Report on MNIST Digit Classification with MLP

千英卓 2017013622

1 Introduction

This experiment aims to train a MLP model to classify handwritten digits in the data set of MNIST.

During the experiment, it's required that all the layers and loss functions be implemented without any usage of deep learning framework. Also, multiple experiments on architecture and hyperparameters are conducted to reach the best performance. Analysis on the differences is included as well.

2 Network Architecture

2.1 General Configuration

Unless stated otherwise, the configuration of networks is set as below:

parameters	learning rate	weight decay	momentum	batch size	max epoch
value	0.01	0.001	0.8	64	50

2.2 Detailed Architecture

It's required that MLP with one hidden layer be first implemented.

2.2.1 One Hidden Layer with RELU

This network uses RELU as activation function. Detailed architecture is presented below.

Layer	in_feature	out_feature

Layer	in_feature	out_feature
Linear	784	256
RELU	256	256
Linear	256	10
RELU	10	10

2.2.2 One Hidden Layer with Sigmoid

This network uses Sigmoid as activation function. Detailed architecture is presented below.

Layer	in_feature	out_feature
Linear	784	256
Sigmoid	256	256
Linear	256	10
Sigmoid	10	10

2.2.3 Models with two hidden layers

Exploration on the architecture of MLP with two hidden layers is suggested. Some models with outstanding performance are listed below.(for convenience, these models will not be named according to their architecture, but sequence of numbers instead)

Note: the traning epochs of following models are 150.

Layer	in_feature	out_feature
Linear	784	392
RELU	392	392
Linear	392	128
RELU	128	128
Linear	128	10

Layer	in_feature	out_feature
Sigmoid	10	10

Model 1

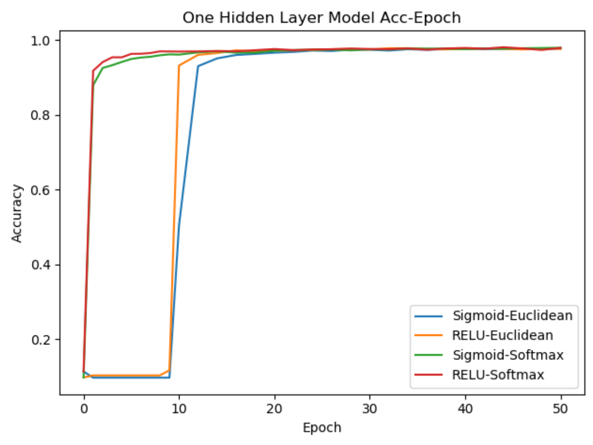
Layer	in_feature	out_feature
Linear	784	392
RELU	392	392
Linear	392	128
Sigmoid	128	128
Linear	128	10
Sigmoid	10	10

