ARTIFICIAL NEURAL NETWORK
HOMEWORK 1 REPORT

Jiahao Li
Department of Computer Science
Tsinghua University
lijiahao17@mails.tsinghua.edu.cn

1 Mechanism

1.1 Back Propagation Algorithm

The back propagation (BP) algorithm, an efficient method to update model weights, mainly relies on the backward gradient flow. To train an effective multi-layer perceptron (MLP) model with BP algorithm, the gradients of all module should be properly defined and correctly computed. A general MLP is depicted in Figure 1.

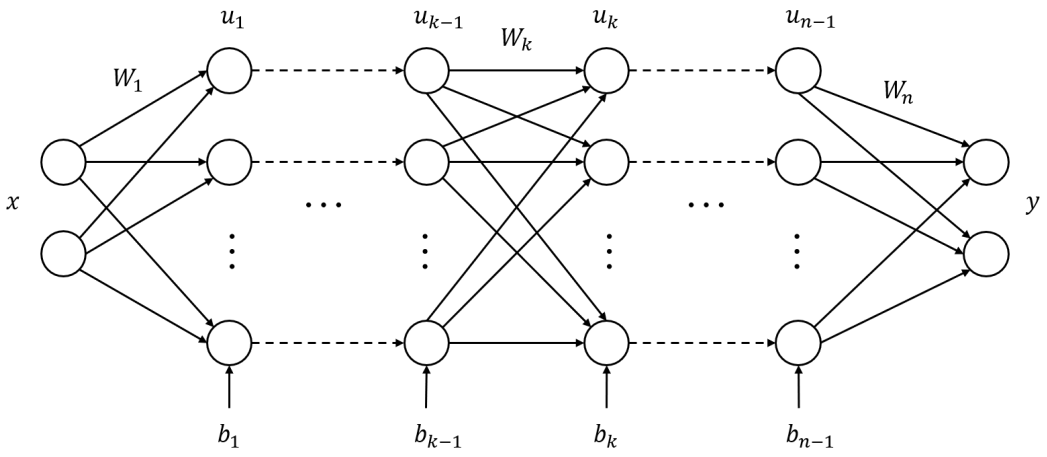


Figure 1: A General MLP

The gradients of all weights and biases are computed as Equation (1) written in tensor format.

$$\frac{\partial y}{\partial W_k} = \frac{\partial y}{\partial u_k}\frac{\partial u_k}{\partial W_k}, \quad \frac{\partial y}{\partial b_k} = \frac{\partial y}{\partial u_k}\frac{\partial u_k}{\partial b_k} \tag{1}$$

Denote y as the model output, u_k as an arbitrary hidden layer, and W_k, b_k as trainable weight and bias of k-th layer, separately. The local gradient $\partial y/\partial u_k$ could be recursively computed for each layer k, as Equation (2).

$$\frac{\partial y}{\partial u_k} = \frac{\partial y}{\partial u_{k+1}}\frac{\partial u_{k+1}}{\partial u_k}, \quad k = 1, 2, \cdots, n-1 \tag{2}$$

Thus, to implement a module, the only thing is to define the gradients of the module itself. Then the global gradients are computed by multiplying local gradients by $\partial y/\partial u_k$, which could be computed recursively. Given a learning rate

η, along with the backward gradient of every trainable weight $\partial E/\partial W$, the BP algorithm updates the weight W as Equation (3).

$$W \leftarrow W - \eta \frac{\partial E}{\partial W} \tag{3}$$

The local gradients of different layers are shown in following sections.

1.2 Linear Layer

The output is computed as Equation (4).

$$y = xW + b \tag{4}$$

Denote W as the weights, and b as the biases. The local gradients are computed as Equation (5)

$$\frac{\partial y}{\partial x} = W^T, \quad \frac{\partial y}{\partial W} = x^T, \quad \frac{\partial y}{\partial b} = 1 \tag{5}$$

1.3 Relu Activation

The output is computed as Equation (6).

$$y = \text{ReLU}(x) = \begin{cases} 0, & x \leq 0 \\ x, & x > 0 \end{cases} \tag{6}$$

The local gradients are computed as Equation (7)

$$\frac{\partial y}{\partial x} = \begin{cases} 0, & x \leq 0 \\ 1, & x > 0 \end{cases} \tag{7}$$

1.4 Sigmoid Activation

The output is computed as Equation (8).

