There are 5 Python files and a letter recognition data file in the project archive file:

1. MLP.py

In this class I create an MLP object with the following methods and attributes:

(1) __init__(self, input_size, hidden_size, output_size): Construct a single hidden layer neural network
    input_size: The Number of units in the input layer.
    hidden_size: The Number of units in the hidden layer.
    output_size: The Number of units in the output layer.
    Loss function: MSE

(2) __randomise(self): Randomly initialize the parameters of the neural network.
    w1: Parameters from the input layer to the hidden layer. (that is, array containing the weights in the lower layer)
    w2: Parameters from the hidden layer to the output layer.
    b1: Bias item on the hidden layer.
    b2: Bias item on the output layer.

(3) _activate(self, x, activation_name): Active the activation function according to the corresponding name of activation function, return the name.

(4) __dactivate(self, x, activation_name): Find the derivative of the activation function at x according to the activation function name.

(5) forward(self, x): Forward propagation, recording output of each layer.
    x: shape (1, input_size)

(6) backwards(self, x, y): Backpropagation, get the corresponding dW1, db1, dW2, db2 of W1, b1, W2, b2.
    x: input of the sample
    y: label of the sample

(7) update(self, dW2, db2, dW1, db1):

(8) train(self, x, y, iterations=1000, lr=0.01, activation_hidden='tanh', activation_output='tanh', loss_func='crossEntropy'): Train the model.

lr: learning rate

activation_hidden: activation function of the hidden layer

activation_output: activation function of the output layer

(9) predict(self, x): Predict based on the trained parameters.

2. Test_XOR.py

(1) Train an MLP with 2 inputs, 2 hidden units and 1 output:

NN = MLP(2, 2, 1)

with examples:

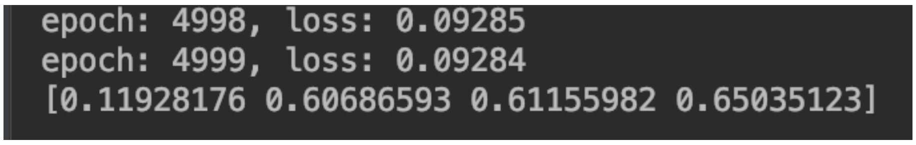
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])

y = np.array([[0], [1], [1], [0]])

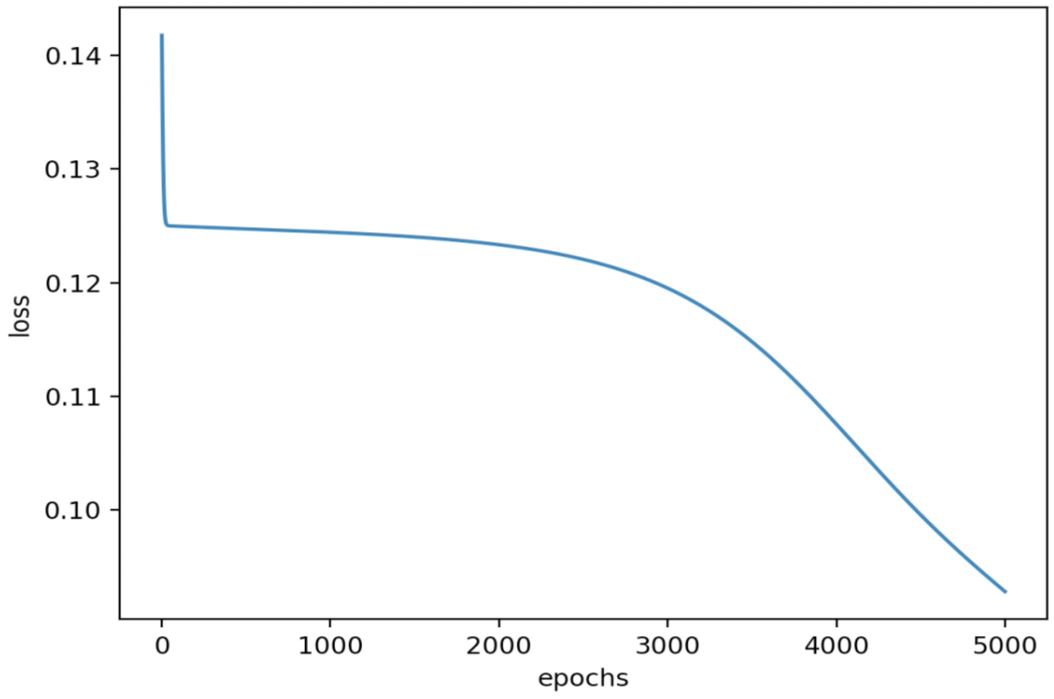
I tried different learning rates (0.5&0.1), different numbers of training and different combinations of activation function.

