Tianxiao Zhao

tzh@kth.se

[redacted]

Assignment 3 – Optional Part

(i) Optimization

1) Adding a random jitter to training data

I add a random jitter (mean: 0, standard deviation: 0.15) for each mini-batch before gradient calculation. Other parameter settings are the same with the best performing 3-layer network mentioned before (eta = 0.3423, lambda = 7.4862e-06). It turns out that this trick could actually bump up the performance of the network to some degree, but the amplitude of the jitters needs to be chosen wisely. Otherwise, it may ruin the network. The test accuracy increases from 45.31% to 46.25%. The evolution of loss function is shown below.

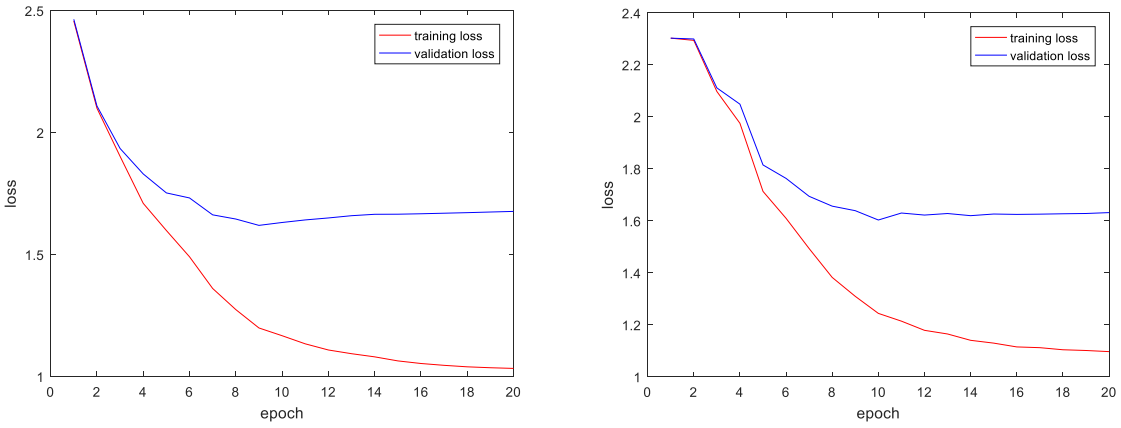

Fig. 1: Evolution of loss function as epoch increases
(Left: no jitters; Right: with jitters)

2) Cross-validation

Another trick I tried is cross-validation. The 3-layer network is trained for a longer time, e.g. 70 epochs, and all the validation data is used for checking whether overfitting happens. The figure below shows how the training and validation loss change as epoch increases. We can see that the validation loss first decays quickly and then goes up a little bit after certain time point (epoch = 12). To overcome overfitting by early stopping, the network is trained for ~12 epochs. Compared with the best network trained before, the test accuracy doesn't improve much (45.31% -> 45.37%). It seems that cross-validation doesn't work well under this situation.

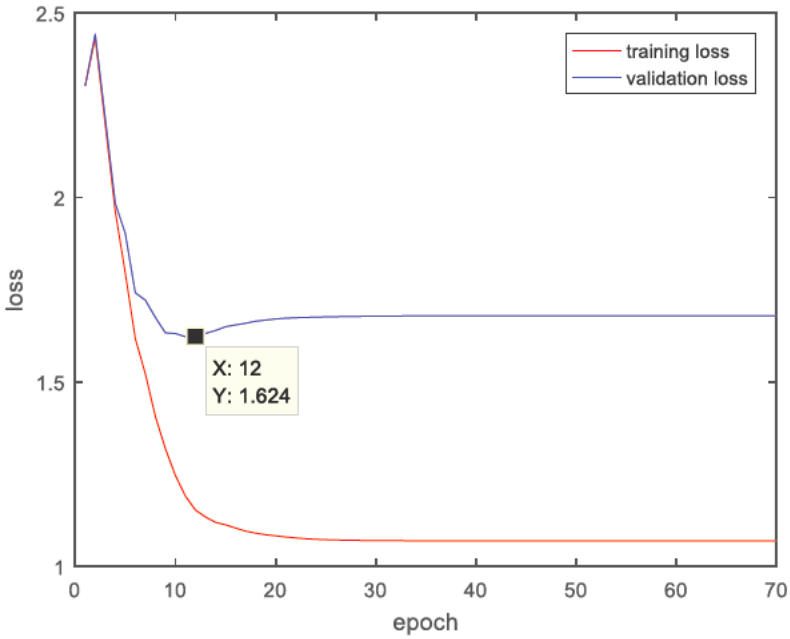

Fig. 2: Evolution of loss function as epoch increases (finding the stop point)

