Neural Networks
Implementing a Neural Network from scratch

Anne-Claire Fouchier, Jingwen Yang

I. INTRODUCTION

The aim of this lab is to build neural networks for classification from scratch in Python with numpy, and evaluate them.

II. DATASET

The dataset comes from the MNIST database of handwritten digits. The MNIST dataset is one of the most used datasets in machine learning research. It consists of 70000 grayscale images of handwritten digits (of size 28x28), divided into a training set of 60,000 examples and a test set of 10,000 examples. It is a subset of a larger set available from NIST. The digits have been size-normalized and centered in a fixed-size image[1].

Fig. 1. A few samples from the MNIST test dataset by Josef Steppan

III. IMPLEMENTATION

There are three python files for this assignment.

- $lab2\_1.py$ The default settings is 500 epochs and learning rate 1. The user should select method 1.
- $lab2\_2.py$ The input argument should be the iteration number. The default settings is 500 epochs.
- $lab2\_3.py$ The input arguments are number of iterations and learning rate. The default settings are 500 epochs with learning rate of 0.7.

IV. METHODS

The first two tasks consist on implementing binary classification. For both two tasks, the final Layer uses a Sigmoid Activation function in equation 1 giving an output between 0-1, and the Cost function is the Binary Cross Entropy Error Function in equation 2.

[1]http://yann.lecun.com/exdb/mnist/

TABLE I

FORWARD PROPOGATION VARIABLES

Variable	Represent	Size
X	Training set	(784, 40000)
W	weights	(1, 784)
b	bias	(1, 1)
Z	output before activation function	(1, 40000)
A	output after activation function	(1, 40000)

$$\sigma(x) = \frac{1}{1 + e^{-}x} \tag{1}$$

$$E_{\text{entropy}} = -\sum_{1}^{N} [y_n \log a_n + (1 - y_n) \log(1 - a_n)] \tag{2}$$

A. Data division

The training set of 60,000 samples is further split into a training set and a validation set. This is done in the training function, as well as shuffling the whole dataset and two subsets before each iteration starts, so that during the training process, the network's performance can be validated on the validation set. Also, a smaller training set is good to avoid over-fitting.

B. Single neuron model

The first neural network implemented is a single neuron network. It needs a weight matrix of size (1,784), 784 being the size of each input sample (of 28x28 pixels) and a bias number (1,1). W is initialized as a normal distribution and b as 0. The model architecture can be seen in figure 2. The W_i indicates the ith weight

1) Forward propagation: Forward propagation consists on feeding the input forward through the network. As there is only one single neuron, it simply consists here on applying equations 24 (preactivation) and 25 (activation, being here the sigmoid function).

$$Z = W \cdot X + b \tag{3}$$

$$A = \sigma(Z) \tag{4}$$

2) Backward propagation: The aim of the training process is to learn the weights and bias of each layer using gradient descent. They are computed following equations 11 and 12.

$$dW = \frac{dJ}{dW} = \frac{dJ}{d2} * \frac{dA}{dZ} * \frac{dZ}{dW} \tag{5}$$

$$db = \frac{dJ}{db} = \frac{dJ}{dA} * \frac{dA}{dZ} * \frac{dZ}{db} \tag{6}$$

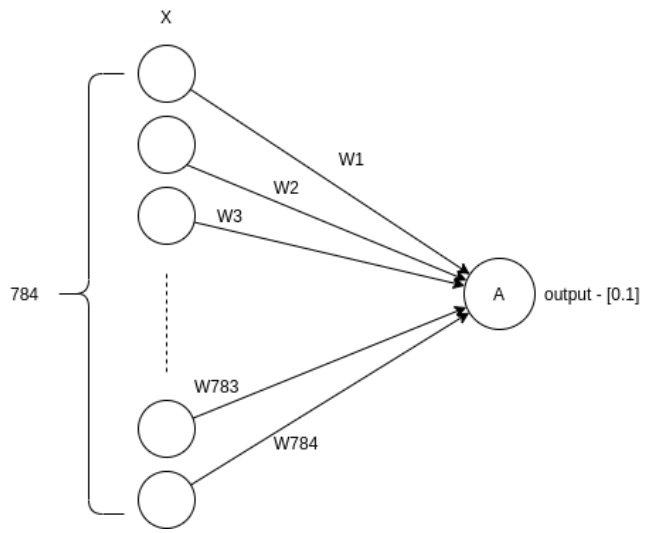


Fig. 2. Single neuron model architecture

C. Model with one hidden layer

In this task, a hidden layer with 64 neurons is introduced to the network.