1) When the hidden layer activation function is sigmoid, and the output layer activation function is tanh:

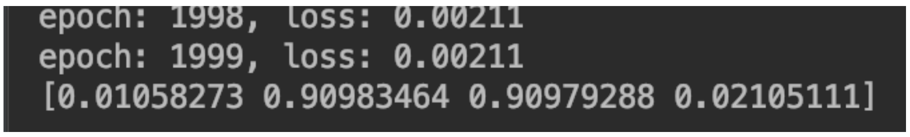
a. Learning rate = 0.1, training times = 5000



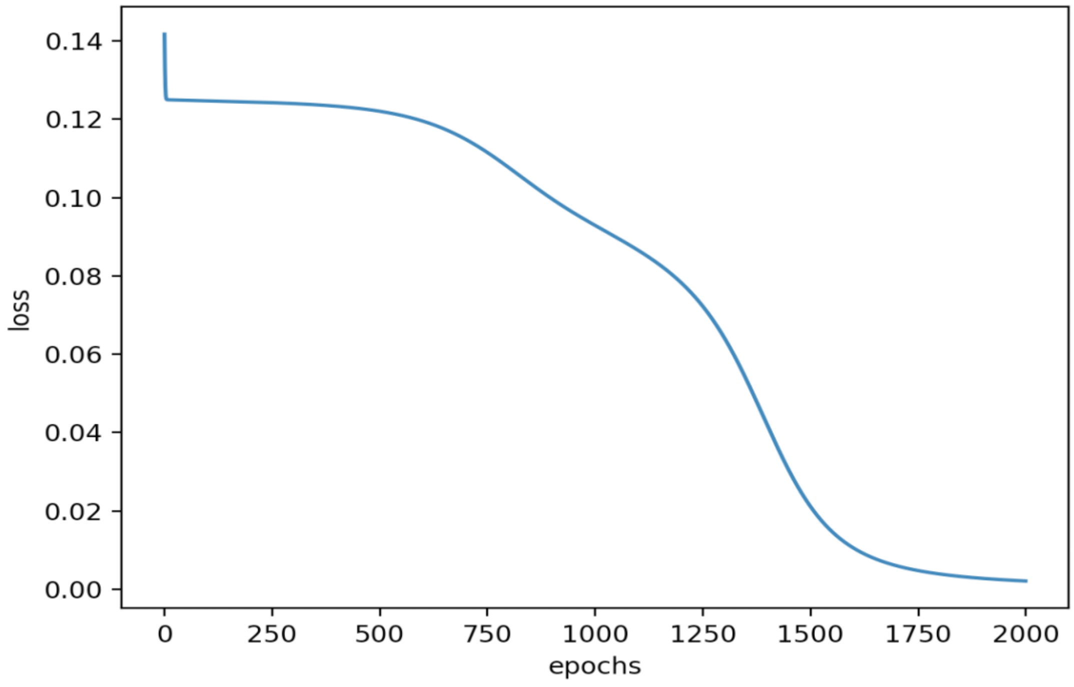
```
epoch: 4998, loss: 0.09285
epoch: 4999, loss: 0.09284
[0.11928176 0.60686593 0.61155982 0.65035123]
```



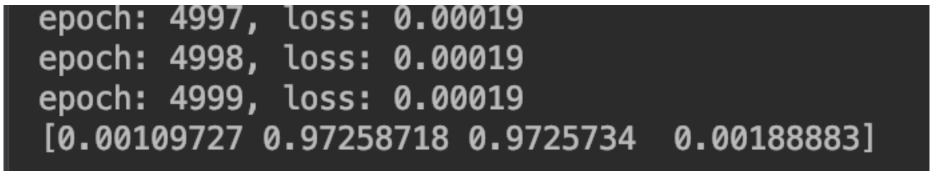
b. Learning rate = 0.5, training times= 2000



