HW1 report

> zhangchi 2022010754 zhang-ch22@mails.tsinghua.edu.cn

1. Introduction

This experiment is to implement a simple neural network with only one or two hidden layers. The network is trained on the MNIST dataset. Different layer functions (including `Selu`, `Swish` and `Gelu`) and loss functions (including `MSELoss`, `SoftmaxCrossEntropyLoss`, `HingeLoss` and `FocalLoss`) are tested. The train&test loss and accuracy of all combinations are recorded and shown below.

2. Experiments on layer functions

Using `HingeLoss` as the loss function, we tested the loss and accuracy of the network with different layer functions. The hyperparameters are listed below:

```
learning_rate: 0.00001
weight_decay: 0.1
momentum: 0.9
batch_size: 100
max_epoch: 200
hidden_layer_size: 128
```

> These hyperparmeters are tested to have a good performance on most of the networks. All the following experiments use the same hyperparameters.

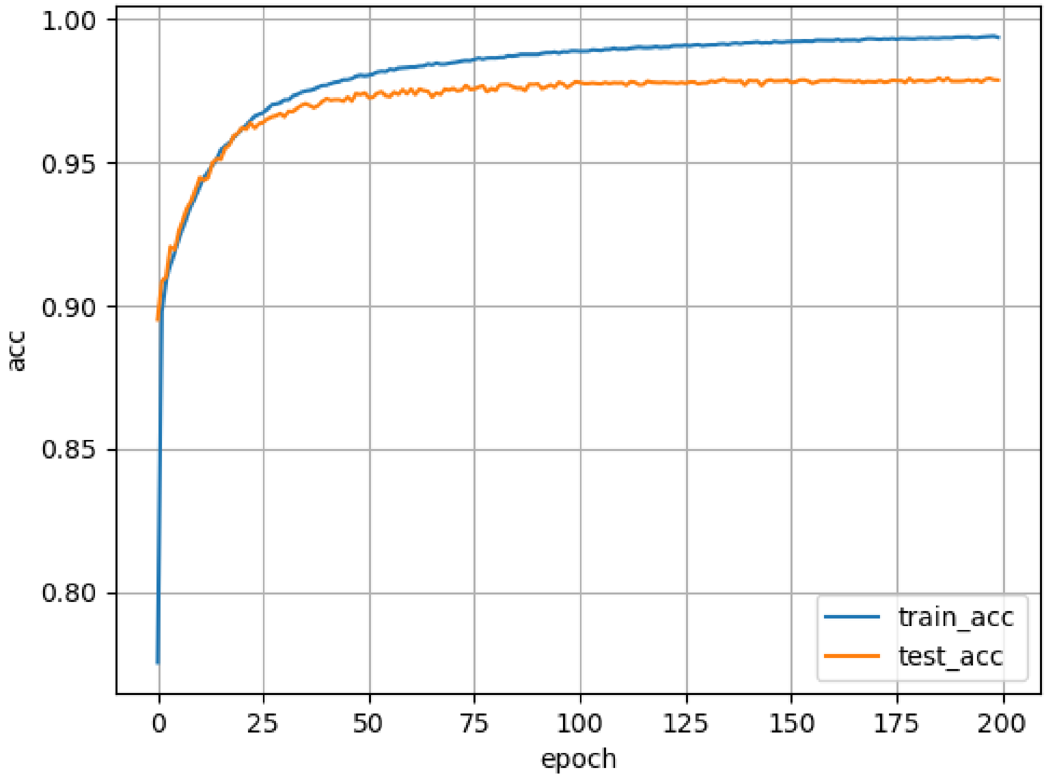
The results are shown in the following table:

layer function	Selu	Swish	Gelu
train loss	0.1420	0.1590	0.1261
test loss	0.4551	0.4526	0.4304
train accuracy	0.9936	0.9926	0.9947
test accuracy	0.9788	0.9793	0.9816
total time cost	415.1s	339.8s	533.7s

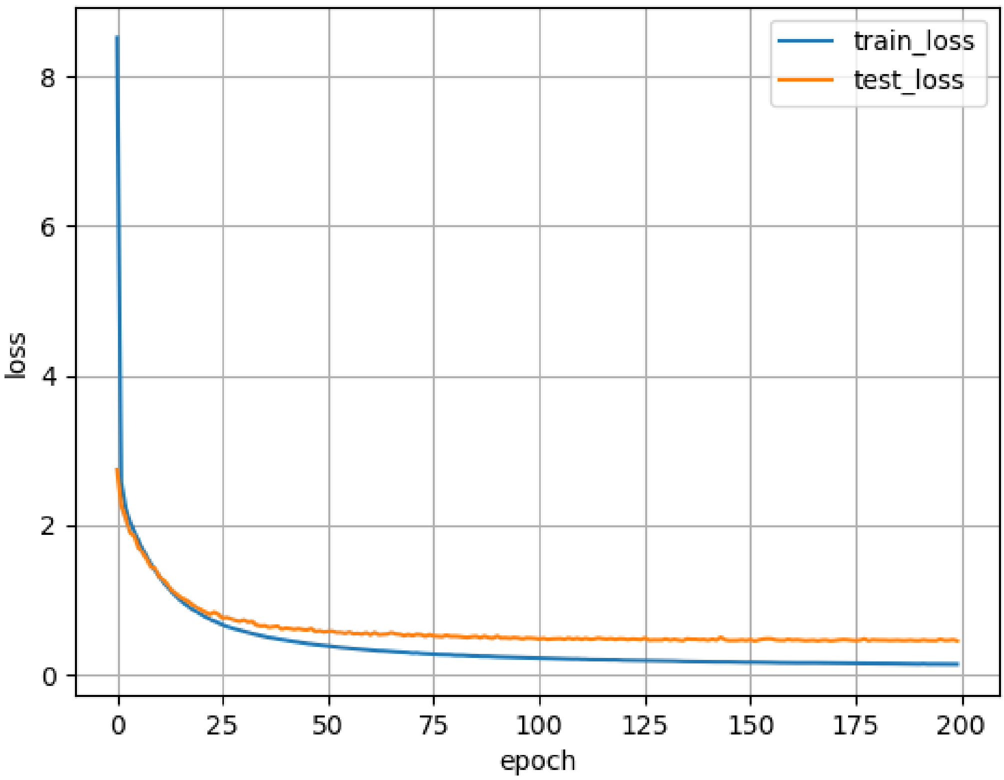
We can see that, using `HingeLoss`, no matter what the activate function is, the test accuracy is generally high (more than 97%). The `Gelu` function has the best performance, but the total time cost is the longest. It may be explainable by the fact that the `Gelu` function contains calculations of hyperbolic tangent and cosine functions, which might be time-consuming. The `Swish` function is the fastest.

The train&test losses and accuracies curves are shown below:

Selu:

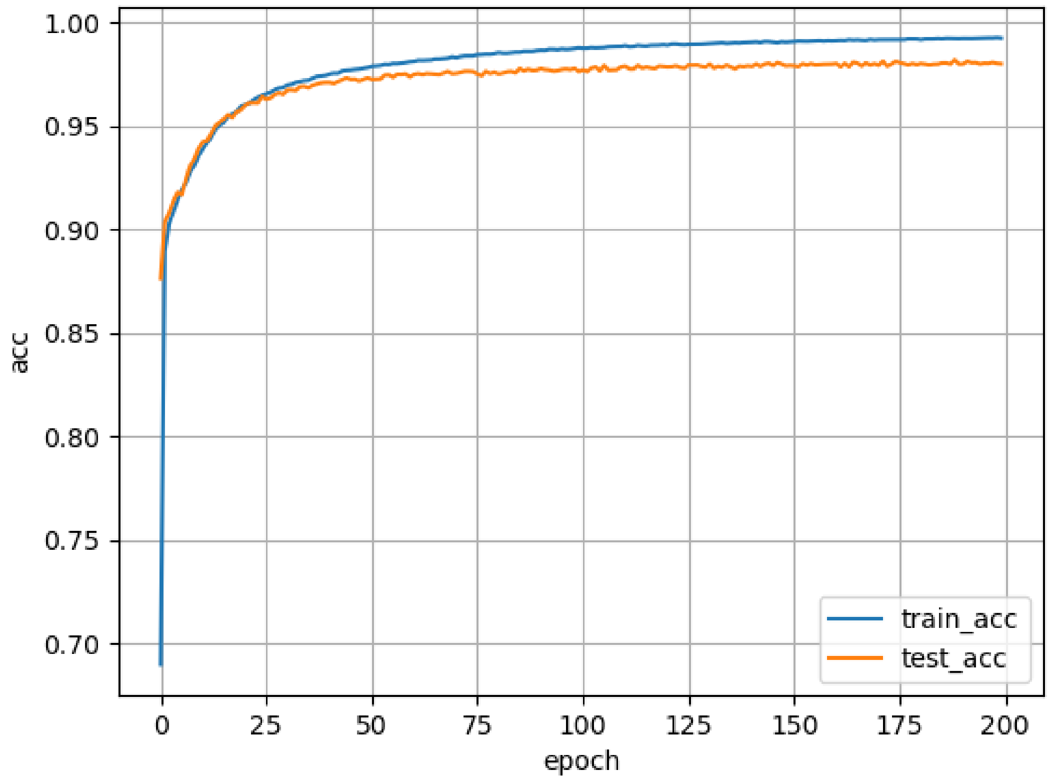


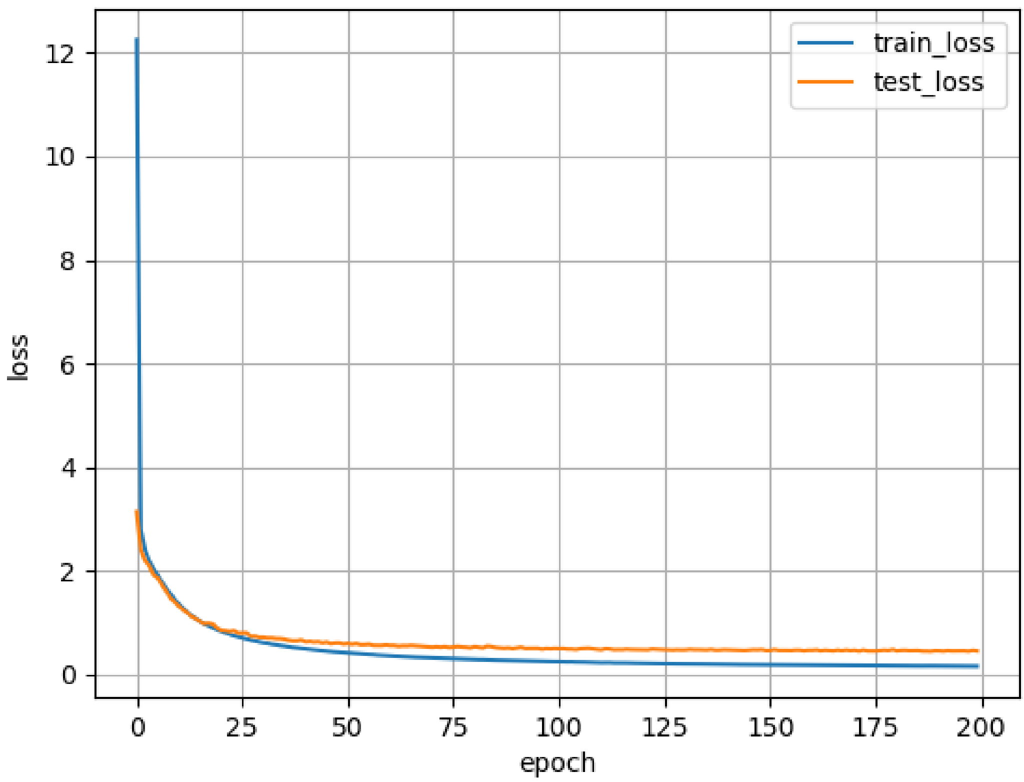
1.00
0.95
0.90
0.85
0.80
acc
0
25
50
75
100
125
150
175
200
epoch
train_acc
test_acc