1) Initialization: First, the parameters w1, w2, b1 and b2 are initialized with random numbers. The initial weights have to be small enough so that they are not too far away from the destination. w1 and b1 correspond to the weight and bias of the hidden layer, w2 and b2 correspond to the weight and bias of the output layer. The architecture of a neural network with one hidden fully-connected layer is shown in figure 3

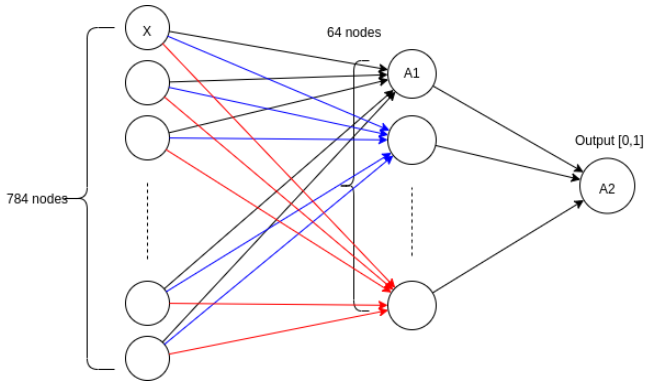


Fig. 3. One hidden layer model architecture

2) Forward propagation: As before, for each epoch, the first step is forward propagation. The basic mechanism is shown in equations 24,25,26,27. The input is propagated through the hidden layer, as seen in equations 24 and 25. The output of the hidden layer is then fed to the output layer, like in equations 26 and 27. For both two layers we are using sigmoid as the activation function.

$$Z_1 = W_1 \cdot X + b_1 \tag{7}$$

TABLE II

FORWARD PROPAGATION VARIABLES

Variable	Represent	Size
X	Training set	(784, 40000)
W_1	first layer weights	(64, 784)
b_1	first layer bias	(64, 1)
Z_1	first layer output before activation function	(64, 40000)
A_1	first layer output after activation function	(64, 40000)
W_2	second layer weights	(1, 64)
b_2	second layer bias	(1, 1)
Z_2	second layer output before activation function	(1, 40000)
A_2	second layer output after activation function	(1, 40000)

$$A_1 = \sigma(Z_1) \tag{8}$$

$$Z_2 = W_2 \cdot A_1 + b_2 \tag{9}$$

$$A_2 = \sigma(Z_2) \tag{10}$$

3) Backward propagation: In the back propagation we have to calculate the derivatives of the loss function with respects to four parameters w1, w2, b1, b2. The computation of w2, b2 follow the equations 11, 12.

$$dW_2 = \frac{dJ}{dW_2} = \frac{dJ}{dA_2} * \frac{dA_2}{dZ_2} * \frac{dZ_2}{dW_2} \tag{11}$$

$$db_2 = \frac{dJ}{db_2} = \frac{dJ}{dA_2} * \frac{dA_2}{dZ_2} * \frac{dZ_2}{db_2} \tag{12}$$

Since they have some terms in common, we write down δ in equation 13 to reduce some calculation.

$$\delta_2 = \frac{dJ}{dA_2} * \frac{dA_2}{dZ_2} = -[\frac{y}{A_2} - \frac{1-y}{1-A_2}] * \sigma'(Z_2) \tag{13}$$

Replacing the two common terms with δ_2, the derivatives with respect to w1, b1 are calculated in equations 14 and 15.

$$dW_2 = \delta_2 * \frac{dZ_2}{dW_2} = \delta_2 * A_1 \tag{14}$$

$$db_2 = \delta_2 * \frac{dZ_2}{db_2} = \delta_2 \tag{15}$$

We do the same operations to w1 ,b1, but with one more layer to differentiate.