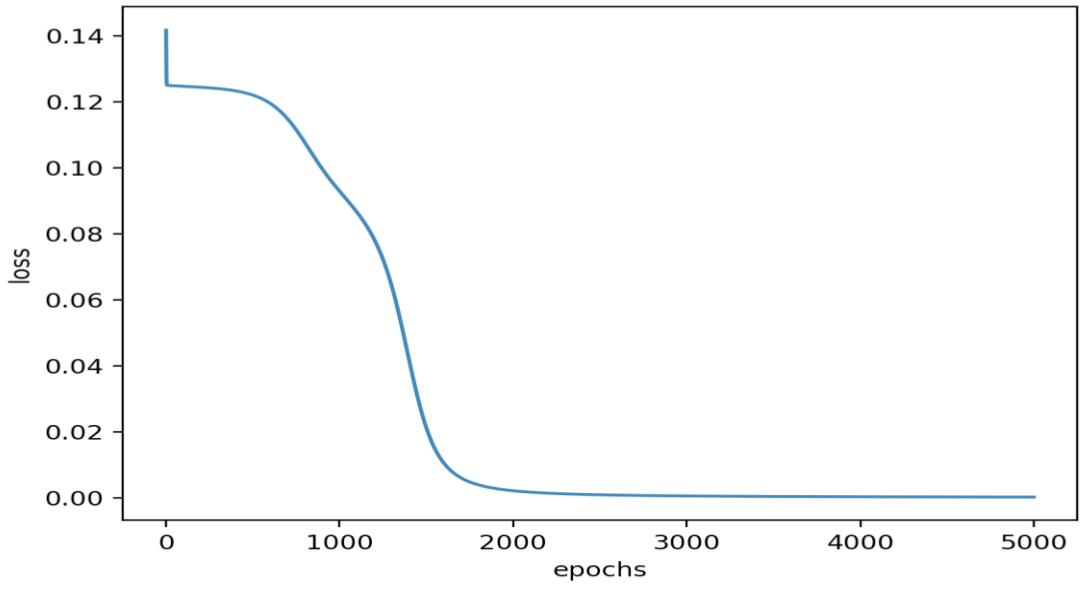
```
epoch: 1998, loss: 0.00211
epoch: 1999, loss: 0.00211
[0.01058273 0.90983464 0.90979288 0.02105111]
```



c. Learning rate = 0.5, training times = 5000



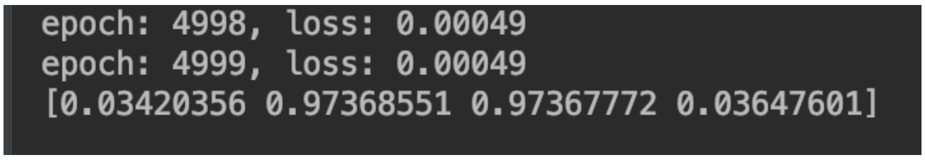
```
epoch: 4997, loss: 0.00019
epoch: 4998, loss: 0.00019
epoch: 4999, loss: 0.00019
[0.00109727 0.97258718 0.9725734  0.00188883]
```



When the learning rate is 0.1, the effect achieved by training 5000 times is not as good as the training 2000 times when the learning rate is 0.5. When the learning rate is 0.5, it can be seen that the prediction results gradually stabilize after 2000 training sessions.

2) When the hidden layer activation function is tanh, and the output layer activation function is sigmoid, I got the same trend of results with combinations of learning rate and training time.

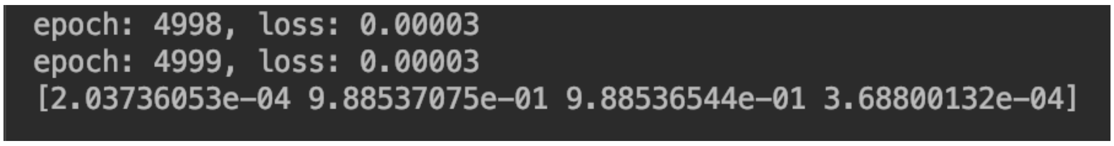
This the result with learning rate = 0.5, training times = 5000:



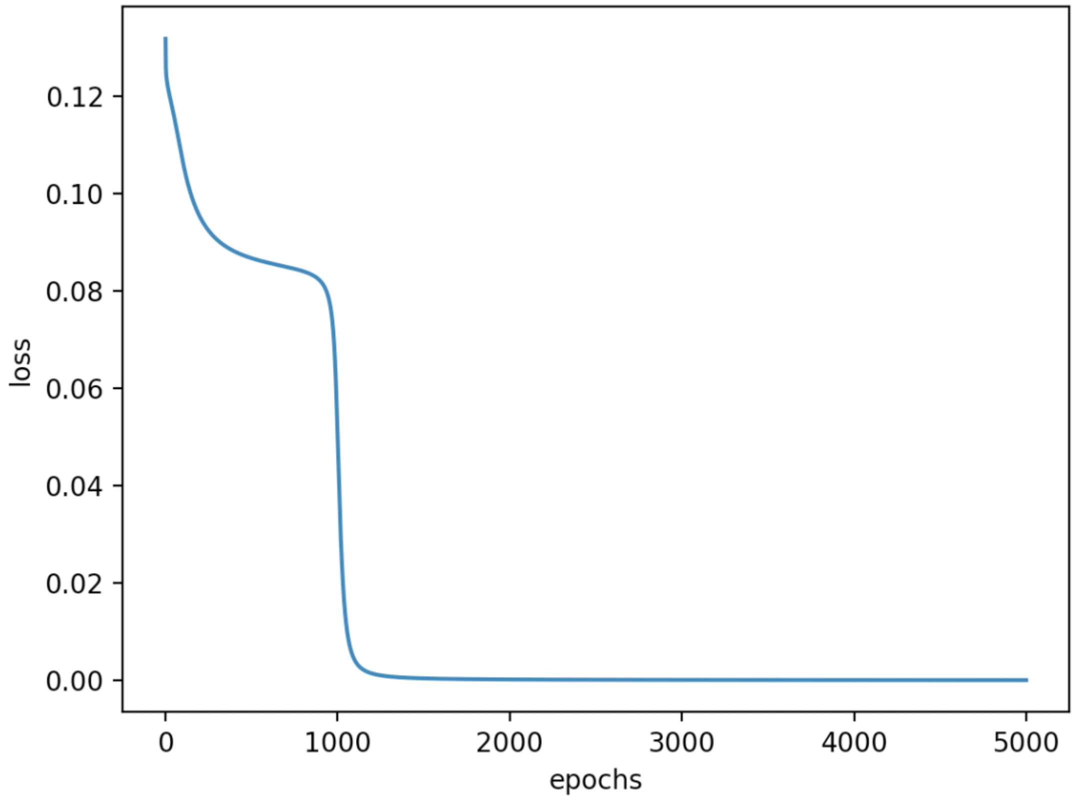
```
epoch: 4998, loss: 0.00049
epoch: 4999, loss: 0.00049
[0.03420356 0.97368551 0.97367772 0.03647601]
```