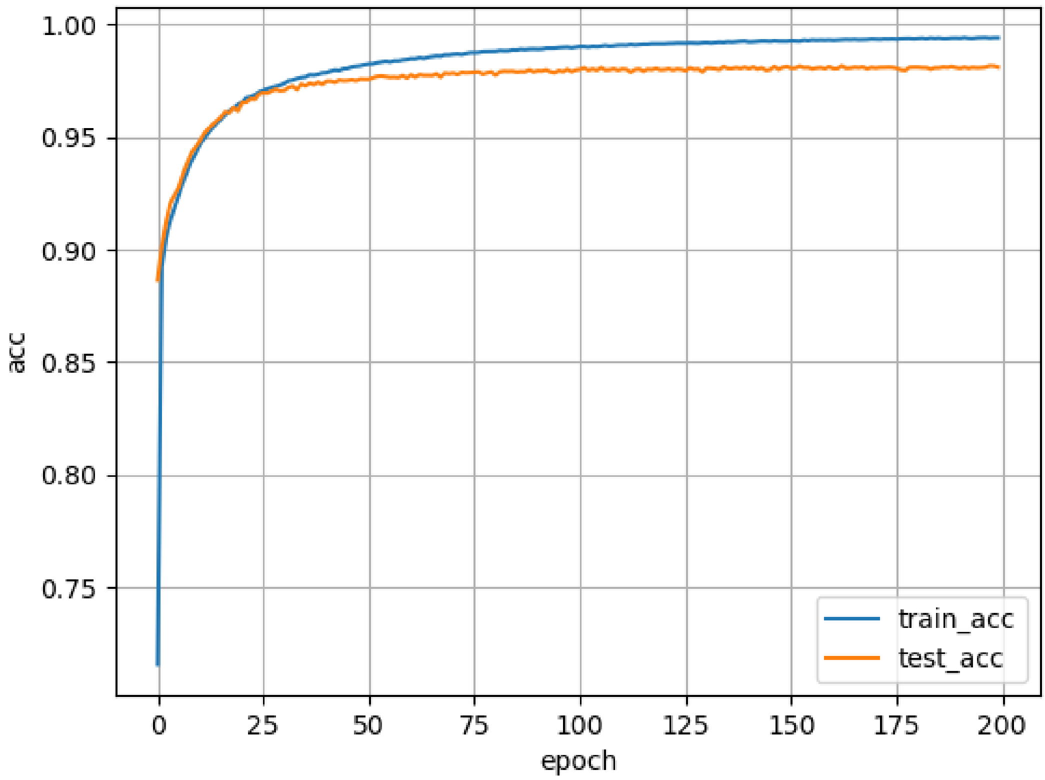
8
6
4
2
0
loss
0
25
50
75
100
125
150
175
200
epoch
train_loss
test_loss

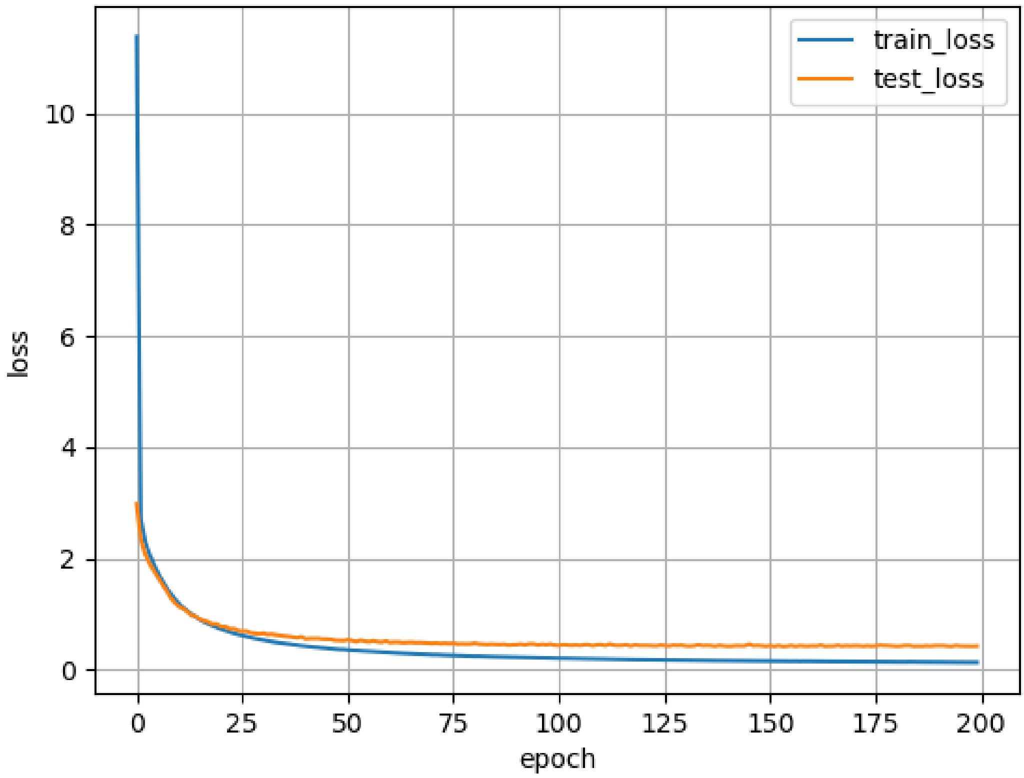
Swish:


1.00
0.95
0.90
0.85
0.80
0.75
0.70
acc
train_acc
test_acc
0
25
50
75
100
125
150
175
200
epoch


12
10
8
6
4
2
0
loss
train_loss
test_loss
0
25
50
75
100
125
150
175
200
epoch

Gelu:

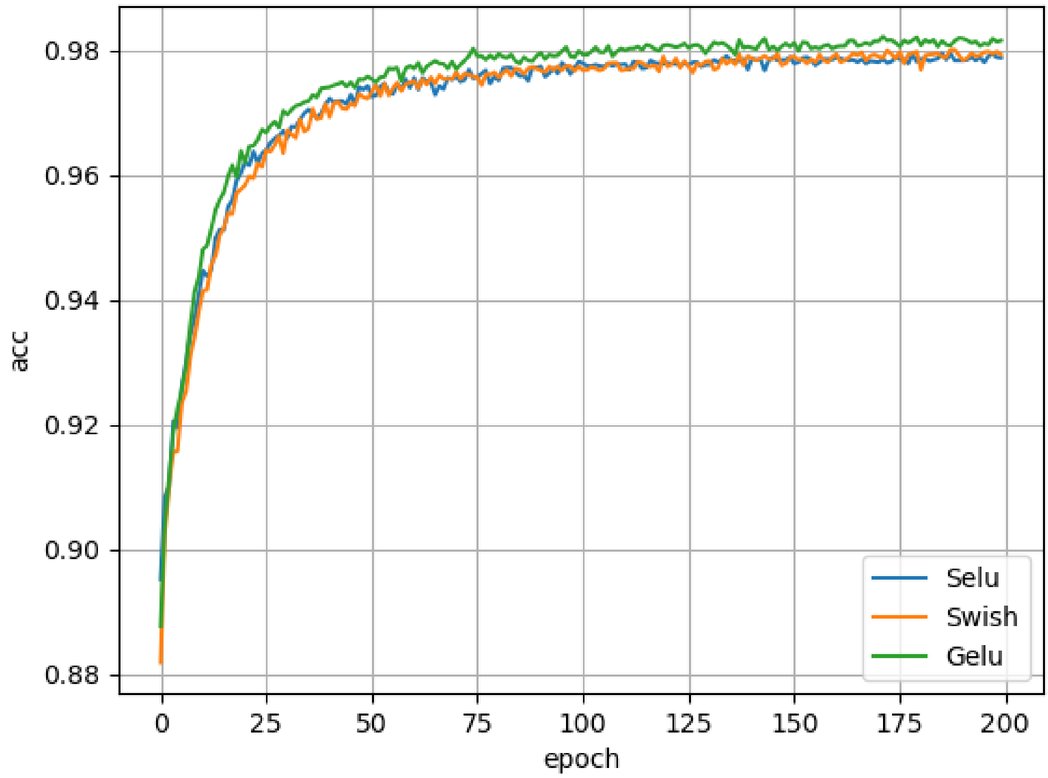


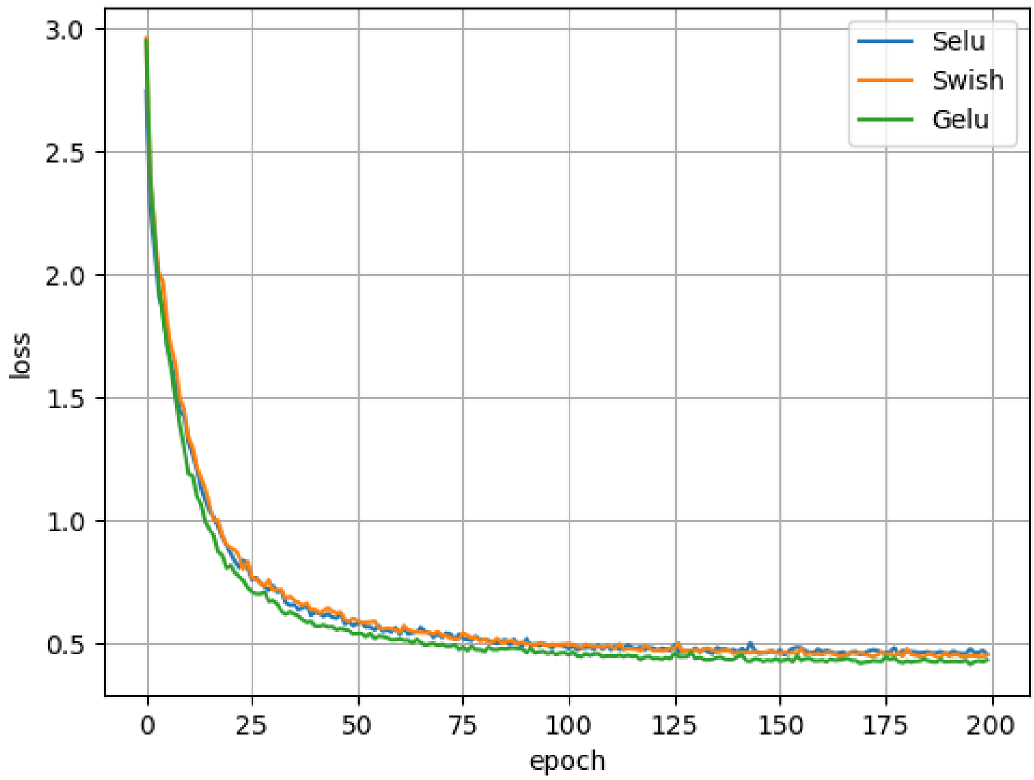


In all the curves above, the train&test losses and accuracies show good convergence. The test accuracy is lower than the train accuracy by a good margin, which means that the network is not overfitting.

