Summary

The homework is to identify handwriting figures with Single-layer Linear Perceptron, Single-layer Perceptron, Multi-layer Perceptron, and Convolutional Neural Network:

1. mini_batch_x, mini_batch_y = get_mini_batch(im_train, label_train, batch_size)

   Training images are separated into small batches. The whole data samples are divided into (length / batch_size) parts, and images are taken from these parts (one part one image) to form a mini batch.

2. y = fc(x, w, b)

   Simply, y = X * W + b.

3. dl_dx, dl_dw, dl_db = fc_backward(dl_dy, x, w, b, y)

   With the back-propagation formula, dl_dx = dl_dy * w, dl_dw = dl_dy * x, dl_db = dl_dy.

4. l, dl_dy = loss_euclidean(y_tilde, y)

   The l is the Euclidean distance of y_tilde to y, dl_dy is the derivative of l.

5. w, b = train_slp_linear(mini_batch_x, mini_batch_y)

   In this Single-layer Linear Perceptron, the learning rate is 0.016 while the decay rate is 0.5 and the iteration time for the training is 5000. The accuracy obtained is 58.2% (**FIG.1**).

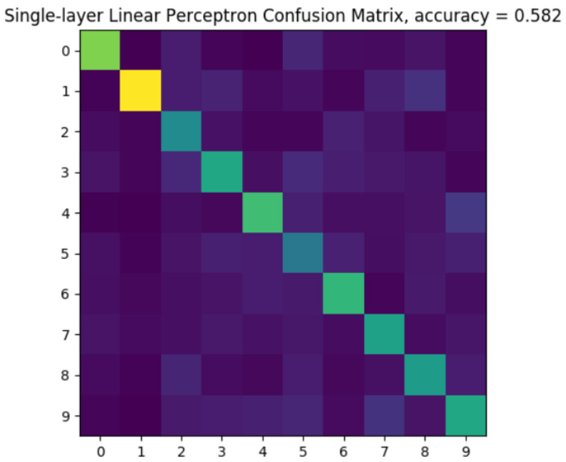

FIG.1 Single-layer Linear Perceptron

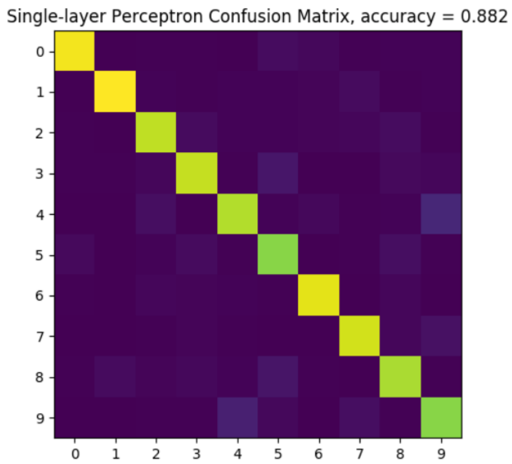

FIG.2 Single-layer Perceptron

6. l, dl_dy = loss_cross_entropy_softmax(x, y)

   Loss cross entropy with softmax is defined as the -1 * the sum of [y * ln(y_prediction)], and dl_dy in this cross entropy is y_prediction – y.

7. w, b = train_slp(mini_batch_x, mini_batch_y)

   In this Single-layer Perceptron, the learning rate is 0.32 while the decay rate is 0.5 and the iteration time for the training is 5000. The accuracy obtained is 88.2% (**FIG.2**), higher than Single-layer Linear Perceptron.

8. y = relu(x)

   In this function, e, 0.01, is set as the leaky ReLU parameter. Leaky ReLU performs better than ReLU in CNN.

9. dl_dx = relu_backward(dl_dy, x, y)

   When using leaky ReLU, the derivation is 1 when y >=0 and e when y < 0(e = 0.01 in the code).