3) When the hidden layer activation function and the output layer activation function is both tanh,

   Learning rate = 0.5, training times = 5000:



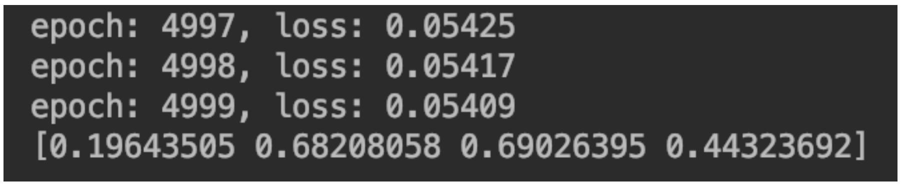
```
epoch: 4998, loss: 0.00003
epoch: 4999, loss: 0.00003
[2.03736053e-04 9.88537075e-01 9.88536544e-01 3.68800132e-04]
```



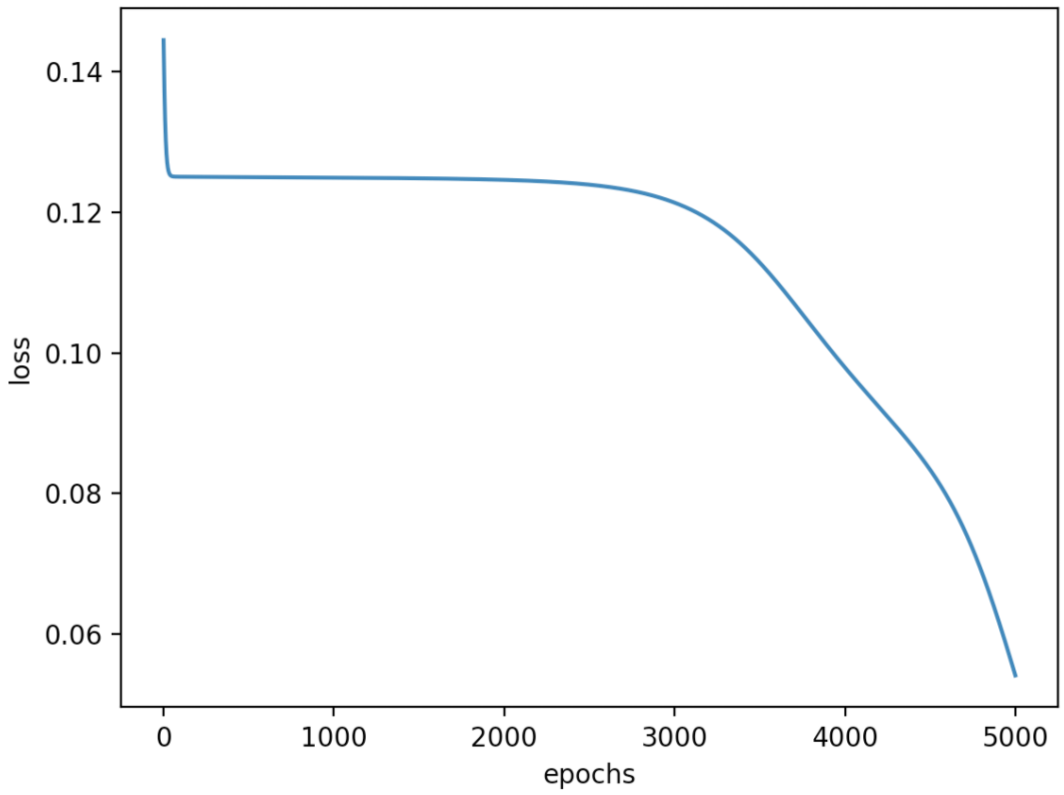
4) Based on the result in 3), maybe when the hidden layer activation function and

the output layer activation function is both sigmoid we can also get a good prediction.