$$dW_1 = \frac{dJ}{dW_1} = \frac{dJ}{dA_2} * \frac{dA_2}{dZ_2} * \frac{dZ_2}{dA_1} * \frac{dA_1}{dZ_1} * \frac{dZ_1}{dW_1} \tag{16}$$

$$db_1 = \frac{dJ}{db_1} = \frac{dJ}{dA_2} * \frac{dA_2}{dZ_2} * \frac{dZ_2}{dA_1} * \frac{dA_1}{dZ_1} * \frac{dZ_1}{db_1} \tag{17}$$

$$\delta_1 = \frac{dJ}{dA_2} * \frac{dA_2}{dZ_2} * \frac{dZ_2}{dA_1} * \frac{dA_1}{dZ_1} = -[\frac{y}{A_2} - \frac{1-y}{1-A_2}] * \sigma'(Z_2) * W_2 * \sigma'(Z_1) \tag{18}$$

$$dW_1 = \delta_1 * \frac{dZ_1}{dW_1} = \delta_1 * X \tag{19}$$

$$db_1 = \delta_1 * \frac{dZ_1}{db_1} = \delta_1 \tag{20}$$

4) Updating parameters: With the gradient descent method, we can update our parameters each iteration as in equation 21,α being the learning rate.

$$W_i = W_i - \alpha * dW_i, i = 1, 2 \tag{21}$$

$$b_i = b_i - \alpha * db_i, i = 1, 2 \tag{22}$$

D. Multi-class network

The previous tasks handle binary classification. In this task, the previous network is extended to handle multi-class classification. The output layer now has 10 neurons. The activation function in the last layer is replaced by softmax as equation 23. The architexture of the multi-class neural network is shown in figure 4.

$$softmax(Z_i) = \frac{exp(Z_i)}{\sum_{j=1}^{N} exp(Z_j))} \tag{23}$$

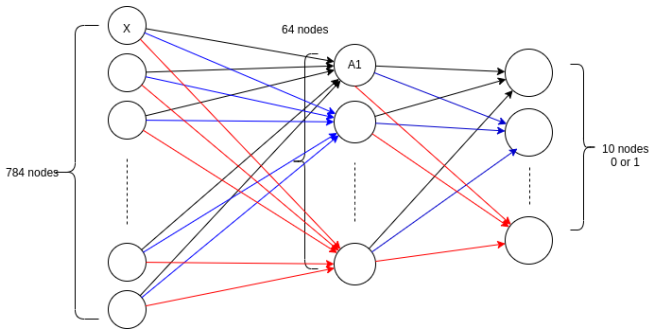


Fig. 4. One hidden layer multi-class model architecture

1) Initialization: First, the parameters w1, w2, b1 and b2 are initialized with random numbers. Note here the size of w2,b2 should change to (10,64) and (10,1) respectively because we want to have the output of size(10,40000). The other variables remain the same.

2) Forward propagation: In this case, we use sigmoid as activation function for the first layer, and softmax for the second layer. The loss function will replaced by general cross-entropy instead of the binary one. The other parts remain the same with the previous task.

$$Z_1 = W_1 \cdot X + b_1 \tag{24}$$

$$A_1 = \sigma(Z_1) \tag{25}$$

$$Z_2 = W_2 \cdot A_1 + b_2 \tag{26}$$

$$A_2 = Softmax(Z_2) \tag{27}$$

3) Backward propagation: The general structure is the same with the previous task. We only need to change the previous derivatives of binary cross-entropy loss and sigmoid activation function to the general cross entropy function and softmax function with respect to each variables. Since it is easier to calculate the derivatives of these two terms together, we write down δ in equation 28 to reduce some calculation. The detailed calculation can be found on the link here[2].

$$\delta_2 = \frac{dJ}{dA_2} * \frac{dA_2}{dZ_2} = A_2 - Y \tag{28}$$

V. RESULTS AND ANALYSIS

A. Evaluation measures

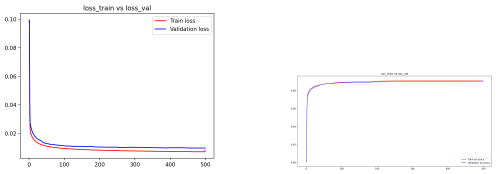
The loss and the accuracy on both the training and validation data are evaluated here. Here we are mainly tuning the learning rate to see its influence on the neural network performance. The other parameters settings are shown in table III.