$$y = \sigma(x) = \frac{1}{1 + e^x} \tag{8}$$

The local gradients are computed as Equation (9)

$$\frac{\partial y}{\partial x} = \frac{1}{1 + e^x} \cdot \frac{e^x}{1 + e^x} = \sigma(x) \cdot (1 - \sigma(x)) \tag{9}$$

1.5 Euclidean Loss Function

The output is computed as Equation (10).

$$y = \frac{1}{2N} \sum_{n=1}^{N} ||x_n - t_n||_2^2 \tag{10}$$

Denote x_n as the predicted probabilities of all categories, t_n as the ground truth label, and N as the batch size. The local gradients are computed as Equation (11)

$$\frac{\partial y}{\partial x_n} = \frac{1}{N}(x_n - t_n) \tag{11}$$

1.6 Softmax Cross Entropy Loss Function

The output is computed as Equation (12).

$$y = -\frac{1}{N}\sum_{n=1}^{N} \ln \frac{\exp\left(x_c^{(n)}\right)}{\sum_{k=1}^{K} \exp\left(x_k^{(n)}\right)} \tag{12}$$

Denote N as the batch size, K as the number of categories and c as the ground truth class. The local gradients are computed as Equation (13)

$$\frac{\partial y}{\partial x_k^{(n)}} = \frac{1}{N} \cdot \left(\frac{\exp\left(x_k^{(n)}\right)}{\sum_{k=1}^{K} \exp\left(x_k^{(n)}\right)} - \delta_{kc} \right) \tag{13}$$

where δ_{kc} is an indicator function, i.e.

$$\delta_{kc} = \begin{cases} 0, & k \neq c \\ 1, & k = c \end{cases}$$

2 Network Overview

It is important to note that the softmax cross entropy loss function (SCE) has a Softmax activation before computing cross entropy loss, while the Euclidean loss function (MSE) does not, a situation that an adaptive last layer is required. In all networks below, a Sigmoid layer is added as the last layer with Euclidean loss function, and removed otherwise.

2.1 No-hidden-layer Network

No-hidden-layer network with Euclidean loss is briefly described in Table 1.

Layer	Type	In	Out
1	Linear	784	10
2	Sigmoid	256	256

Table 1: No-hidden-layer Network with Euclidean Loss

No-hidden-layer network with softmax cross entropy loss is briefly described in Table 2.

Layer	Type	In	Out
1	Linear	784	10

Table 2: No-hidden-layer Network with Softmax Cross Entropy Loss

2.2 One-hidden-layer Network

One-hidden-layer network with Sigmoid and Euclidean loss is briefly described in Table 3.

Layer	Type	In	Out
1	Linear	784	256
2	Sigmoid	256	256
3	Linear	256	10
4	Sigmoid	10	10

Table 3: One-hidden-layer Network with Sigmoid and Euclidean Loss

One-hidden-layer network with Relu and Euclidean loss is briefly described in Table 4.

Layer	Type	In	Out
1	Linear	784	256
2	Relu	256	256
3	Linear	256	10
4	Sigmoid	10	10

Table 4: One-hidden-layer Network with Relu and Euclidean Loss

One-hidden-layer network with Sigmoid and softmax cross entropy loss is briefly described in Table 5.

Layer	Type	In	Out
1	Linear	784	256
2	Sigmoid	256	256
3	Linear	256	10

Table 5: One-hidden-layer Network with Sigmoid and Softmax Cross Entropy Loss

One-hidden-layer network with Relu and softmax cross entropy loss is briefly described in Table 6.

Layer	Type	In	Out
1	Linear	784	256
2	Relu	256	256
3	Linear	256	10

Table 6: One-hidden-layer Network with Relu and Softmax Cross Entropy Loss

2.3 Two-hidden-layer Network

Two-hidden-layer network with Sigmoid and Euclidean loss is briefly described in Table 7.

Layer	Type	In	Out
1	Linear	784	256
2	Sigmoid	256	256
3	Linear	256	128
4	Sigmoid	128	128
5	Linear	128	10
6	Sigmoid	10	10

Table 7: Two-hidden-layer Network with Sigmoid and Euclidean Loss

Two-hidden-layer network with Relu and Euclidean loss is briefly described in Table 8.

Layer	Type	In	Out
1	Linear	784	256
2	Relu	256	256
3	Linear	256	128
4	Relu	128	128
5	Linear	128	10
6	Sigmoid	10	10

Table 8: Two-hidden-layer Network with Relu and Euclidean Loss

Two-hidden-layer network with Sigmoid and softmax cross entropy loss is briefly described in Table 9.