So When the hidden layer activation function and the output layer activation function is both sigmoid, learning rate = 0.5, training times = 5000:



```
epoch: 4997, loss: 0.05425
epoch: 4998, loss: 0.05417
epoch: 4999, loss: 0.05409
[0.19643505 0.68208058 0.69026395 0.44323692]
```



There is a very bad prediction.

The experiments I did didn't include all the cases, but in general, the prediction result obtained when the learning rate is 0.5 is better than the learning rate of 0.1, and when the hidden layer activation function and the output layer activation function is both sigmoid, the result is not as good as other function combinations.

(2) When there are two categories, sigmoid can only output one value. If the value is greater than 0.5, we will judge the label as 1, and less than 0.5 will be judged as 0. However, when there are more than two categories of classification results, if there is only one output, we can't judge which type it is based on the

output, so I have also done a one-hot encoding experiment, which may be used in the later letter recognition. For example, there are two types of sample tags, 0 and 1, and I mark the original 0 as [1,0], the original 1 as [0,1], and then let the output of neural network from the 1 unit turn into 2 units, get the output through the softmax function, we just need to make the output as large as possible in the first dimension or the second dimension.

(Reference to James's blog, available at https://jameskle.com/writes/neural-networks-101)

Train an MLP with 2 inputs, 2 hidden units and 2 output:

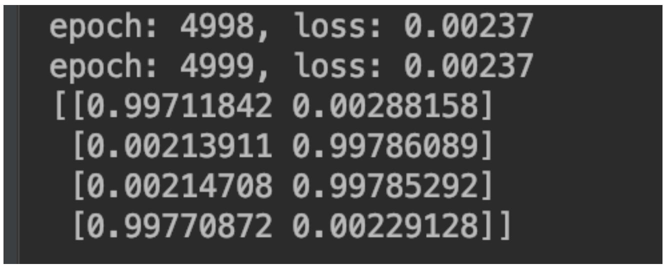
NN = MLP(2, 2, 2)

with examples:

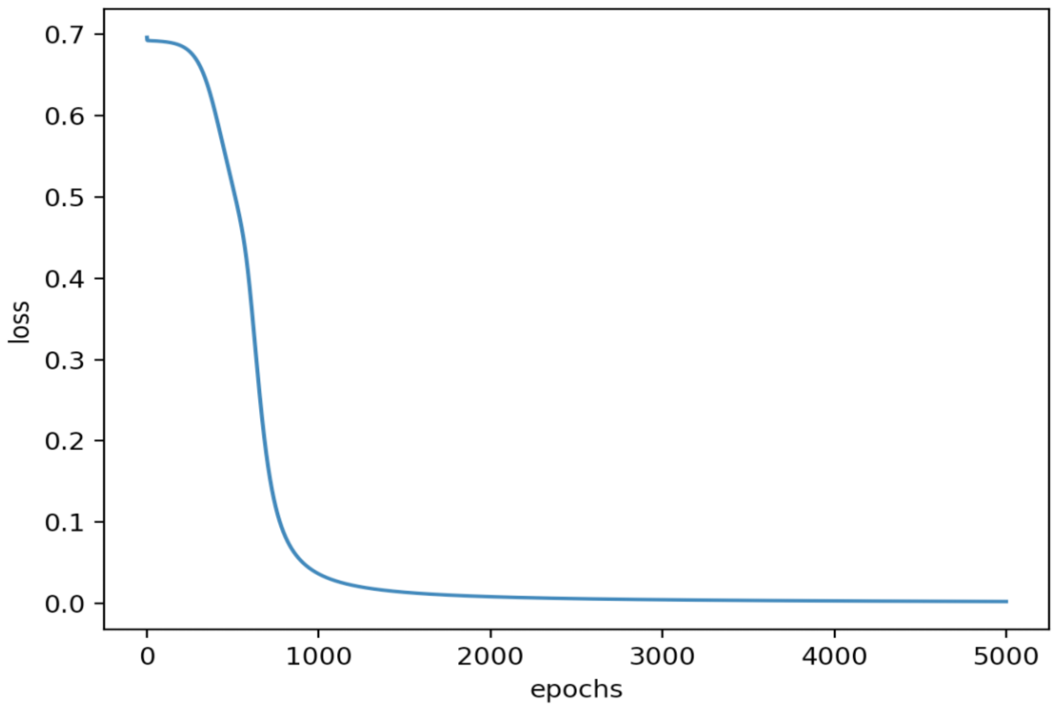
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])

y = np.array([[1, 0], [0, 1], [0, 1], [1, 0]])

when learning rate = 0.5, training times = 5000:



```
epoch: 4998, loss: 0.00237
epoch: 4999, loss: 0.00237
[[0.99711842 0.00288158]
 [0.00213911 0.99786089]
 [0.00214708 0.99785292]
 [0.99770872 0.00229128]]
```