10. w1, b1, w2, b2 = train_mlp(mini_batch_x, mini_batch_y)

    In this Multi-layer Perceptron, the learning rate is 0.64 while the decay rate is 0.9 and the iteration time for the training is 5000. The accuracy obtained is 91.3% (**FIG.3**).

11. im_col = im2col(im, f_h, f_w, stride=1, pad=1)

    I defined a function im2col. For the input, im is the image, f_h and f_w is the size of the filter, stride is the stride for the convolution, pad is the size of the zeros around the original images. Im2col function can help increase the speed of the CNN training.

12. y = conv(x, w_conv, b_conv)

    With the input x, x_new is obtained with im2col(x…), w_conv is reshaped to fit the size of x_new and y is obtained by the equation y = w_conv * x_new + b.

13. dl_dw, dl_db = conv_backward(dl_dy, x, w_conv, b_conv, y)

With the reshaping and im2col methods, dl_dw = dl_dy * x, dl_db = dl_dy.

14. y = pool2x2(x)

With a 2x2 window and stride 2, the largest pixel is kept in the window.

15. dl_dx = pool2x2_backward(dl_dy, x, y)

This function is to move pixels in dl_dy into corresponding dl_dx coordinates, it's the reverse method of max pooling.

16. y = flattening(x)

Flatten x into a long single line vector.

17. dl_dx = flattening_backward(dl_dy, x, y)

Reverse what we did in function flattening.

18. w_conv, b_conv, w_fc, b_fc = train_cnn(mini_batch_x, mini_batch_y)

In CNN, the learning rate is 3.2 while the decay rate is 0.8 and the iteration time for the training is 12500. The accuracy obtained is 92.2% (**FIG.4**). In my case, the training speed of CNN is slow, so once I got good W and B, I saved them and put them into the model again with other learning rates or decay rates to expect a higher accuracy. The processes are shown in TABLE.1 below.

TABLE.1 Processes for increasing the accuracy of CNN prediction

	Learning rate	**Decay rate**	**iterations**	**Iterations to decay**	**Accuracy obtained**
1	3.2	0.8	10000	1000	0.916
2	3.2	0.8	1000	100	0.917
3	3.2	0.5	500	100	0.919
4	3.2	0.1	500	100	0.920
5	3.2	0.1	500	100	0.922

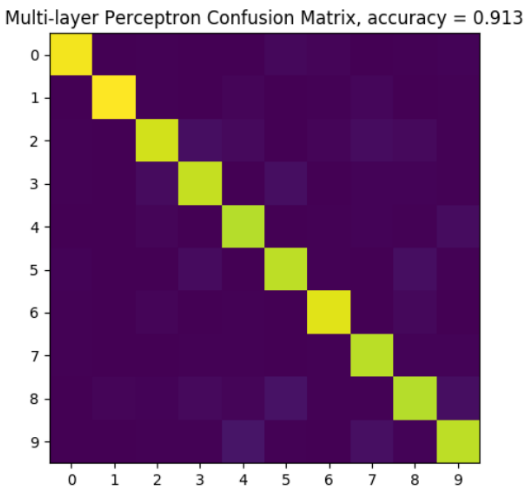

FIG.3 Multi-layer Perceptron

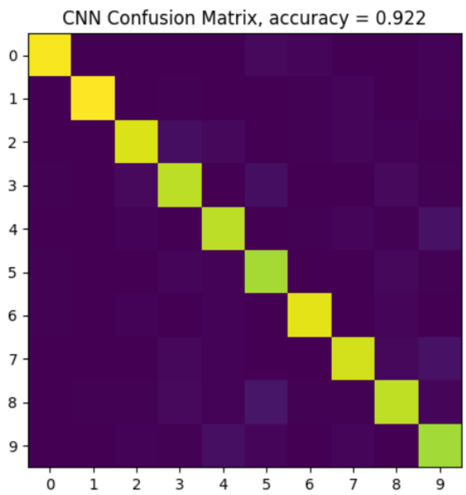

FIG.4 Convolutional Neural Network