Layer	Type	In	Out
1	Linear	784	256
2	Sigmoid	256	256
3	Linear	256	128
4	Sigmoid	128	10
5	Linear	10	10

Table 9: Two-hidden-layer Network with Sigmoid and Softmax Cross Entropy Loss

Two-hidden-layer network with Relu with softmax cross entropy loss is briefly described in Table 10.

Layer	Type	In	Out
1	Linear	784	256
2	Relu	256	256
3	Linear	256	128
4	Relu	128	128
5	Linear	128	10

Table 10: Two-hidden-layer Network with Relu with Softmax Cross Entropy Loss

3 Experiments

3.1 Experiments on MNIST

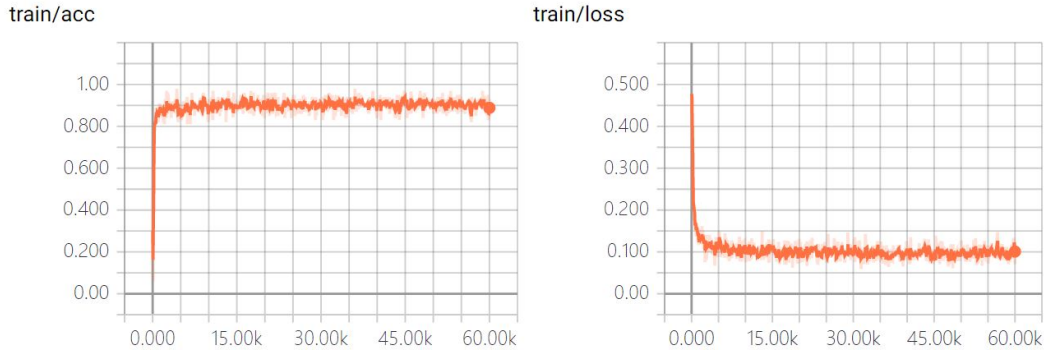
To control variables, all the experiments in this section are conducted with the same settings unless otherwise specified. MNIST serves as the only dataset for training and testing. The optimizer is initialized with learning rate of 0.01, momentum of 0.9, weight decay of 0.0005, the same as the default configuration of SGD optimizer in popular deep learning frameworks. The model is trained with batch size of 100 for 100 epochs, and tested after training. The experimental results are shown in Table 11.

Model	Train Time(s)	Train Loss	Train Accuracy(%)	Test Loss	Test Accuracy(%)
0-layer + MSE	168	0.0968	90.52	0.0926	91.09
0-layer + SCE	**161**	0.2732	92.32	0.2732	92.34
1-layer + Sigmoid + MSE	706	0.0913	91.13	0.0857	91.80
1-layer + Relu + MSE	666	0.0351	97.10	0.0357	97.04
1-layer + Sigmoid + SCE	723	0.1326	96.67	0.1336	96.49
1-layer + Relu + SCE	688	0.0350	99.34	0.0621	98.18
2-layer + Sigmoid + MSE (std 0.1)	814	0.0970	90.69	0.0911	91.34
2-layer + Relu + MSE	796	0.0222	98.23	**0.0243**	97.84
2-layer + Sigmoid + SCE	813	0.1219	96.88	0.1191	96.79
2-layer + Relu + SCE	781	**0.0174**	**99.72**	0.0541	**98.34**

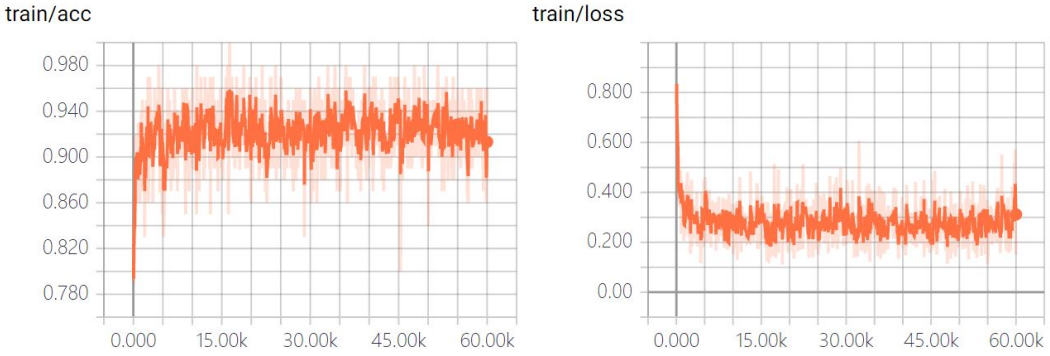
Table 11: Experimental Results on MNIST

It is worthy to mention that the two-hidden-layers network with Sigmoid activation and Euclidean loss performs poorly with the default configuration due to the vanishing gradients of Sigmoid. However, setting the initial std of Linear layers to a higher value, 0.1 for example, would solve this problem.