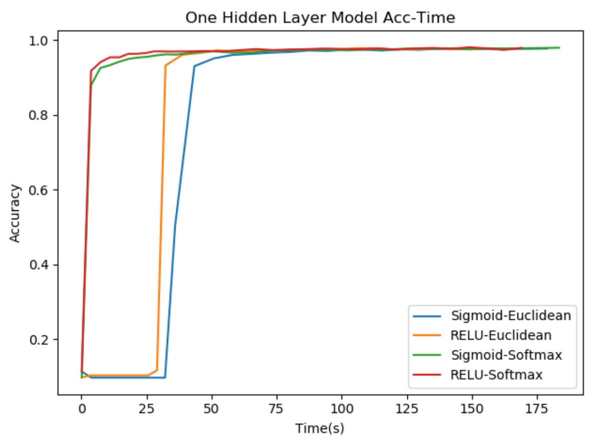
Model 2

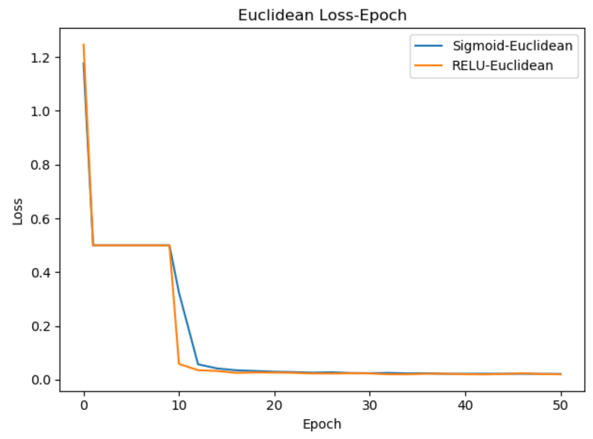
3 Results

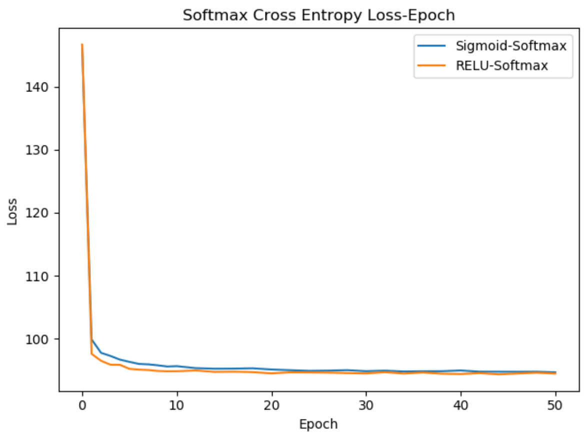
3.1 Results of models with one hidden layer

Following figures depict the change of accuracy and loss during the training of models presented in 2.2.1 and 2.2.2:









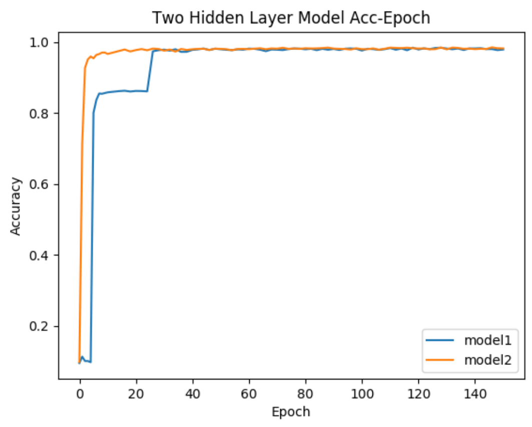
Architecture	Accuracy	Time(s)
Sigmoid & Euclidean Loss	0.9778	179
RELU & Euclidean Loss	0.9768	161
Sigmoid & Softmax Cross Entropy Loss	0.9798	183
RELU & Softmax Cross Entropy Loss	0.9795	169

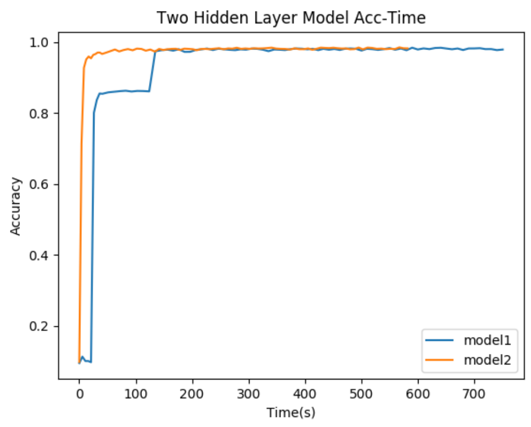
As can be seen, all four models reached rather high performance within a short period of time. In particular, model with Sigmoid as activation function and Softmax Cross Entrophy loss achieved best accuracy among the four, but it also takes longest to train. Comparatively, although slightly lower in accuracy, model with RELU & Softmax Cross Entropy gained high accuracy almost immediately (within one epoch), and was the fastest to reach convergence.

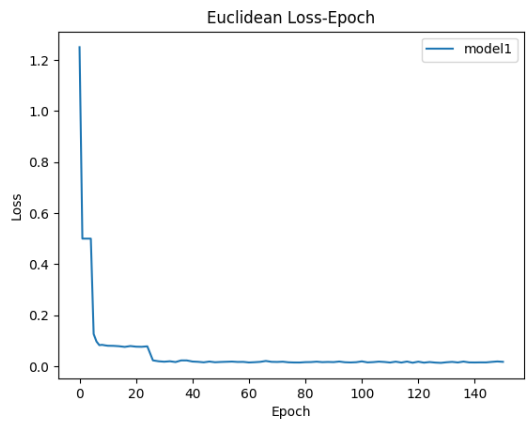
It can be assumed that the RELU activation function accelarates the training without greatly compromising accuracy. Softmax Cross Entropy as loss function can effectively improve accuracy but takes longer time to reach convergence.