We can make a clearer comparison by plotting the test accuracy curves of the three layer functions together:

> The plotted accuracies and losses are the test accuracies and losses.





We can see the curves of the different activate functions are very close to each other, showing that the influence of the activate function on the training of the network is not very significant.

In fact, using other loss functions, the `Gelu` function also outperforms the other two. For example, if we use `MSELoss` as the loss function, the test/train accuracies are shown below:

layer function	Selu	Swish	Gelu
train accuracy	0.9666	0.9670	0.9713
test accuracy	0.9634	0.9657	0.9686

3. Experiments on loss functions

Using `Gelu` as the layer function, we tested the loss and accuracy of the network with different loss functions. The hyperparameters are the same as above.

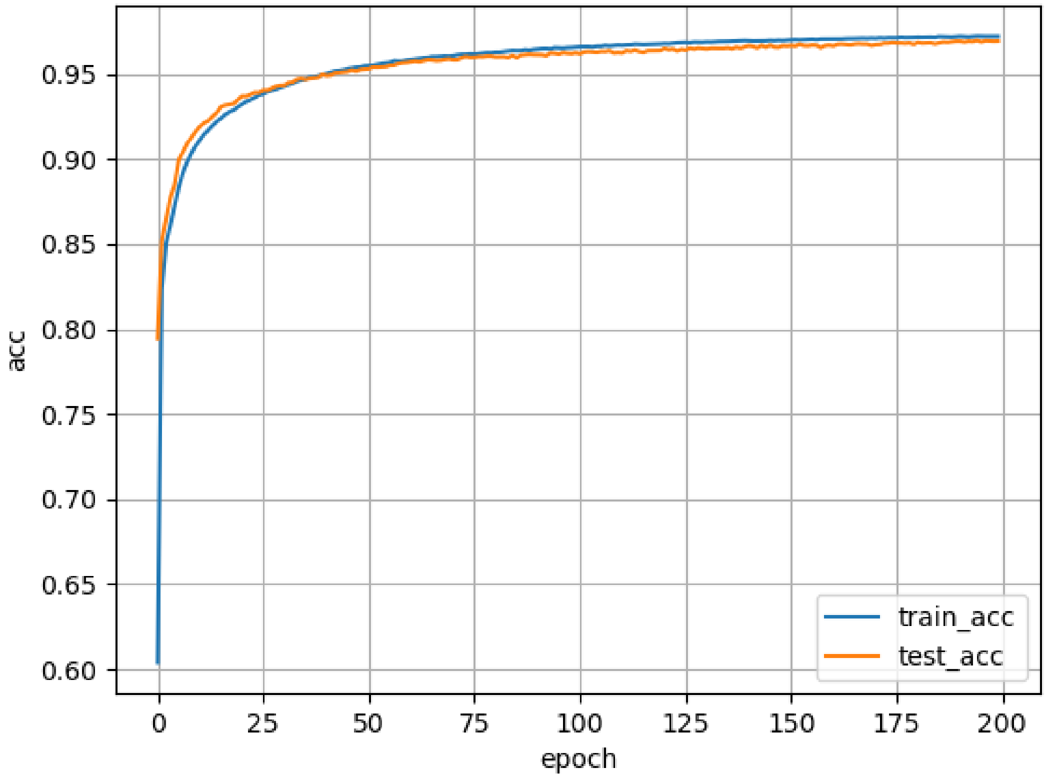
The results are listed in the following table:

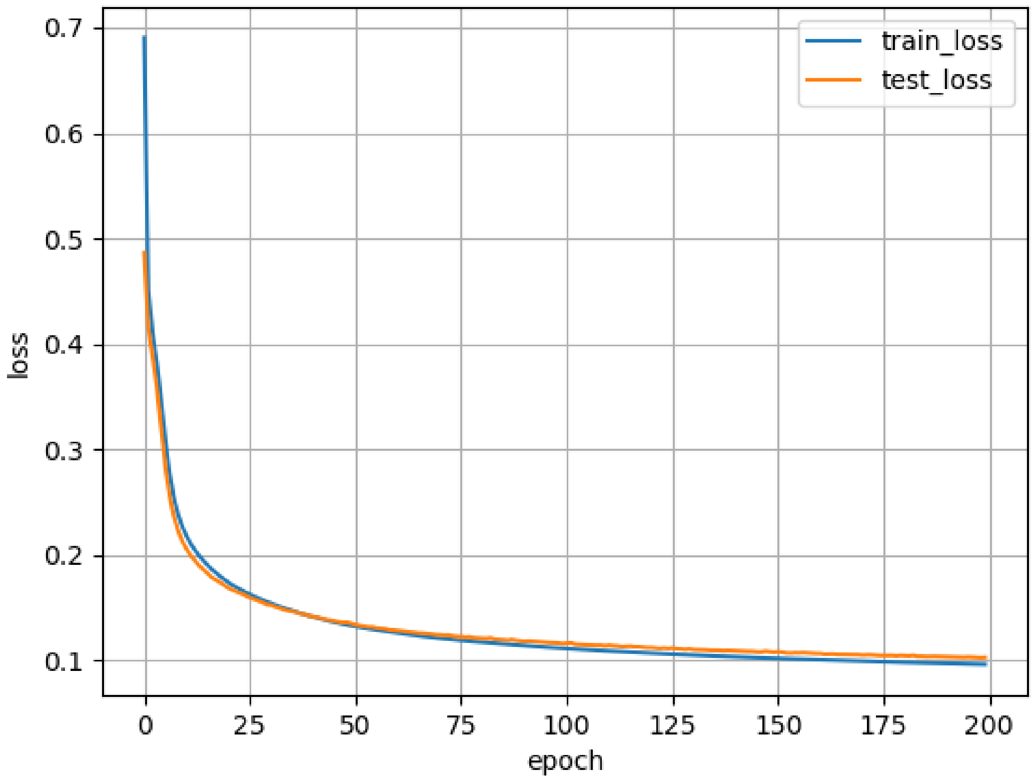
loss function	MSELoss	SoftmaxCrossEntropyLoss	HingeLoss	FocalLoss
train loss	0.0960	0.0842	0.1261	0.0178
test loss	0.1021	0.0979	0.4304	0.0171
train accuracy	0.9713	0.9785	0.9947	0.9081
test accuracy	0.9686	0.9730	0.9816	0.9095
total time cost	532.2s	544.6s	533.7s	580.5s

Changing the loss function makes a greater impact on the lossse and accuracies than changing the activate function. Meanwhile, it makes a rather smaller impact on the total time cost for training, because the loss function is called only once for each iteration. So, choosing a good loss function for a certain task is more important than choosing the activate function. `HingeLoss` performs the best in this experiment, followed by `SoftmaxCrossEntropyLoss`. It is reasonable because they are designed for classification tasks, while `MSELoss` is designed for regression tasks. `FocalLoss` performs the worst, a possible explanation is that `FocalLoss` is designed for imbalanced classification tasks, while the MNIST is a balanced dataset.

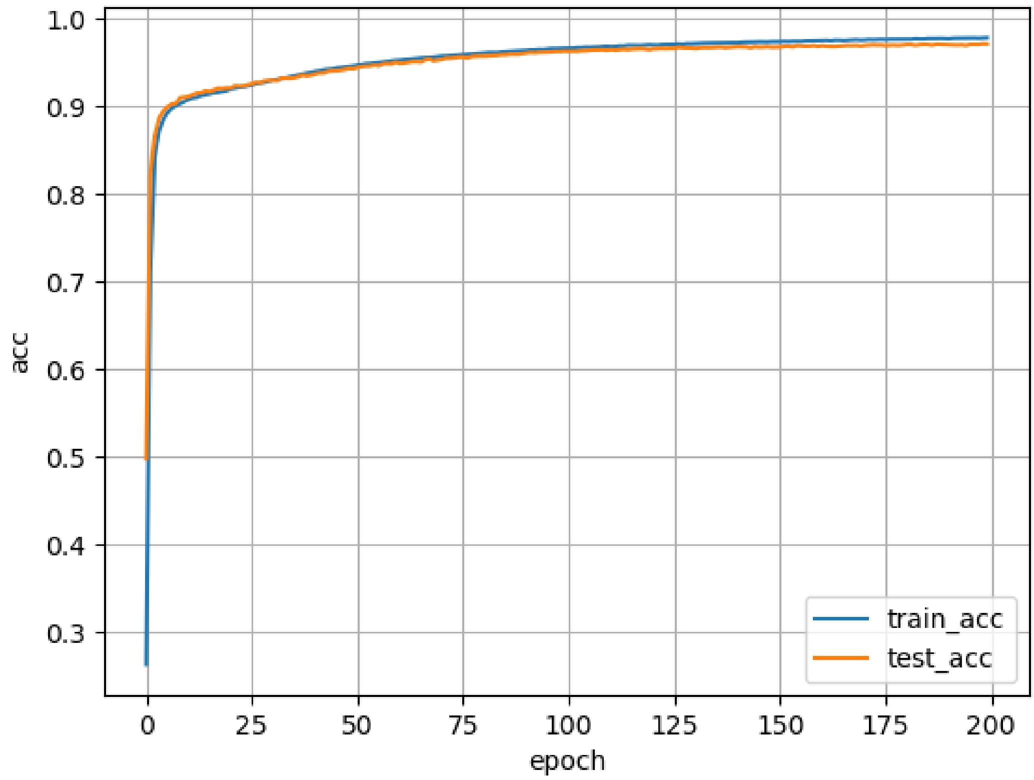
The train&test losses and accuracies curves are shown below:

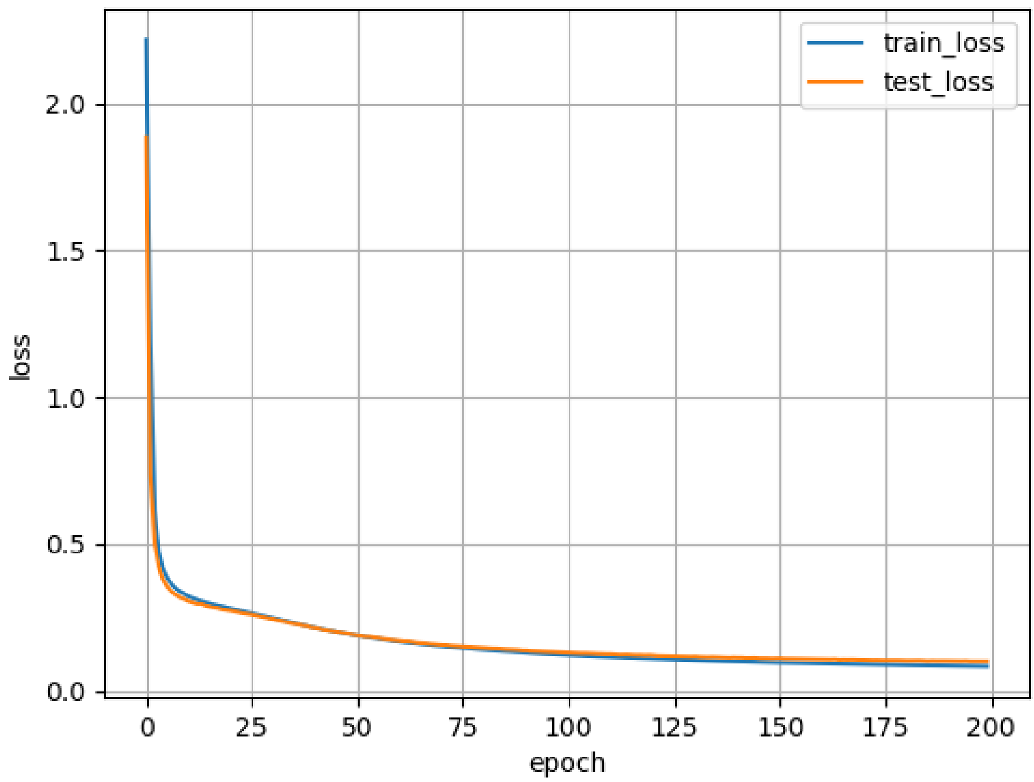
MSELoss:



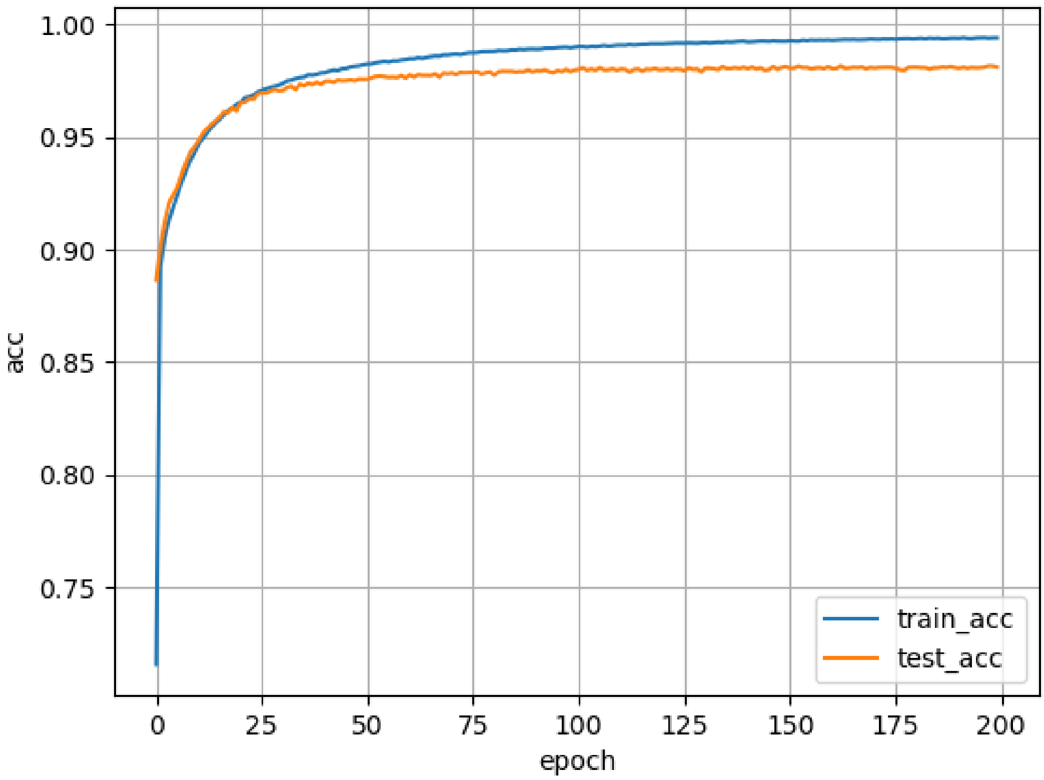


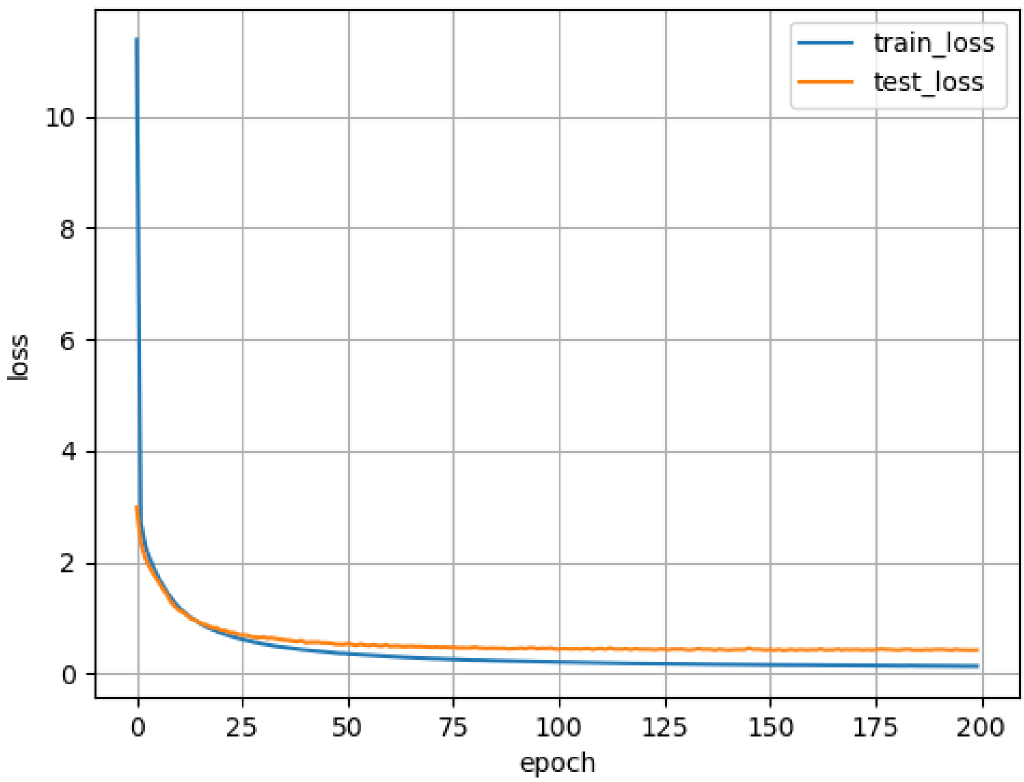
SoftmaxCrossEntropyLoss:


1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
acc
train_acc
test_acc
0
25
50
75
100
125
150
175
200
epoch

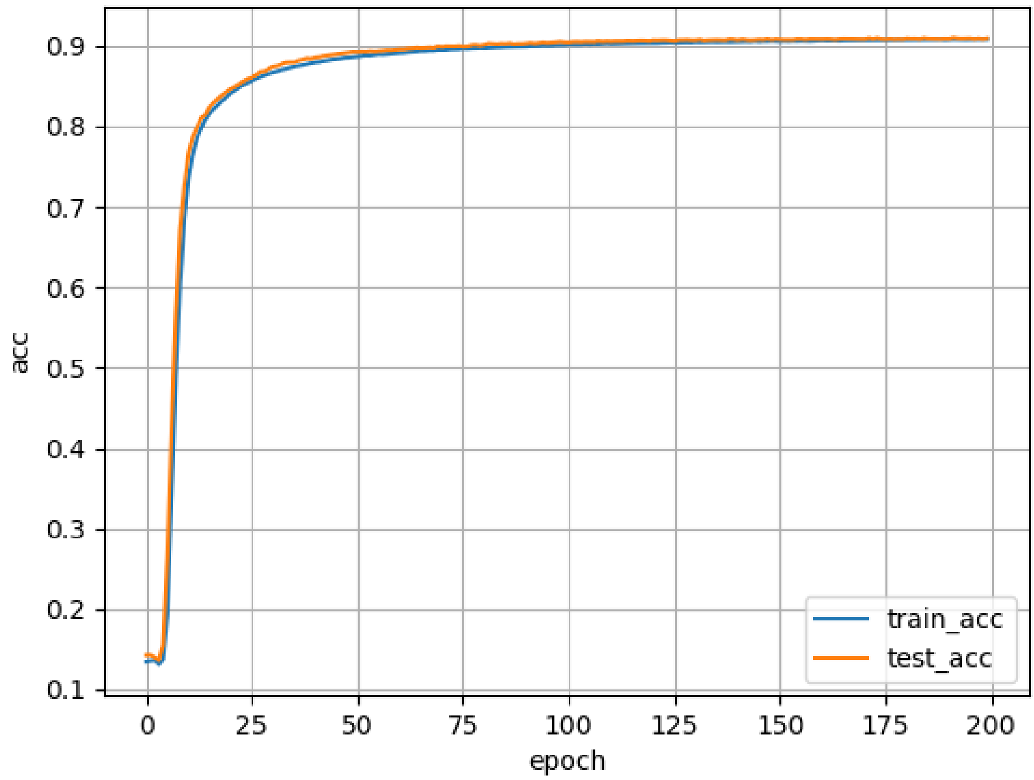

2.0
1.5
1.0
0.5
0.0
loss
train_loss
test_loss
0
25
50
75
100
125
150
175
200
epoch