Tab. 1: Performance of networks using early stopping

epochs	11	12	13	14
Training accuracy	57.22%	58.50%	60.62%	62.48%
Test accuracy	44.28%	44.54%	44.79%	45.37%
Improvement	-1.03%	-0.77%	-0.52%	+0.06%

3) Increasing hidden layers

The 3-layer network could also be expanded to a 4-layer network. Here I add another hidden layer with 20 nodes, and two rounds of parameter search (coarse and fine search) are conducted. The five parameter settings which make the networks perform well on the validation data are picked out, and tested with test data. The results are listed in the following table. The results indicate that adding hidden layers cannot improve generalization due to an increased model complexity.

Tab. 2: Performance of the 4-layer networks

eta	0.2539	0.1781	0.8564	0.2621	0.2022
lambda	1.8618e-5	9.5001e-6	2.8071e-4	9.5999e-6	2.5360e-07
Training accuracy	59.58%	55.85%	57.01%	58.30%	57.15%
Test accuracy	44.69%	43.83%	44.59%	44.92%	44.66%
Improvement	-0.62%	-1.48%	-0.72%	-0.39%	-0.65%

(ii) Various Activation

Since the ReLu non-linear activation function may be faced with the "dying ReLu" problem, I replace it by a leaky ReLu activation function, which is max $(0.01s, s)$. And when propagating the gradient vector to the previous layer, $g \quad g \cdot diag(Ind(s > 0))$ should be changed to $g \quad g \cdot [diag(Ind(s > 0)) + 0.01diag(Ind(s < 0)$. Here, I test the performances of networks with leaky ReLus and compare them to the networks trained with ReLu activation function. All the networks are trained for 20 epochs. Tab. 3 shows the results of comparison, and from the table we can see that using leaky ReLu can slightly bring an improvement of the test accuracy. And sometimes the network with an original ReLu function outperforms that with a leaky one. In this sense, leaky ReLu may be a good alternative for activation function.

Tab. 3: Comparison between networks with ReLu and with leaky ReLu

eta	0.3423	0.4138	0.8745	0.3850	0.3595	0.1499
lambda	7.4862e-6	2.8193e-5	8.1984e-4	1.3144e-4	7.7693e-6	7.2835e-4
ReLu	45.31%	45.24%	45.31%	44.99%	45.24%	44.59%
Leaky ReLu	45.78%	44.96%	45.50%	45.11%	44.95%	45.14%
improvement	+0.47%	-0.28%	+0.19%	+0.12%	-0.29%	+0.55%

(iii) Best Performance

The best performance of the network over test dataset is achieved by Bagging. Here, I use the five batch data as training data, and randomly generate 6 bootstrap duplicates (each has a size of 10000). Each bootstrap duplicate is used for training one base classifier, which is a 3-layer network with 50 nodes in the first hidden layer and 30 in the second one, in a parallel way. The networks are all trained for 30 epochs. And the final classification is decided by a majority vote from the 6 base classifiers.

As for the parameter settings, I sort out six groups of parameters in Tab. 4 which give the top six performances during the fine search process. Besides, leaky ReLu activation function and random jitters are applied to enhance the network's generalization. The final classifier could reach a **54.8%** test accuracy.

Tab. 4: Parameter settings for the six base classifiers (3-layer network)

No.	1	2	3	4	5	6
eta	0.4138	0.3850	0.8745	0.1499	0.5264	0.3595
lambda	2.8193e-5	1.3144e-4	8.1984e-4	7.2835e-4	6.4889e-4	7.7693e-6