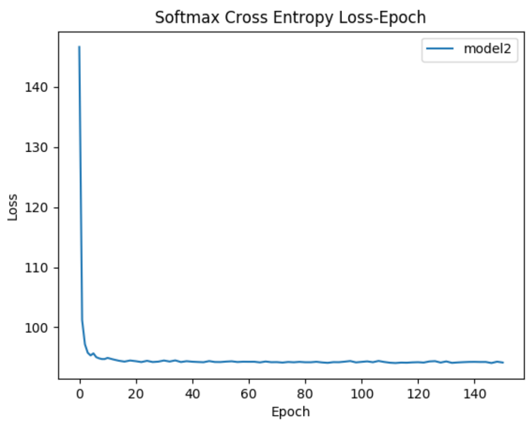
3.2 Results of models with two hidden layers

Following figures depict the change of accuracy and loss in the training of model1 & model2 in 2.2.3:









Architecture	Accuracy	Time(s)
Model 1	0.9788	750
Model 2	0.9820	582

It's obvious that model2 outperforms model1 in both terms of accuracy and traing speed. Also, the accuracy of 98.20% is higher than any model with one hidden layer.

4 Improvement

4.1 Modification on the incorporation of momentum and weight decay

In the codes provided by TA, the incorporation of momentum and weight decay is described as follow:

$$\Delta\theta_t = \beta\Delta\theta_{t-1} + (\nabla_\theta E + \gamma\theta)$$
$$\theta_t = \theta_{t-1} - \alpha\Delta\theta_t$$

where α denotes learning rate, β denotes momentum coefficient and γ denotes weight decay coefficient.

However, according to *Ilya Sutskever et al.* (1999) and *Krogh & Hertz* (1992), the formula are more to be described as

$$\Delta\theta_t = \beta\Delta\theta_{t-1} - \alpha(\nabla_\theta E + \gamma\theta)$$
$$\theta_t = \theta_{t-1} + \Delta\theta_t$$

After applying the change to models in 2.2.2 & 2.2.3, the result of accuracy and training time is depicted below:

Architecture	Accuracy	Time(s)	Accuracy(after)	Times(after)
Sigmoid_SmCE Loss	0.9798	183	0.9776	183
RELU_SmCE Loss	0.9795	169	0.9796	165
Model 1	0.9788	750	0.9849	747
Model 2	0.9820	582	0.9829	806

As is shown above, though more time is required in training, the improvement in model1 & model2 in terms of accuracy is significant. However, no improvement is shown in models with one hidden layer.

4.2 Modification on convergence judging

In the experiments above, the training ends after certain numbers of epoch. However, the training may end before or long after reaching convergence. Therefore, another method is adopted: if the accuracy score on validation set doesn't exceed current highest for consecutive *c* times, where *c* is a predetermined hyperparameter, then stop the training. The result is shown below: (*c* is set at 5; modification on gradient descent adopted)

Architecture	Accuracy	Time(s)	Accuracy(after)	Times(after)
Model 1	0.9849	750	0.9791	240
Model 2	0.9829	806	0.9796	178

Note: when adopting this method, the last 10000 digits in training set will be seperated as validation set

In terms of performance, the improvement doesn't show clear advantage compared as it compromises accuracy for shorter training time. However, this method can serve as a way of self-adapted criteria, especially when the number of epochs needed is unknown.

5 Conclusion

In this experiment, I learned about the process of constructing and training neural networks, also bringing up some modification on current framework. I would like to thank professor Huang and all the TAs for providing this task.

Reference

Sutskever, I., Martens, J., Dahl, G., & Hinton, G. (2013, February). On the importance of initialization and momentum in deep learning. In International conference on machine learning (pp. 1139-1147).

Krogh, A., & Hertz, J. A. (1992). A simple weight decay can improve generalization. In Advances in neural information processing systems (pp. 950-957).