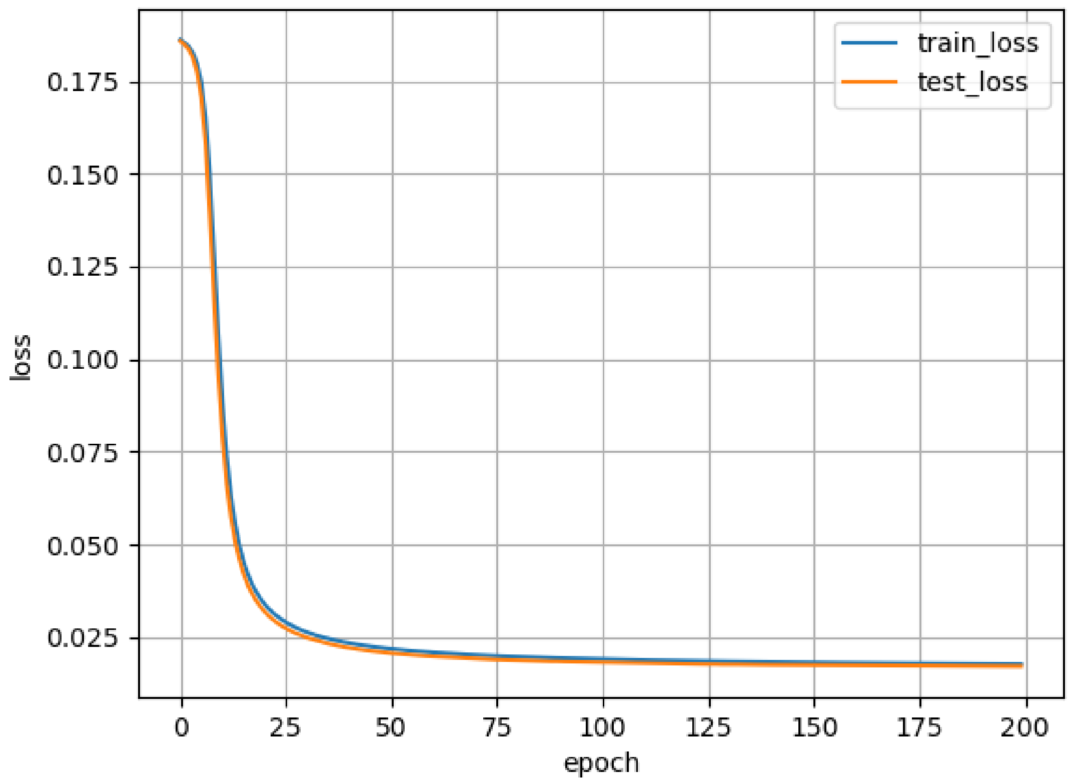
HingeLoss:


1.00
0.95
0.90
0.85
0.80
0.75
acc
train_acc
test_acc
0
25
50
75
100
125
150
175
200
epoch


10
8
6
4
2
0
loss
train_loss
test_loss
0
25
50
75
100
125
150
175
200
epoch

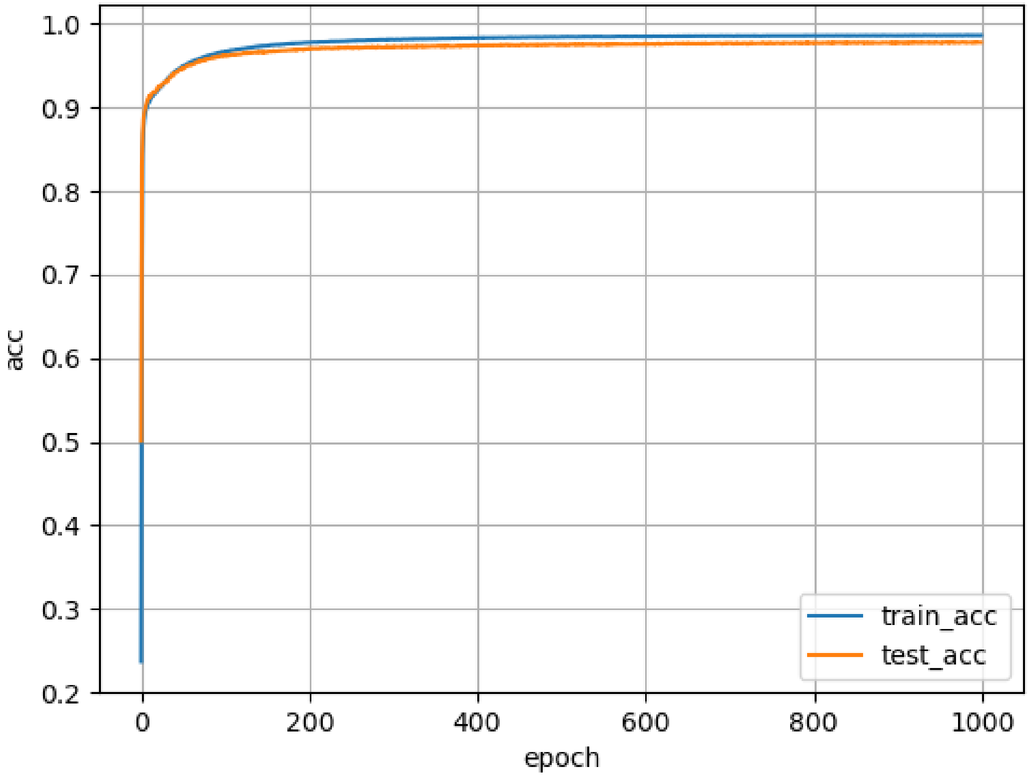
FocalLoss:

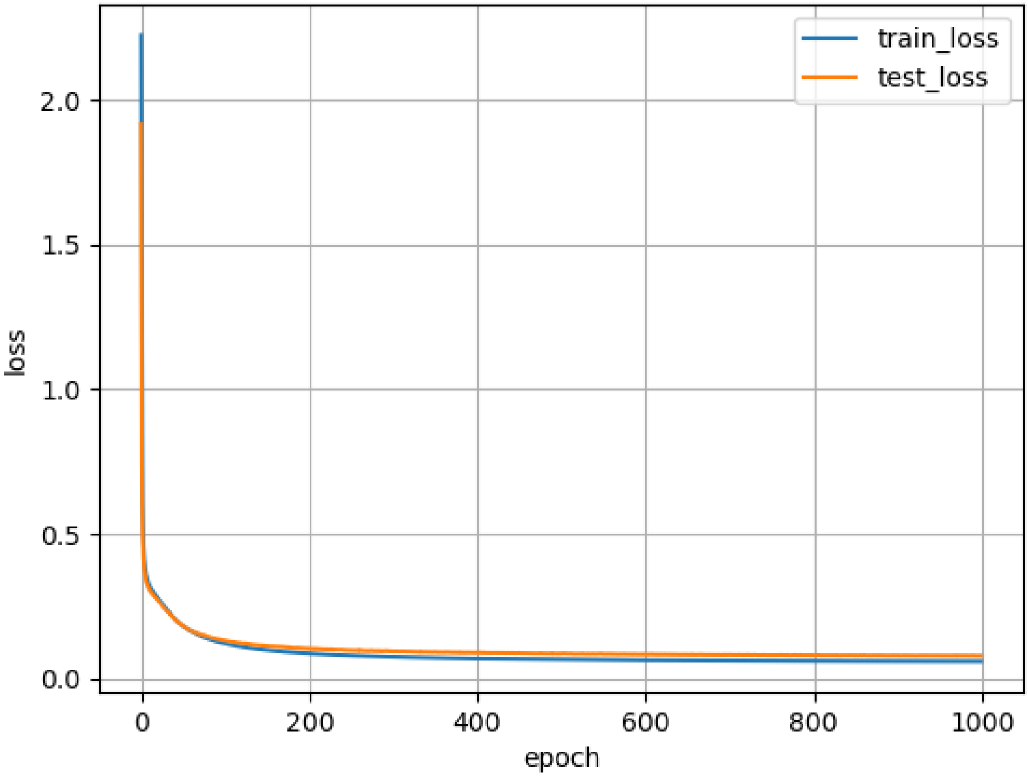




Only the `HingeLoss` network shows a suitable gap between the train and test curves, which means the size of the model corresponds to the size of the dataset.

I once wondered if the small gap of the other networks are due to unsufficient training, so I added the total epoch to 1000, and increased the learning rate. The results are shown below:

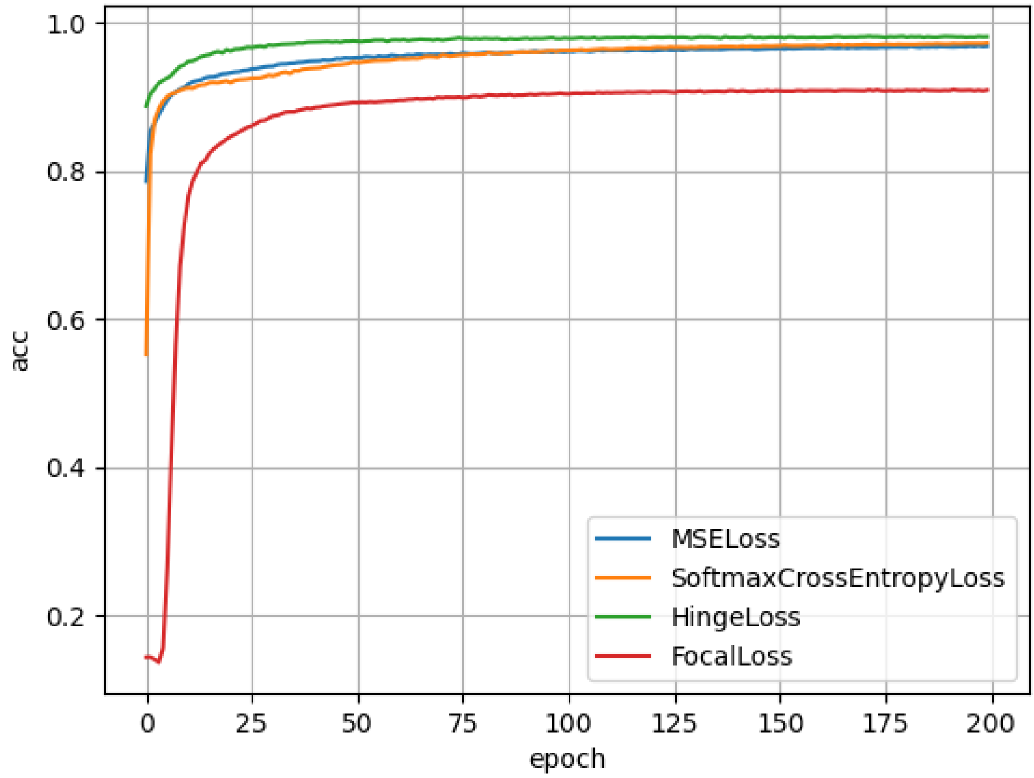




The gap between the train and test curves are basically the same. So the small gap seems to have other causes. Perhaps a bigger model might be needed for the other loss functions.

It takes some iterations for the loss to start to drop in the `FocalLoss` network, which means the initialization might not fit this loss function.