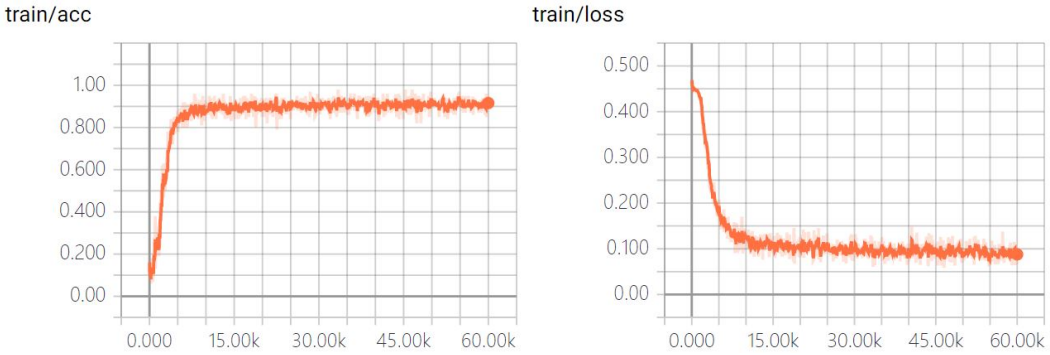
To display the results in nice-looking style, the graphs of loss and accuracy against iteration of all experiments are plotted separately with third-party library `tensorboardX`, as shown in Figure 2.