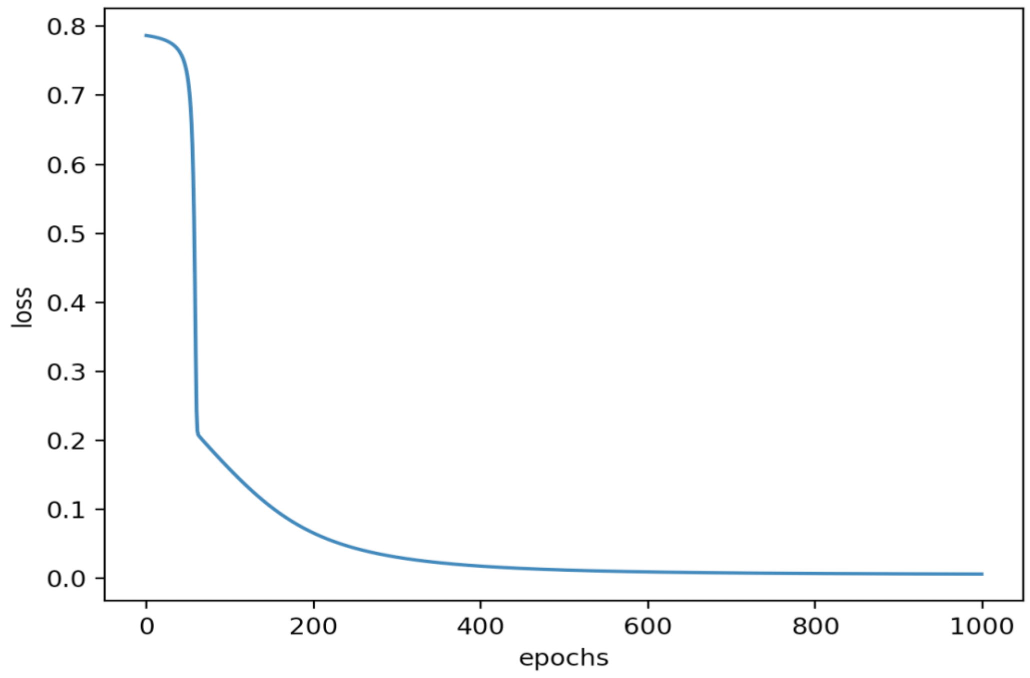
The result is better than the above experiments.

3. Test_sin.py

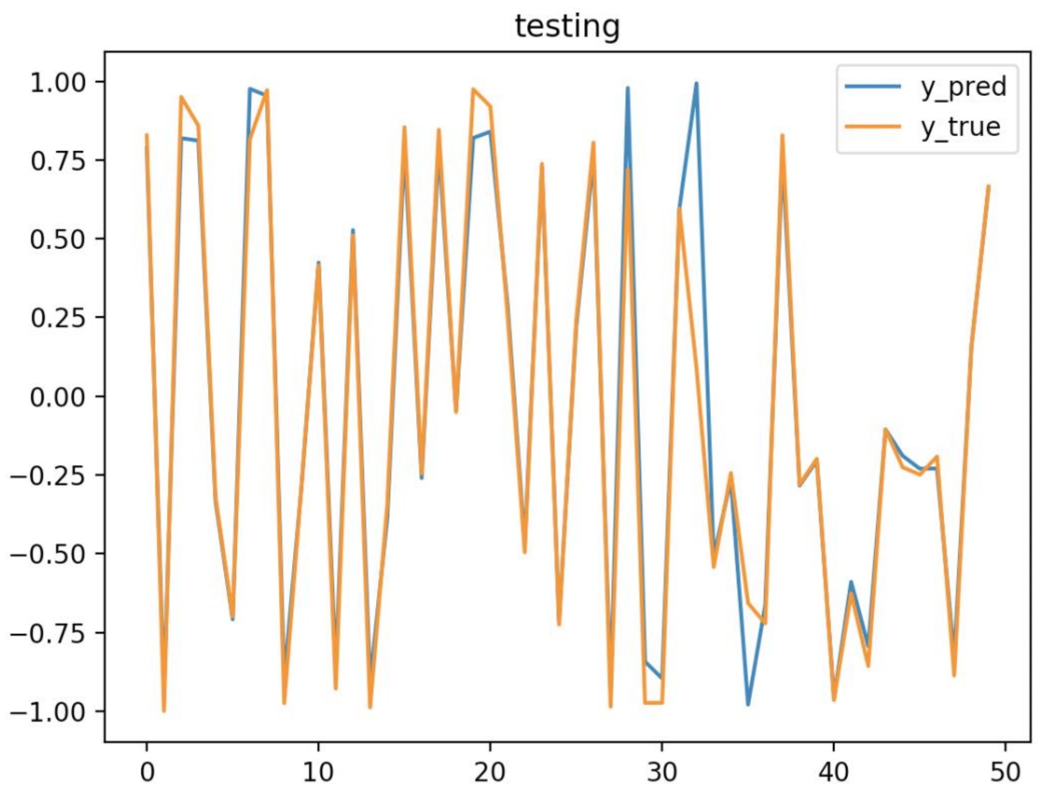
Train an MLP with 4 inputs, 16 hidden units and 1 output.