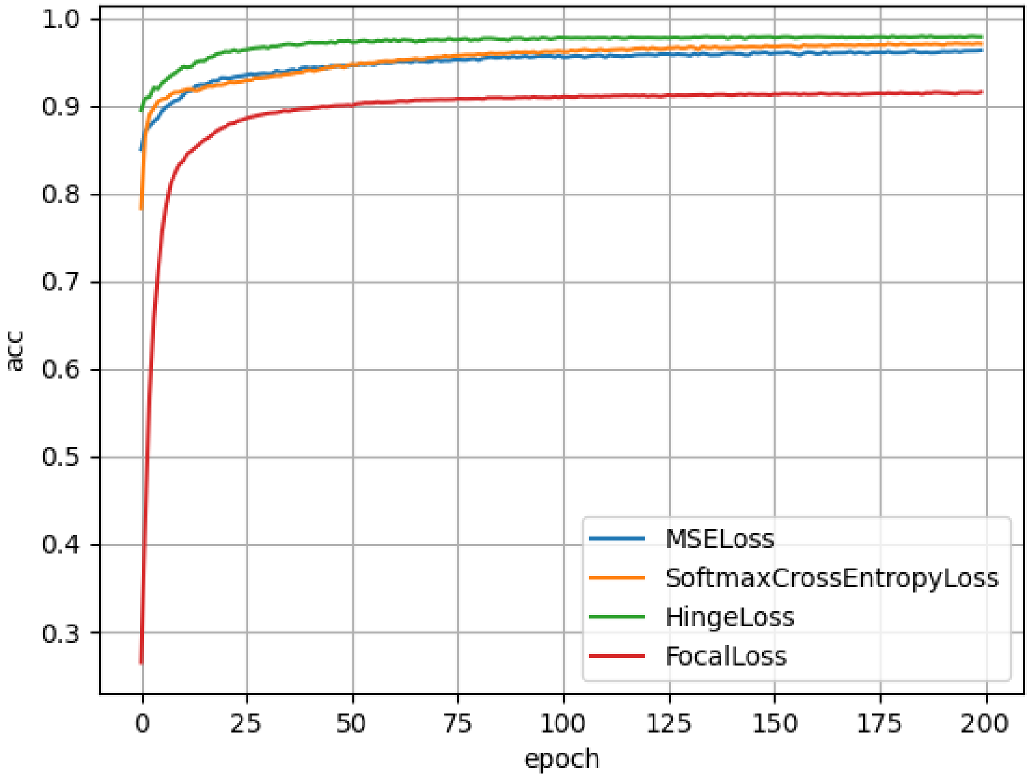
To make clearer comparison, we plot the test accuracy curves of the four loss functions together:



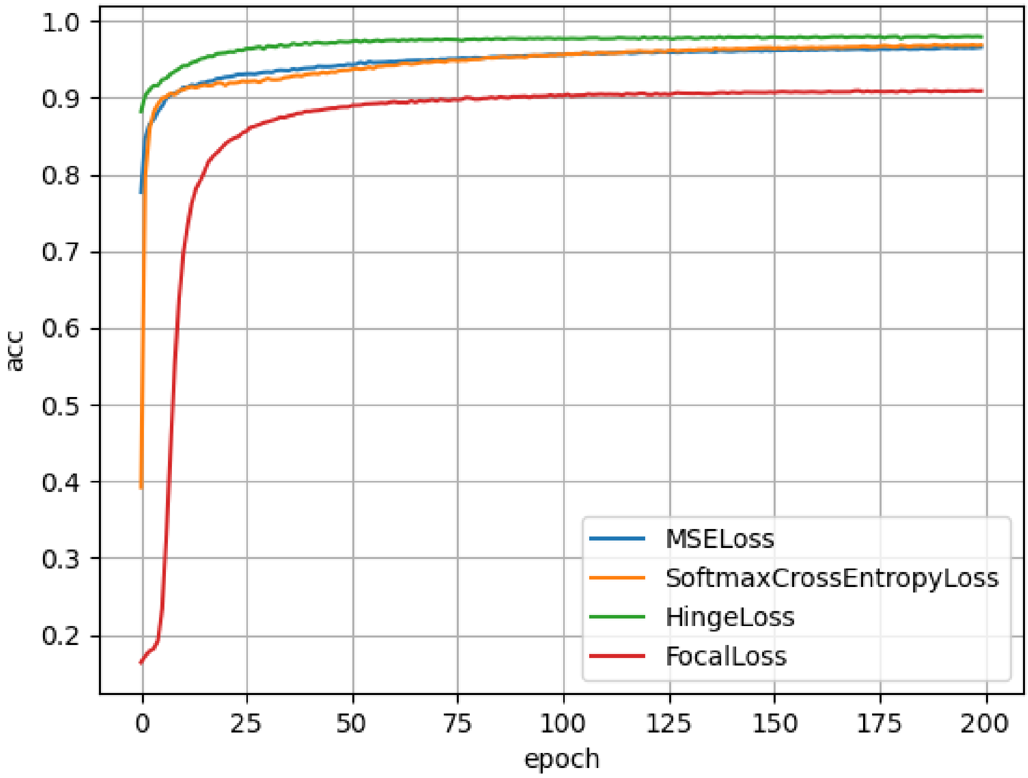
We can see that the `HingeLoss` network outperforms the other network, either in final accuracy or speed of convergence.

If we change the `Gelu` function to other activate functions, as below:

Selu:



Swish:



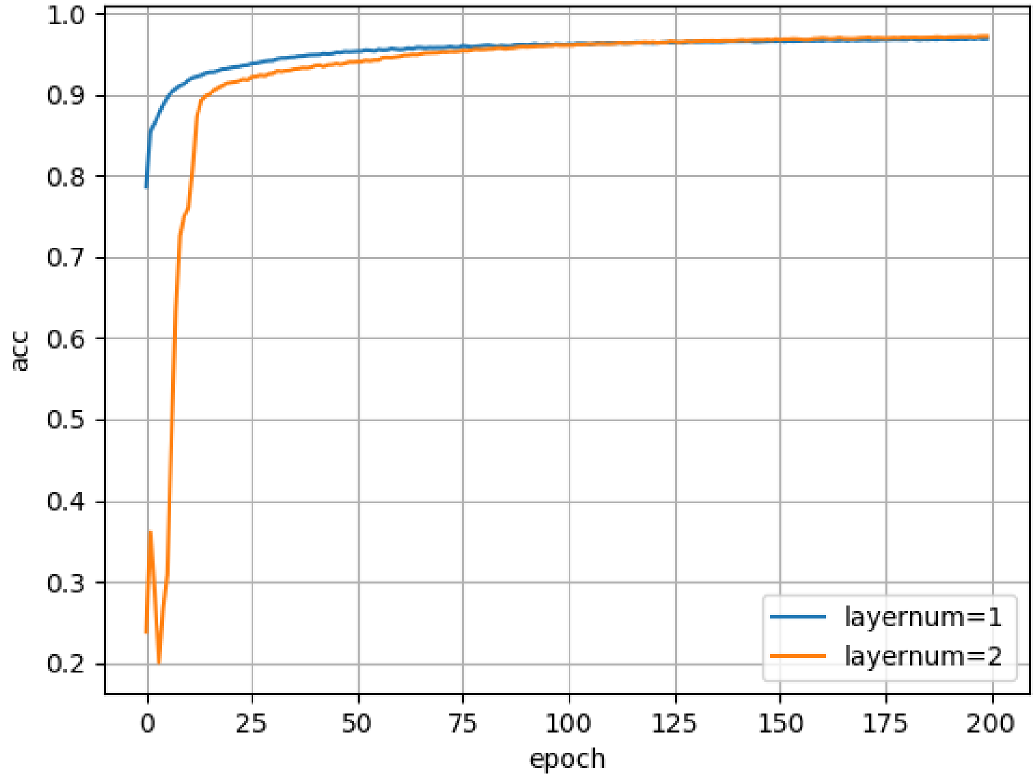
The curves are quite similar, with `HingeLoss` still performs the best, showing that the influence of the loss function is greater than that of the activate function. `FocalLoss` combined with `Selu` seems to be able to avoid bad initializing, but the final accuracy is still not good.

4. Experiments on layer num

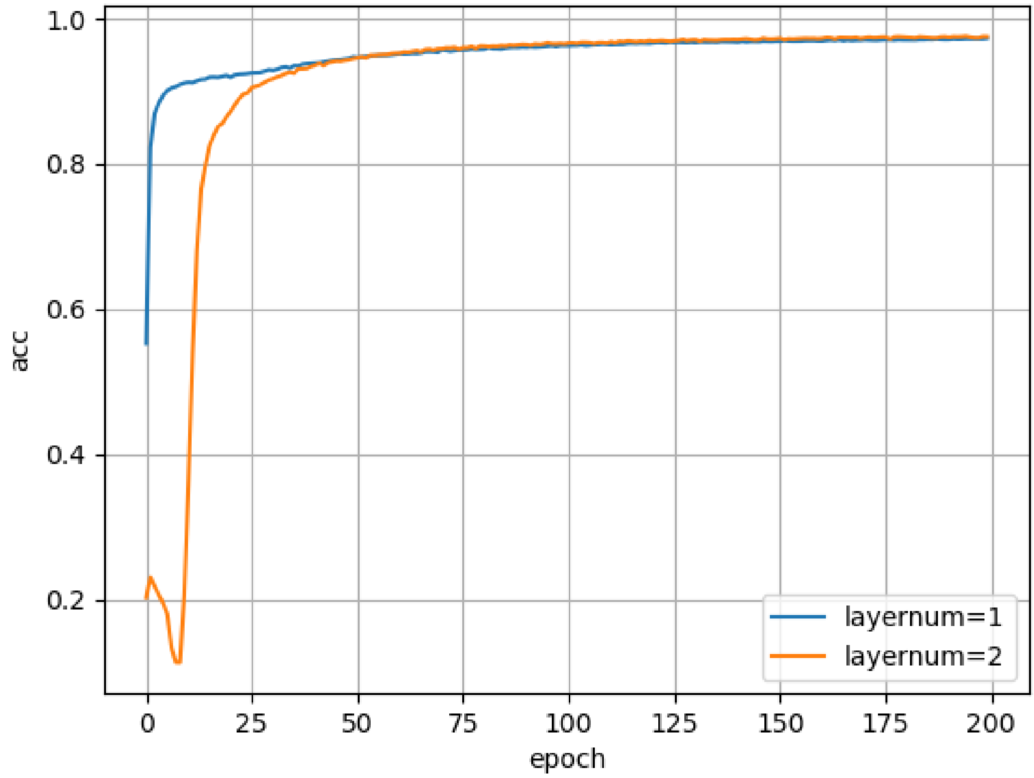
We added the layer num to 2, and conducted the same experiments. The new hidden layer'size is setted to 32. So the network is a 784-128-32-10 network.