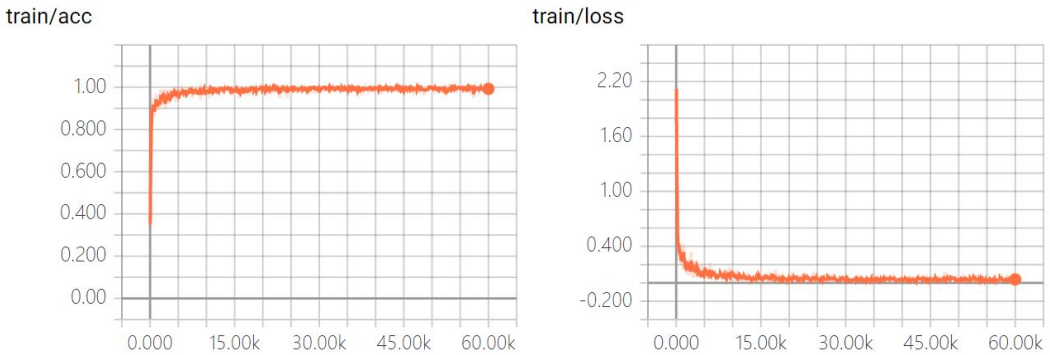
(a) 0-layer + MSE



(b) 0-layer + SCE

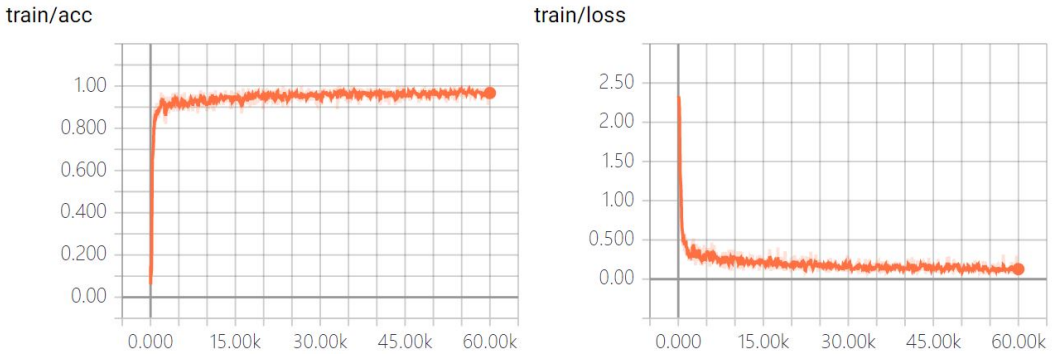


(c) 1-layer + Sigmoid + MSE

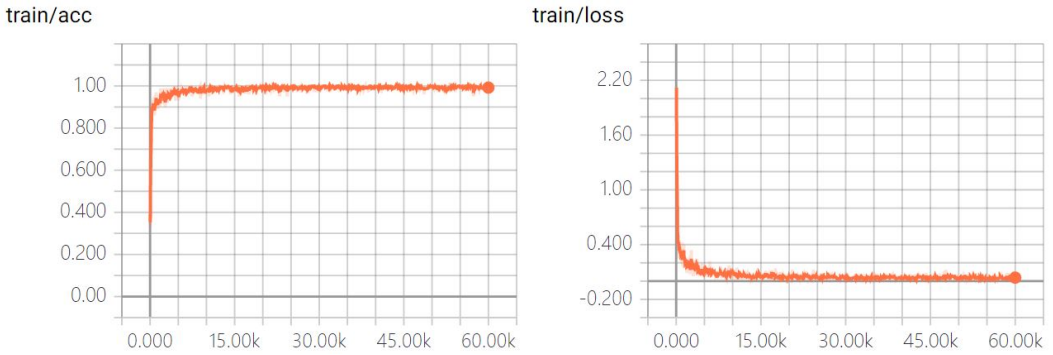


(d) 1-layer + Relu + MSE

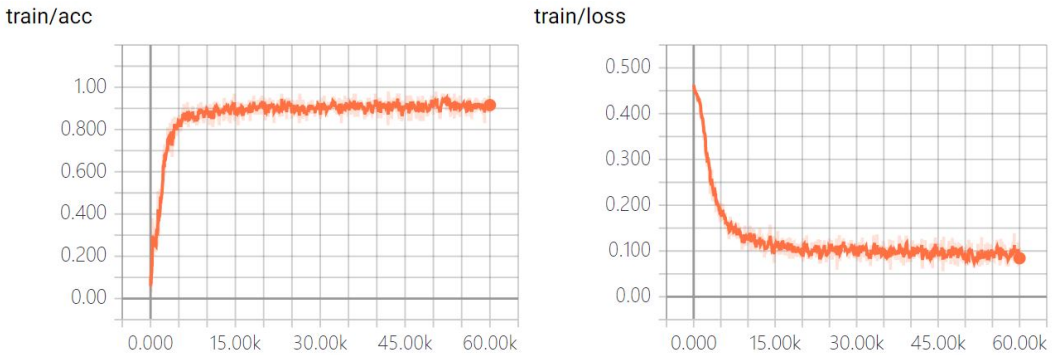
Figure 2: Loss and Accuracy against Iteration