TABLE III
PARAMETERS TUNING

Variable	chosen values
epochs	500
training set size	40000
learning rate	0.3, 0.7, 1.1, 2.0

B. Single neuron model

The baseline for this analysis is the binary classification of 0 and others, and the learning rate equal to 1. In this case, there is convergence around 200 epochs; the accuracy on the validation test reaches 0.988 at epoch 74, and 0.99 at epoch 317. After 500 epochs, the test accuracy is 0.9919. The validation accuracy is below the training accuracy after about 100 epochs, but it is steady (Figure 5(a)).



(a) loss (b) accuracy

Fig. 5. Single neuron 0 vs. all, lr=1

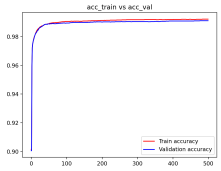
Changing the learning rate causes changes in the time of convergence and the final test accuracy. With a lower learning rate, here 0.5 (Figure 6(c)), the slopes start distinguishing themselves late, around 300 epochs. The test accuracy is 0.9915. With a higher learning rate (Figures 6(a), 6(b)), the final test accuracy is a little bit higher, 0.9923 with lr=2 and 0.9925 with lr=5. However, the convergence

[2]http://www.adeveloperdiary.com/data-science/deep-learning/neural-network-with-softmax-in-python/?fbclid=IwAR3UAjb4Hpv8UkBbuG8sFack-cCFp6ly8XfgY4h8JjJqVZUlFG63W8eyvR8

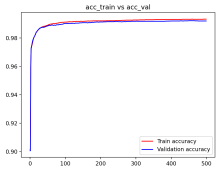
is faster with lr=2, around epoch 60.

Finally, not all number are classified equivalently. With a binary classification 5 vs. all and lr=1, the slopes are not smooth until around epoch 50 (Figure 6(d)). Also, the final test accuracy is lower than with 0 vs all and lr=1. It is in fact 0.9736. Also, convergence starts around epoch 200.

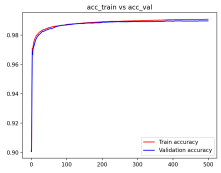
Note that in this case, the implemented loss function is the mean square error.



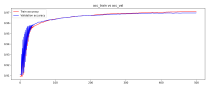
(a) accuracy, lr=2, 0 vs. all



(b) accuracy, lr=5, 0 vs. all



(c) accuracy, lr=0.5, 0 vs. all



(d) accuracy, lr=1, 5 vs. all

Fig. 6. Accuracy graphs for various learning rates and Single neuron tasks

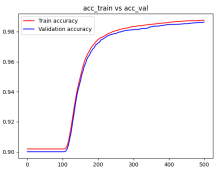
C. Model with one hidden layer

As shown in figures 7(a), 7(b), a small learning rate is not converging fast in this case. The loss for validation set keeps staying a plateau at around first 100 epochs. With a learning rate of 0.7, it starts to converge at around epoch 70. In both scenarios, the training loss of the training set starts decreasing a lot earlier than the validation loss, meaning it starts overfitting. With a learning rate of 1.1, it starts to converge before epoch 50 and before epoch 20 for the learning rate of 2.0. For the learning rates we have tested on, the validation losses all converge to around 0.25, meanwhile the training losses converge almost at 0.

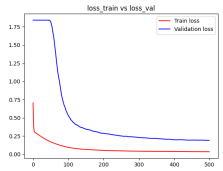
As for the accuracy, the first 100 epochs with a learning rate of 0.3 stay in a plateau because the gradient descent is relatively slow, the changes in outputs for the last layer are always lower than certain threshold such as 0.5. When we calculate the accuracy,we round the output to be 0 or 1 to have binary classification. Through debugging all the predictions are classified as 0 during the first 100 epochs. Therefore the accuracy remains the same. The parameters updates weren't big enough to make a difference on the accuracy.