Choosing the `Gelu` function as the activate function, we can plot the following curves (of test accuracy):

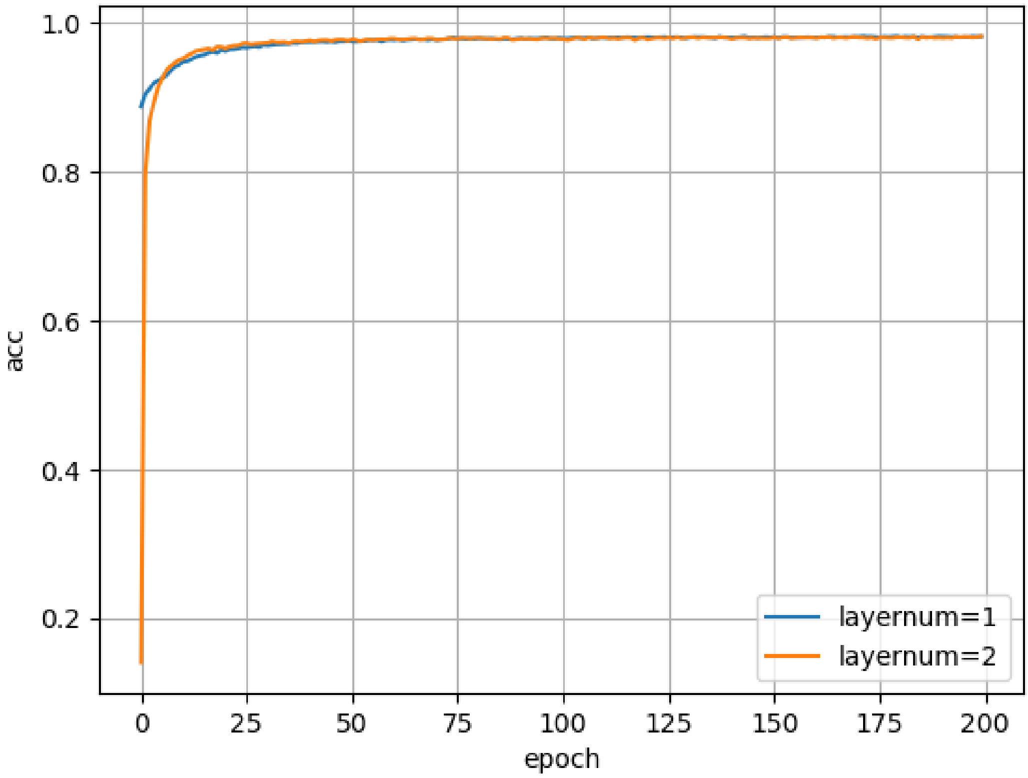
MSELoss:



SoftmaxCrossEntropyLoss:



HingeLoss:



The final accuracies are close in the above curves, but the 2-layer network took a longer time to converge. The 2-layer version of the `MSELoss` and `SoftmaxCrossEntropyLoss` seems to have some problems with the initialization, as the accuracies dropped in the beggining epochs. There is a corresponding peak in the loss curves as well.

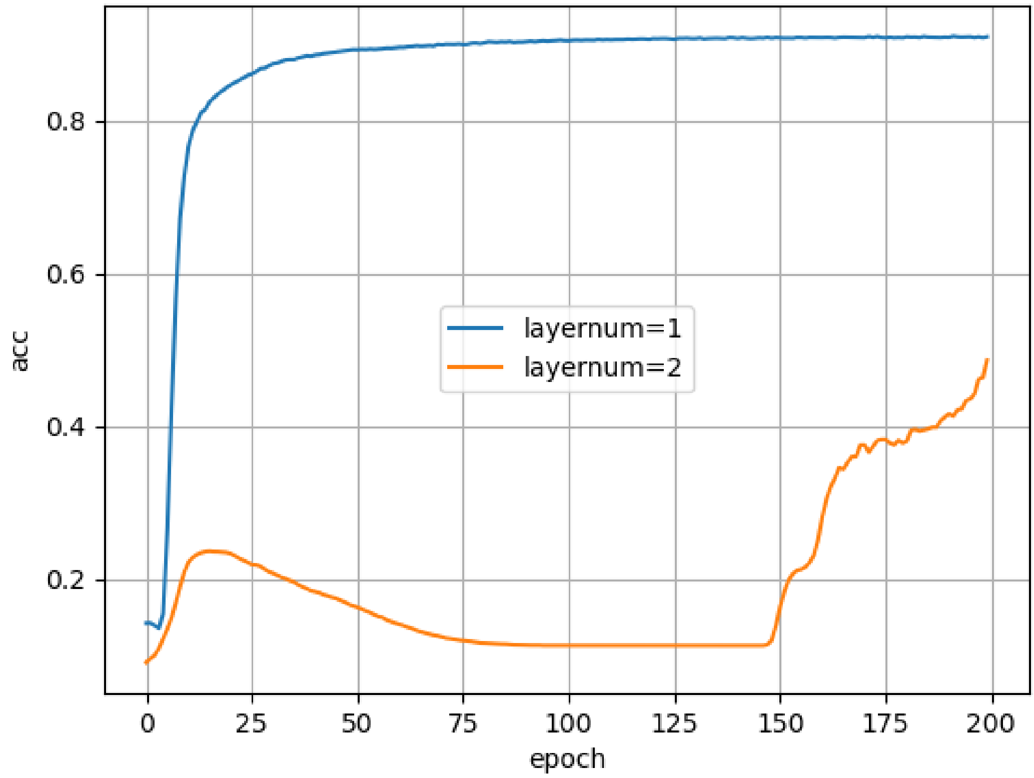
The time consumption of the 2-layer network is also larger, as the following table shows:

total time	1-layer	2-layer
MSELoss	532.2s	610.1s
SoftmaxCrossEntropyLoss	544.6s	616.5s
HingeLoss	533.7s	627.8s
FocalLoss	580.5s	658.7s

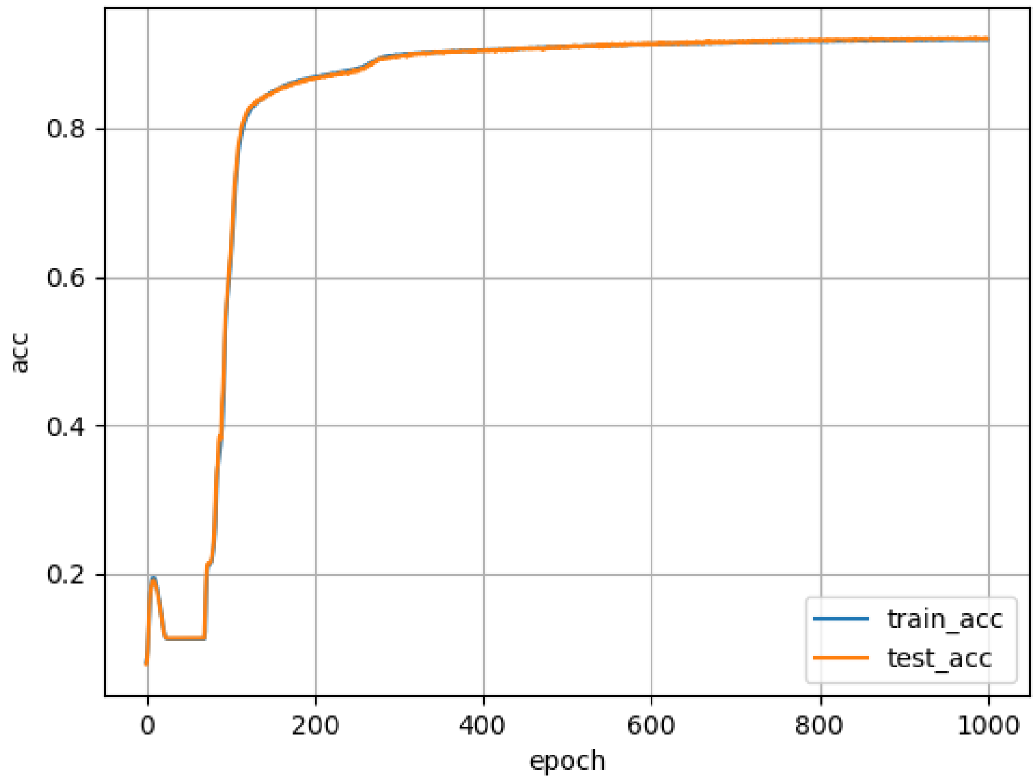
Because the second layer has only 32 neurons, the training time doesn't go to long.

Things become different for `FocalLoss`, as shown below:

FocalLoss:



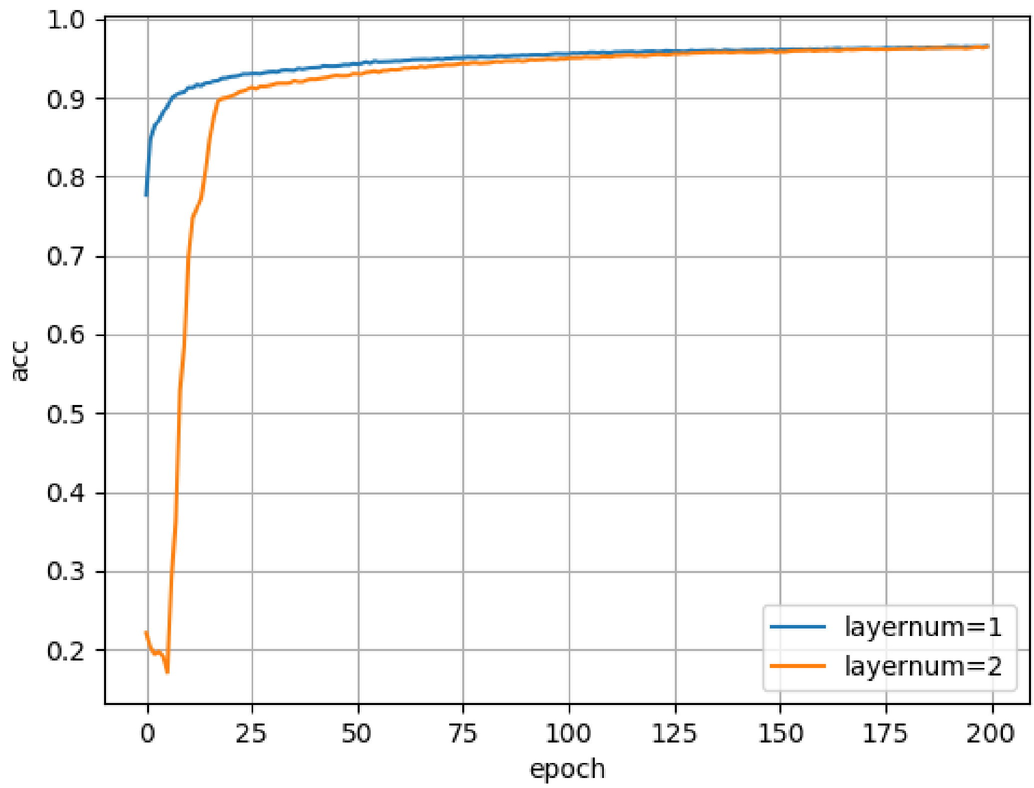
The speed of convergence is too slow that the accuracy remains under 50% for 200 epochs. To prove that the poor performance is due to lack of training iterations, I trained this network for 1000 epochs, and increased the learning rate to 0.0002. The results are as follows:



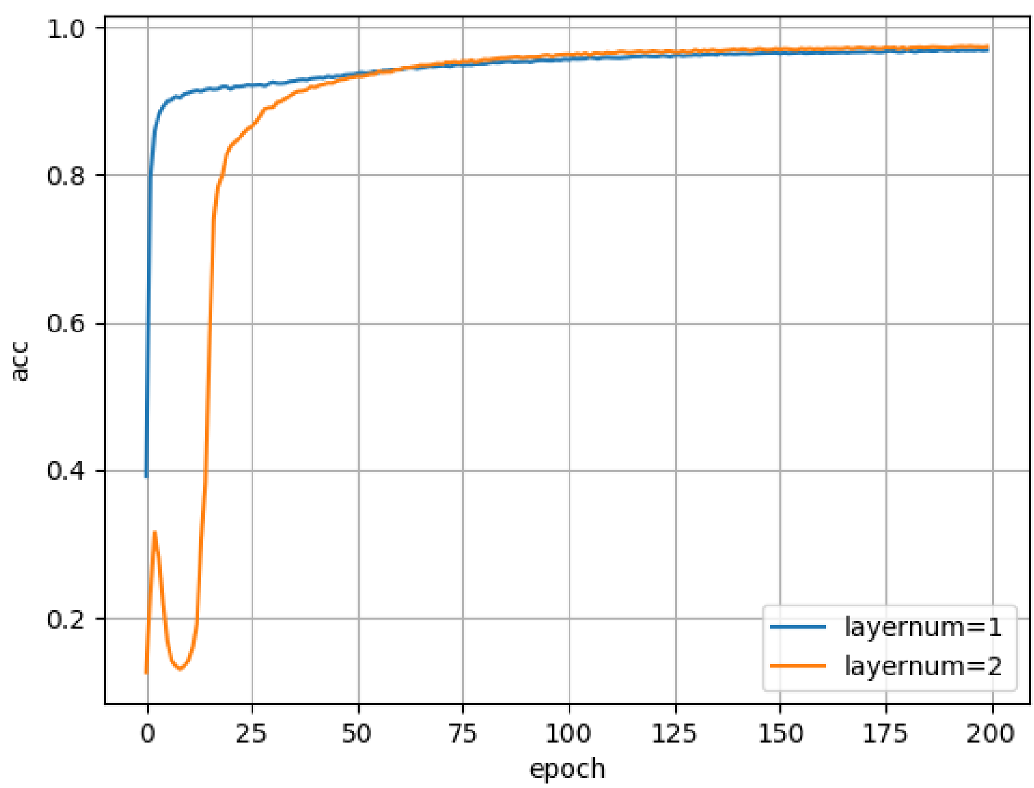
It costs much longer time for the 2-layer `FocalLoss` network to converge. Still, the final accuracy doesn't outperform the 1-layer network.