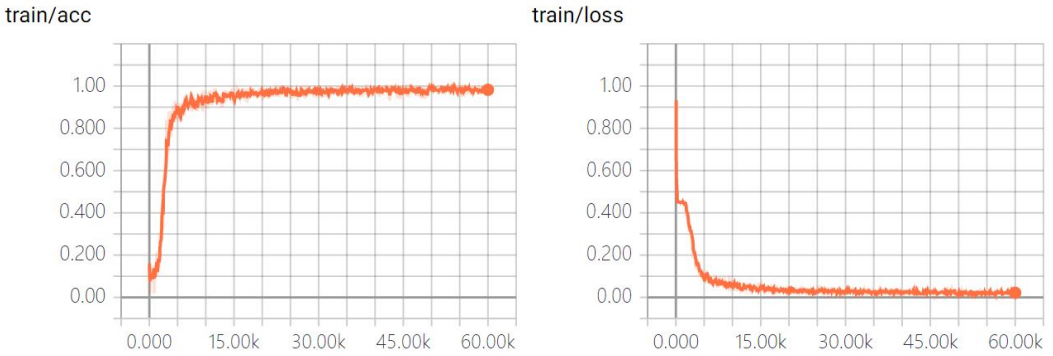
(a) 1-layer + Sigmoid + SCE



(b) 1-layer + Relu + SCE

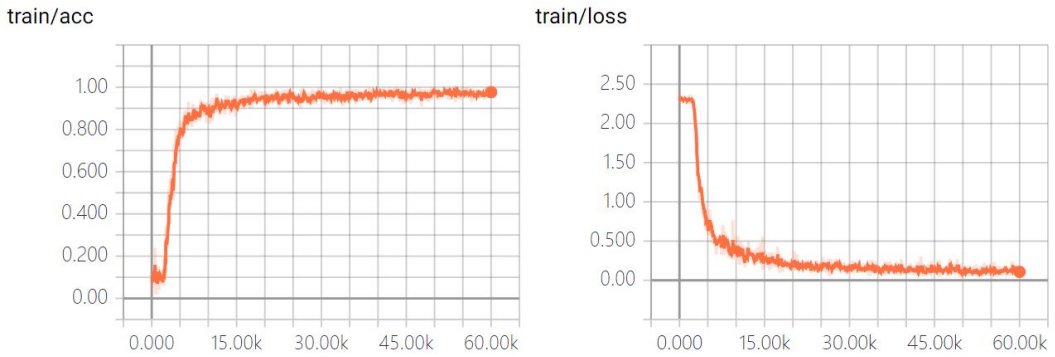


(c) 2-layers + Sigmoid + MSE

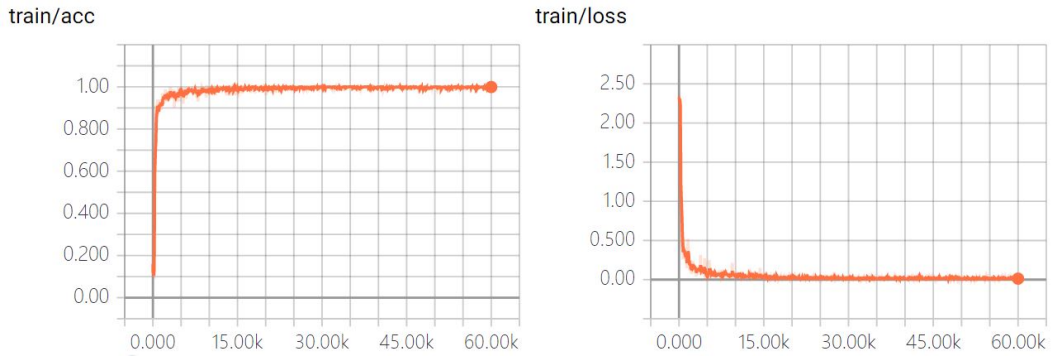


(d) 2-layers + Relu + MSE

Figure 2: Loss and Accuracy against Iteration (continued)



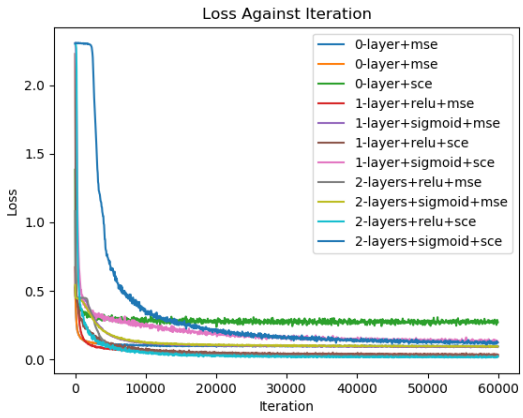
(e) 2-layers + Sigmoid + SCE



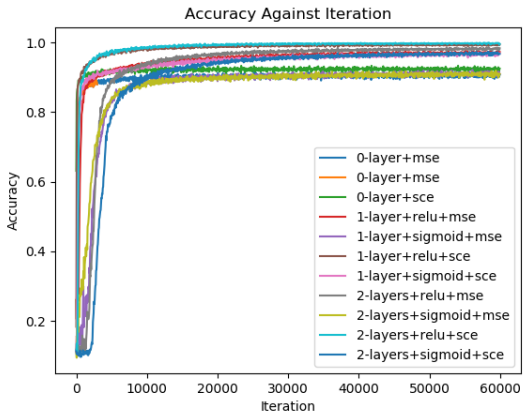
(f) 2-layers + Relu + SCE

Figure 2: Loss and Accuracy against Iteration (continued)

To better analyze the performances of different models, the down-sampled loss and accuracy results are plotted together in a single graph as shown in Figure 3.