From the figure 7(f),7(d) we can see the accuracy curve of learning rate 1.1 start to increase earlier than the curve with a learning rate of 0.7, approximately epoch 80 and epoch 30 respectively. However, with a learning rate of 2.0, the accuracy starts increasing at around epoch 10. The final accuracy they achieve are very similar, at more than 98%.

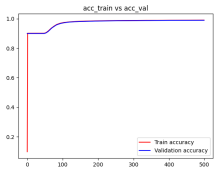
(a) loss, lr=0.3



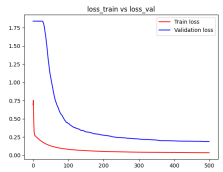
(b) accuracy, lr=0.3



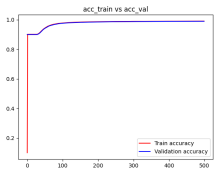
(c) loss, lr=0.7



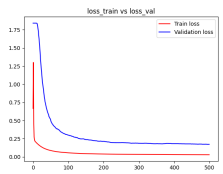
(d) accuracy, lr=0.7



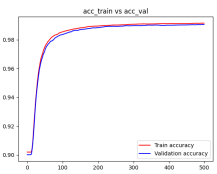
(e) loss, lr=1.1



(f) accuracy, lr=1.1



(g) loss, lr=2.0



(h) accuracy, lr=2.0

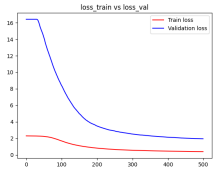
Fig. 7. loss, accuracy graphs, task2

D. Multi-class network

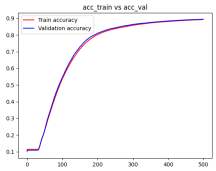
As shown in figure 8(a),8(c),8(b),8(d), we can say with a slightly higher learning rate the loss and accuracy converge much faster. They achieve an accuracy of 92% and 94% on the test dataset. As in figure 8(e),8(f),8(g) and 8(h), there is an oscillation phenomenon. That's probably caused by the big learning rate. When we update the parameters, it jumps too far from the ideal point and make a temporary high loss/ low accuracy. However, with the learning rate of 2.0 we achieve the accuracy of 99% on the test dataset, which is way better than the 92% from the learning rate 0.3. We assume the reason behind it is a small learning rate can be stuck in a local minima and it cannot jump away from it.

VI. CONCLUSION

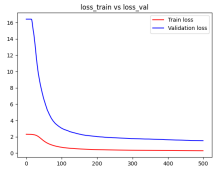
The single neuron networks performs well for binary classification. For the same task, introducing a hidden layer leads to an earlier convergence. The learning rate has an important role in the performance. However, it is important to take into account that not all labels perform similarly. Finally, the importance of the learning rate is highlighted in the more complex task of multi-class classification.



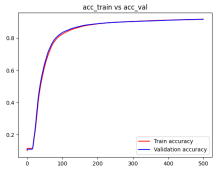
(a) loss, lr=0.3



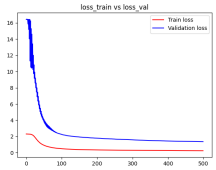
(b) accuracy, lr=0.3



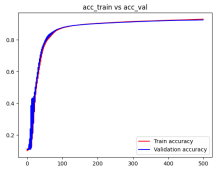
(c) loss, lr=0.7



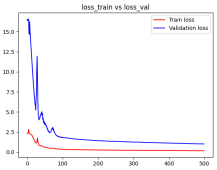
(d) accuracy, lr=0.7



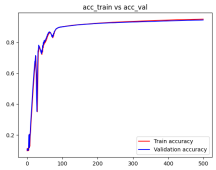
(e) loss, lr=1.1



(f) accuracy, lr=1.1



(g) loss, lr=2.0



(h) accuracy, lr=2.0

Fig. 8. loss, accuracy graphs, task3