Same patterns occurs when we choose `Swish` as the activate function:

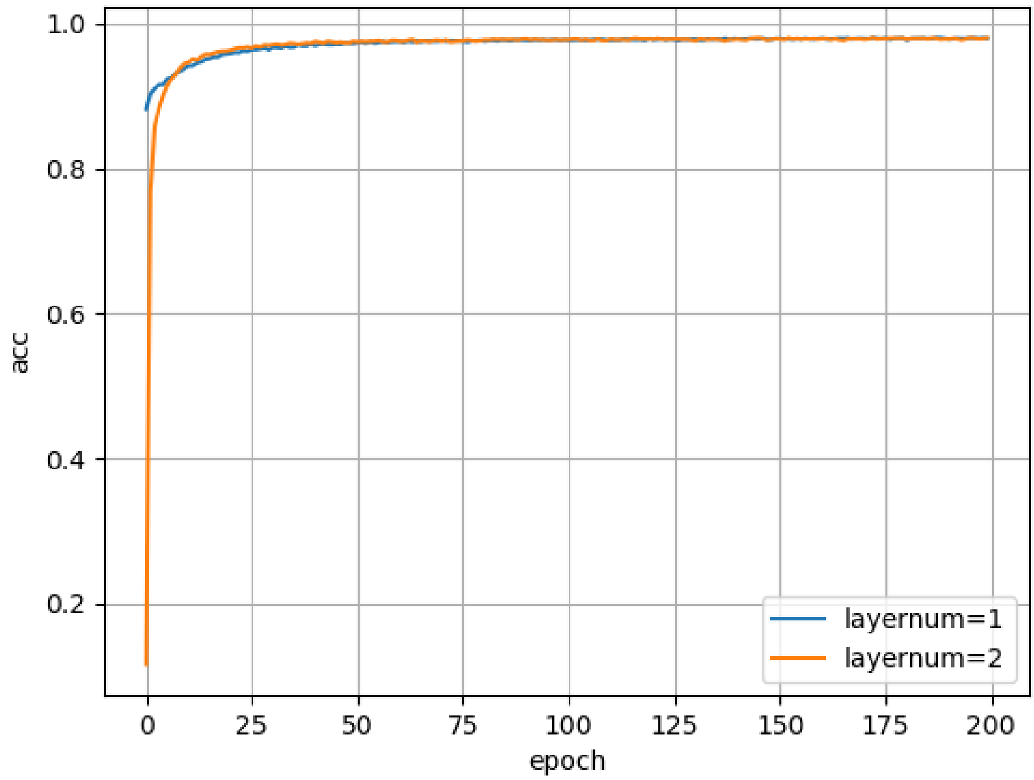
MSELoss:



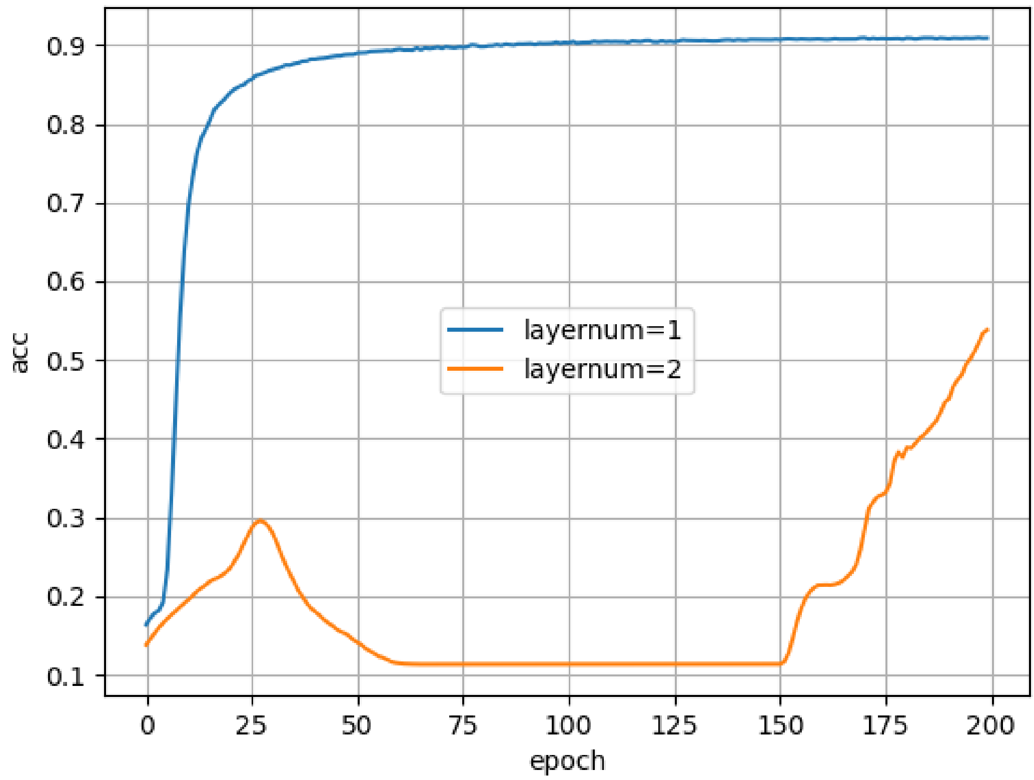
SoftmaxCrossEntropyLoss:



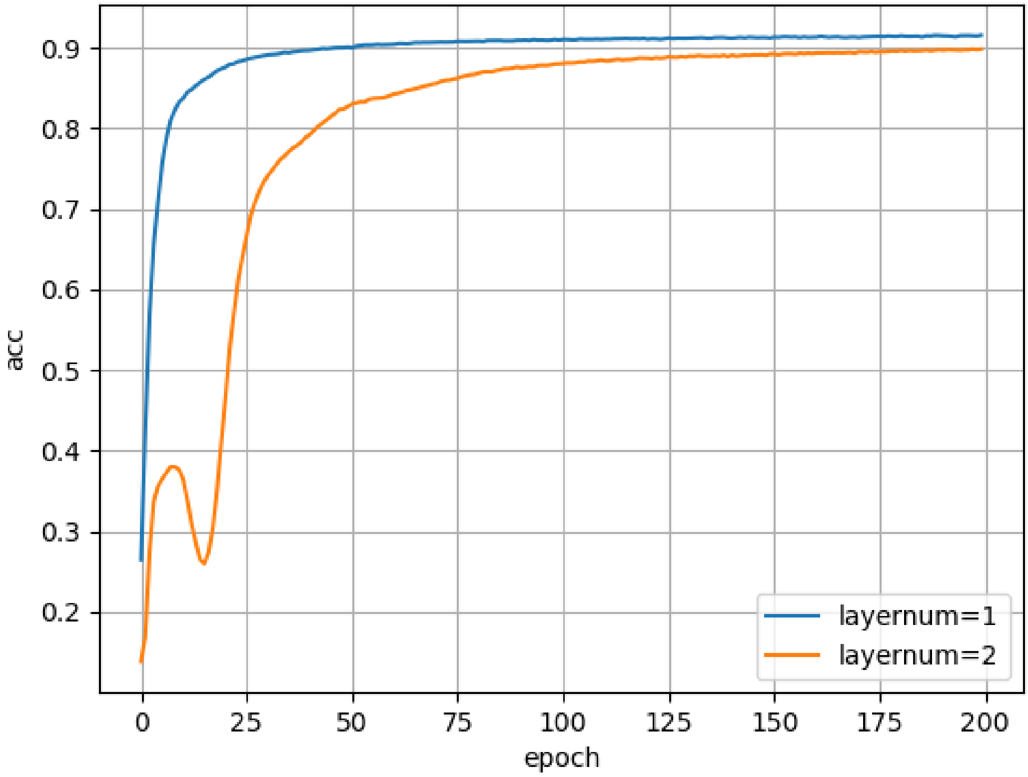
HingeLoss:


1.0
0.8
0.6
0.4
0.2
acc
0
25
50
75
100
125
150
175
200
epoch
layernum=1
layernum=2

FocalLoss


0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
acc
0
25
50
75
100
125
150
175
200
epoch
layernum=1
layernum=2

In the `Selu` network however, the 2-layer `FocalLoss` network had a regular performance. The test accuracy reached 90%, although it did not outperform the 1-layer network. The curves are as below:



5. Stablizing tactics

When calculating the softmax function:

$$P(t_k = 1|x) = \frac{e^{x_k}}{\sum_i e^{x_i}}$$

Often when x_i is too large, e^{x_i} will exeed the maximum limit of the `np.float`, causing the calculation result to be `nan`, and interrupts the training process. To prevent this, we note the fact that:

$$\frac{e^{x_k}}{\sum_i e^{x_i}} = \frac{e^{x_k - max_i\{x_i\}}}{\sum_i e^{x_i - max_i\{x_i\}}}$$

while $x_i - max_i\{x_i\} \leq 0$, so $e^{x_i - max_i\{x_i\}} \in (0, 1]$. Thus the storage exceeding is avoided.

6. Conclusion

From the above experiments, we make the following conclusions:

1. The best activate function for this task is the `Gelu` function, although it's the most time-consuming. `Swish` function is faster for calculation, and the accuracy is only a bit lower.
2. The best loss function for this task is the `HingeLoss` function.
3. The significance of the loss function on the final model accuracies is much higher than that of the activate function. Meanwhile it makes less influence on the training time. So choosing a good loss function is more important.

4. The 2-layer networks cost more epochs to converge, spend more time to train each epoch, and gain little or no improvement on the accuracies. So the 1-layer network suit this task better.