(g) Training Loss against Iteration



(h) Training Accuracy against Iteration

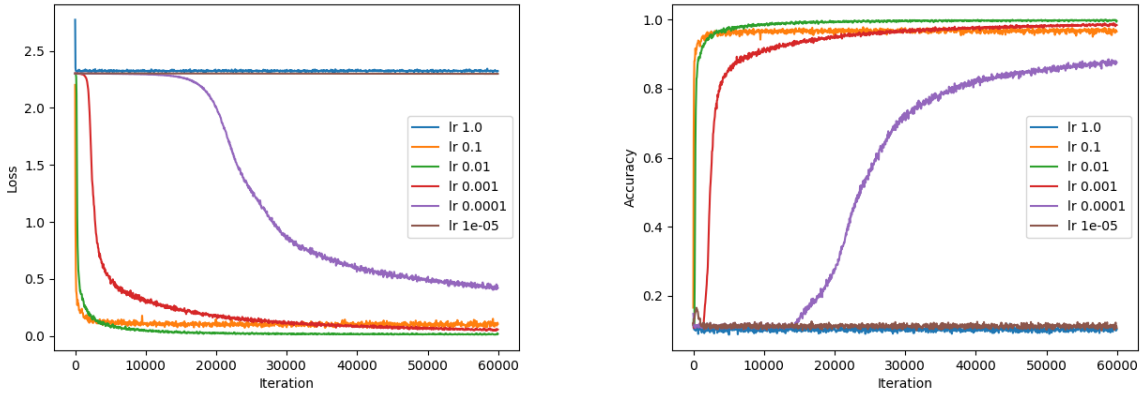
Figure 3: Training Process of Different Models

3.2 Hyper Parameter Fine-tuning

In the following experiments, hyper parameters including learning rate, momentum and batch size would be adjusted for better performance. All experiments are conducted on a two-hidden-layer network with Relu and SCE with the same configuration stated in Section 3.1, except for the hyper parameter being adjusted.

3.2.1 Learning Rate

Learning rate is an essential factor in model training, since it controls the scale of weight update. In this experiment, the optimal learning rate will be explored from 1e-5 to 1.0 logarithmically. The training process with different learning rates is shown in Figure 4.



(a) Training Loss against Iteration

(b) Training Accuracy against Iteration

Figure 4: Training Process with Different Learning Rates

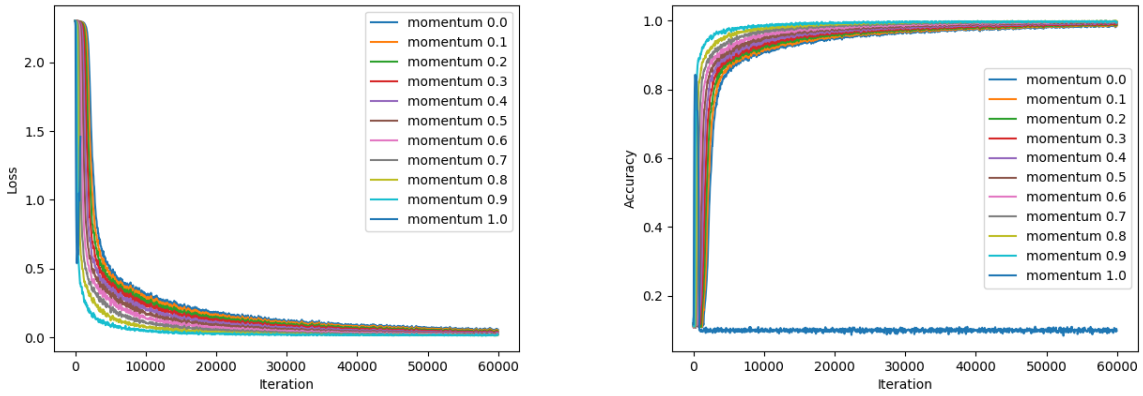
The experimental results are shown in Table 12.

Learning Rate	Train Loss	Train Accuracy(%)	Test Loss	Test Accuracy(%)
1.0	2.3244	0.1025	2.3037	0.1135
0.1	0.1023	0.9674	0.0853	0.9742
0.01	**0.0174**	**0.9972**	**0.0541**	**0.9834**
0.001	0.0792	0.9782	0.0864	0.9738
0.0001	0.5686	0.8287	0.4174	0.8769
0.00001	2.2999	0.1124	2.2988	0.1606

Table 12: Experimental Results with Different Learning Rates

3.2.2 Momentum

A proper momentum setting could effectively accelerate the training process and avoid being trapped in a local minima. In this experiment, the optimal momentum will be explored from 0.0 to 1.0 by step of 0.1. The training process with different momentums is shown in Figure 5.