Set learning rate = 0.1, training times = 1000, activation_hidden = 'sigmoid', activation_output='tanh', loss_func = 'mse':

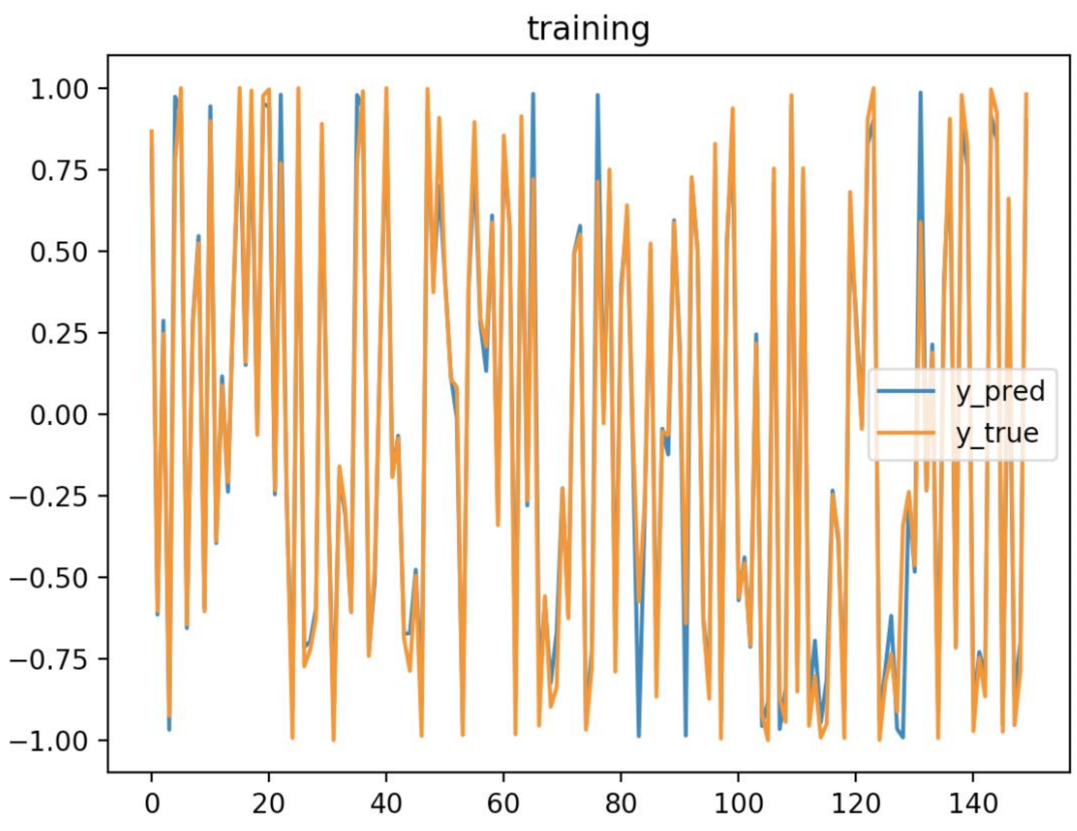
Plot the change in loss during the training process:



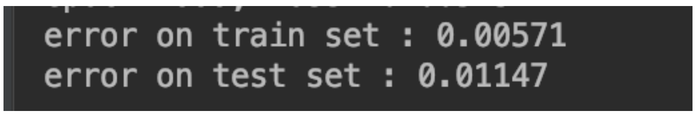
Plot the comparison between test set predictions and the real values:



Plot the comparison between training set predictions and the real values:

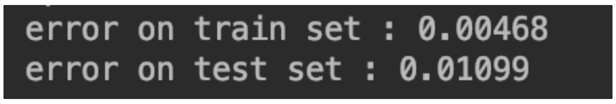


Calculate the average error on the training set and the test set:



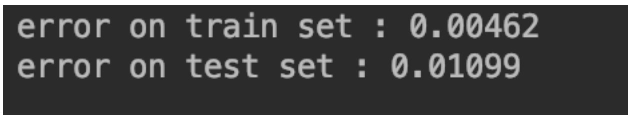
```
error on train set : 0.00571
error on test set : 0.01147
```

The error generated on the training set is less than that produced on the test set, they all didn't get the zero error. When the learning rate is 0.5 and the training times are 5000:



```
error on train set : 0.00468
error on test set : 0.01099
```

And when the learning rate is 0.5 and the training times are 10000:



```
error on train set : 0.00462
error on test set : 0.01099
```

There is still the error. And I couldn't figure out the reason.

