe62bf4ffe:/home/DL/assn1$ python two_hidden_layer_neural_network_classifier.py --noise 0.1 --units1 9 --units2 7
Accuracy of two hidden layer neural network classifier with 0.1 noise parameter and 9 / 7 hidden units is 93.0%!
(assn) user@7ffe62bf4ffe:/home/DL/assn1$ python two_hidden_layer_neural_network_classifier.py --noise 0.1 --units1 9 --units2 9
Accuracy of two hidden layer neural network classifier with 0.1 noise parameter and 9 / 9 hidden units is 97.0%!
```

The following is the result of measuring the performance while changing the number of hidden units of two hidden layers according to the noise parameter. For each noise, the combination of the highest performance hidden units had a green highlight, followed by a yellow highlight.

First Layer / **Second Layer**	**1**	**3**	**5**	**7**	**9**
1	64.5	88.0	71.5	85.5	86.5
3	68.0	86.5	88.0	87.0	89.5
5	52.0	88.0	93.5	92.5	93.5
7	53.0	88.5	86.5	88.0	93.0
9	69.5	87.0	87.0	97.5	97.0

Noise Parameter = 0.1

Second Layer \ First Layer	1	3	5	7	9
1	62.0	83.5	71.5	81.0	83.0
3	68.0	84.0	83.5	85.0	85.5
5	56.0	85.5	88.0	86.0	86.0
7	51.0	86.0	84.0	81.0	87.5
9	68.5	84.5	84.0	87.5	90.5

Noise Parameter = 0.3

Second Layer \ First Layer	1	3	5	7	9
1	66.0	83.5	76.5	84.0	84.0
3	73.0	84.0	49.0	84.5	82.0
5	62.0	87.0	85.0	85.0	85.0
7	59.5	86.5	84.0	81.0	87.0
9	69.5	86.5	84.5	86.5	86.5

Noise Parameter = 0.5

Second Layer \ First Layer	1	3	5	7	9
1	69.5	71.0	72.5	72.5	74.0
3	72.0	72.0	71.0	74.0	73.5
5	72.0	73.0	73.5	73.5	70.5
7	72.0	72.5	71.5	72.5	75.5
9	72.0	74.0	72.0	73.0	73.0

Noise Parameter = 0.7

Second Layer \ First Layer	1	3	5	7	9
1	70.5	70.5	69.0	69.0	68.5
3	69.0	69.0	70.5.	69.5	69.5
5	69.0	69.5	69.5	69.0	67.5
7	69.0	68.5	69.0	69.5	69.5
9	69.0	69.0	68.0	71.0	69.5

Noise Parameter = 0.9

4.2. What did you learn through this project?

First of all, only the best performance out of a total of three classifier performances conducted in this task is summarized separately.

Method \ Noise Parameter	0.1	0.3	0.5	0.7	0.9
Logistic	88.0	85.0	85.0	72.5	69.0
One Hidden	88.5	85.5	85.0	73.5	69.5
Two Hidden	97.5	90.5	87.0	75.5	71.0

Looking at the performance change according to the number of hidden units of the hidden layer, it may be seen that the number of units of two hidden layers does not have a significant effect, but the performance is certainly measured differently. Since it is a simple classification task with only two input features, I think that using two hidden layers already has sufficient expressive power. Therefore, I think that too many hidden layers or hidden units proceed close to overfitting. When comparing noise to each other, I think it is clear that the smaller the noise value, the better the performance, and this is because the classifiers can learn a given dataset better.

Through this project, I had time to build layers by writing code one by one without using the good frameworks that were widely used. By writing the process of creating and calculating the layer myself with a formula, I could see how the model learns from the data. And through various experiments, I realized the importance of hyper parameters, and I think it was a good experience in the process of building a model in the future.