(a) Training Loss against Iteration

(b) Training Accuracy against Iteration

Figure 5: Training Process with Different Momentums

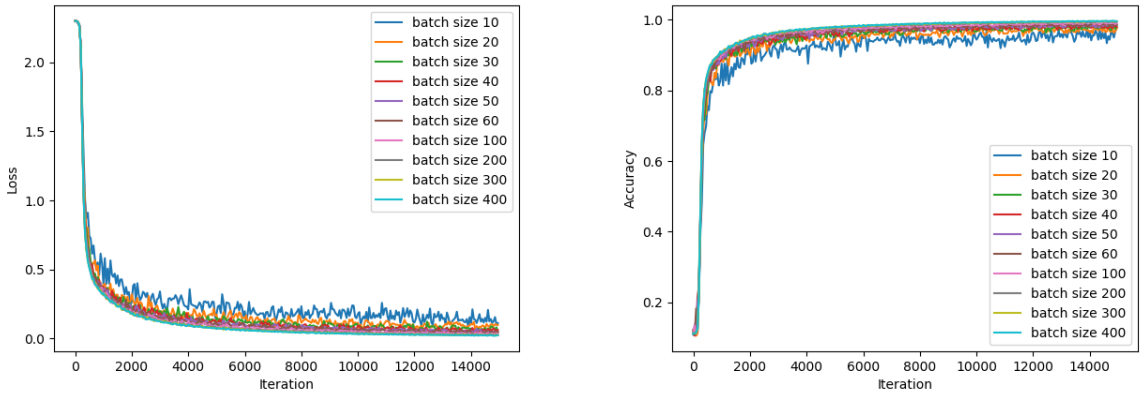
The experimental results are shown in Table 13.

Momentum	Train Loss	Train Accuracy(%)	Test Loss	Test Accuracy(%)
0.0	0.0791	0.9781	0.0848	0.9736
0.1	0.0706	0.9807	0.0807	0.9753
0.2	0.0621	0.9832	0.0772	0.9760
0.3	0.0537	0.9858	0.0747	0.9770
0.4	0.0452	0.9885	0.0727	0.9780
0.5	0.0371	0.9911	0.0703	0.9787
0.6	0.0298	0.9935	0.0674	0.9795
0.7	0.0233	0.9957	0.0650	0.9807
0.8	0.0183	**0.9973**	0.0605	0.9823
0.9	**0.0174**	0.9972	**0.0541**	**0.9834**
1.0	nan	0.0987	nan	0.0980

Table 13: Experimental Results with Different Momentums

3.2.3 Batch Size

A moderate batch size provides a stable convergence path and save memory. In this experiment, the optimal batch size will be explored from 10 to 400. The training process with different batch sizes is shown in Figure 6.



(a) Training Loss against Iteration

(b) Training Accuracy against Iteration

Figure 6: Training Process with Different Batch Sizes

The experimental results are shown in Table 14.

Batch Size	Train Loss	Train Accuracy(%)	Test Loss	Test Accuracy(%)
10	0.0837	97.32	0.0763	97.72
20	0.0452	98.56	0.0603	98.26
30	0.0327	99.01	**0.0511**	**98.51**
40	0.0259	99.29	0.0525	98.33
50	0.0224	99.45	0.0539	98.28
60	0.0204	99.55	0.0518	98.29
100	**0.0174**	99.72	0.0541	98.34
200	0.0183	**99.73**	0.0605	98.23
300	0.0233	99.57	0.0623	98.18
400	0.0297	99.36	0.0662	98.07

Table 14: Experimental Results with Different Batch Sizes

4 Conclusions

Compared to the Sigmoid activation, the Relu activation provides faster convergence, higher accuracy and less computational cost.

Two-hidden-layer networks usually achieve higher accuracy than one-hidden-layer networks do, but require longer time to train. Besides, Sigmoid activation in two-hidden-layer network might lead to vanishing gradients, making the model un-trainable. On the contrary, Relu activation could be applied to deeper networks without diminishing gradients.

Softmax cross entropy loss enables the networks to converge more rapidly than Euclidean loss does, resulting in a higher accuracy after training for same epochs.

Training process is largely controlled by learning rate. An excessively high learning rate might cause a gradient explosion, and an excessively low one would slow down the convergence.

Using momentum also speeds up the training process. The model usually achieves best performance with momentum of 0.9, but might crash with momentum higher than 0.9.

A suitable batch size provides stable gradient flow, and thus enables the model to converge steadily. It is usually set empirically.