4. Test_letter_recognition.py & utils.py

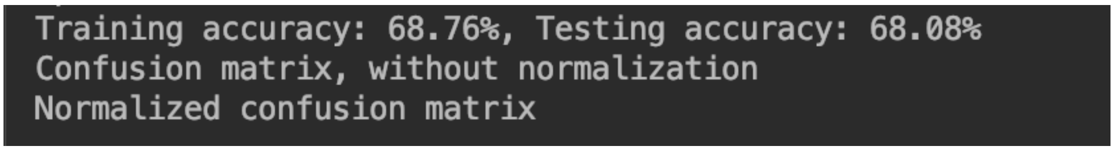
Using one-hot encoding, calling the utils.py to plot the confusion matrix.

utils.py is an example of confusion matrix usage to evaluate the quality of the output of a classifier on the iris data set, available at https://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html

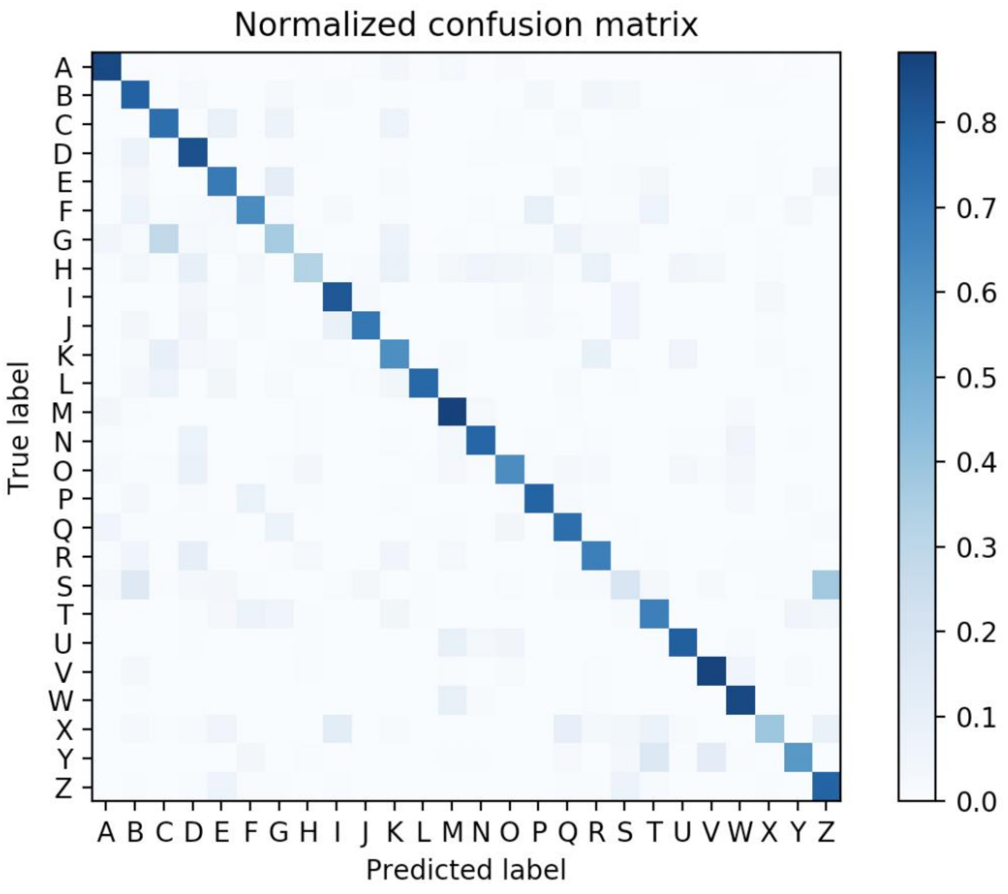
Set the number of input units as 16, number of hidden units as 32 and number of outputs as 26, the learning rate is 0.1, and train the MLP 2000 epochs:

The training accuracy is very close to the testing accuracy, but only 68%. If the learning rate is higher, and iterate more times, there will be a better accuracy.

The training accuracy is very close to the testing accuracy, but only 68%. If the learning rate is higher, and iterate more times, there will be higher accuracy.



```
Training accuracy: 68.76%, Testing accuracy: 68.08%
Confusion matrix, without normalization
Normalized confusion matrix
```


Normalized confusion matrix
True label
A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
P
Q
R
S
T
U
V
W
X
Y
Z
Predicted label