DD2424 Assignment2 Optional Part

Shuyuan Zhang shuyuanz@kth.se

April 2019

1 Optimize the performance of the network

In this part I choose to use 4 methods to optimize my network.

(a) Do a more exhaustive random search for hyper-parameters.

(b) Increase the number of hidden nodes.

(c) Augment training data by applying random jitters.

(d) He initialization of parameters.

1.1 Search for hyper-parameters

I did a more exhaustive random search on parameters like the amount of regularization, the length of the cycles, number of cycles etc...
This time, the range of lambda is 1e-5 to 1e-3. The number of cycles is chosen randomly from {3, 4, 5}. Because I used 49000 images as the training set, n_s was chosen randomly from {490, 980, 1960} while batch size is 100.
I ran ~ 50 tests and the best parameters are:

$$lambda = 1.3463e-04, n\_s = 980, num\_cycles = 4(16\ epochs)$$

With:

$$test\_accuracy = 0.5204$$

1.2 Increase the number of hidden nodes

Increasing the number of hidden nodes may be a good choice when we want to optimize the performance of the network. If a network has more hidden nodes, then it is able to approximate some more complicated functions.
I increased the number of hidden nodes from 50 to higher values. Also, I increased the amount of regularization accordingly to prevent the network from over-fitting. 49000 images were used as training set.
The parameters are:

$n\_batch = 100, n\_epoch = 12, n\_s = 980, eta\_min = 1e-5, eta\_max = 1e-1$

# of hidden nodes	lambda	Test accuracy
50	0.001	0.5153
100	0.001	0.5322
200	0.002	0.5476
500	0.005	0.556

Table 1: Number of hidden nodes

I achieved the highest accuracy of 0.556 with 500 hidden nodes.

1.3 Random jitters

To augment the training data, I applied a random Gaussian jitter to each image in the mini-batch before doing the forward and backward pass:

$$jitter = std * randn(size(Xbatch));$$

The parameters are:

$n\_batch = 100, n\_epoch = 12, n\_s = 980, eta\_min = 1e-5, eta\_max = 1e-1, lambda = 0.001$
The results with different *std* settings are shown below:

std	Test accuracy
0 (no jitter)	0.5153
0.001	0.5157
0.005	0.5217
0.01	0.5175
0.05	0.5204
0.1	0.5167

Table 2: Jitter with different std

From the table we can conclude that adding jitter to images can improve the performance of our network. I achieved $\sim 0.75\%$ improvement on test set by using augmented data set.

1.4 He initialization

The original version of the network used Xavier initialization. I applied He initialization instead and compared their effects on test accuracy.

$$W_{i,lm} \sim \mathcal{N}(0; \sigma^2)$$

$$\sigma = \sqrt{\frac{2}{n_{in}}}$$

The parameters are:
$n\_batch = 100, n\_epoch = 12, n\_s = 980, eta\_min = 1e-5, eta\_max = 1e-1, lambda = 0.001$

Xavier test accuracy	He test accuracy
0.5153	0.5182

Table 3: Different initialization methods

He initialization can slightly improve the performance of the network($\sim 0.3\%$).

1.5 Using all methods

In this section I applied method (b), (c) and (d) together to achieve the best performance.
I didn't used results from method (a) directly because some of the hyper parameters found may conflict with our requirements (more nodes need more regularization). So some newly found hyper parameters were used in this part.
The parameters are:
$m = 500\_nodes, lambda = 0.005, size\_train = 49000$
$n\_batch = 100, n\_epoch = 12, n\_s = 980, eta\_min = 1e-5, eta\_max = 1e-1$
With random jitter $std = 0.005$ and He initialization activated.

$$Test\_accuracy = 0.5625$$

The final test accuracy is 0.5625.
Among all the methods used, adding hidden nodes brought the largest gains. Other methods also brought slight gains, but adding nodes seems to be very efficient.

2 Setting eta_min and eta_max

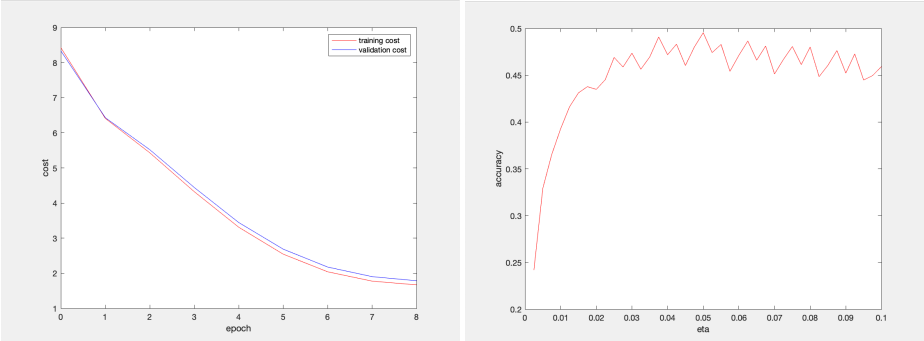
According to "Cyclical learning rates for training neural networks", we can figure out reasonable minimum and maximum boundary values by running the model for several epochs while letting the learning rate increase linearly between low and high LR values.

I used the model in part 1.5 as the baseline:
$m = 500\_nodes, lambda = 0.005, size\_train = 49000$
$n\_batch = 100, n\_epoch = 40, n\_s = 980, eta\_min = 1e-5, eta\_max = 1e-1$
With random jitter $std = 0.005$ and He initialization activated.

I performed a "LR range test" on this network. I ran 8 epochs and n_s was set to 3920, which equals to $(49000/batch\_size) * 8$. In this way the learning

rate will increase linearly from $1e-5$ to $1e-1$ throughout the whole training process.

Then I plotted the accuracy as a function of increasing learning rate to find good values for boundaries. Results are shown in figure 1:



(a) Cost plot of LR range test

(b) Accuracy as a function of increasing learning rate

Figure 1: Curves of LR range test

From the figure we may notice that the accuracy reached its highest and then started to fall when eta is around 0.05. So it is reasonable to set upper bound $= 0.05$. Also, the model starts converging right away, so it is reasonable to set lower bound $= 1e-5$.

To further specify the effect of choosing an appropriate LR range, I trained two networks with different LR bounds for 10 cycles. Their parameters are specified below:

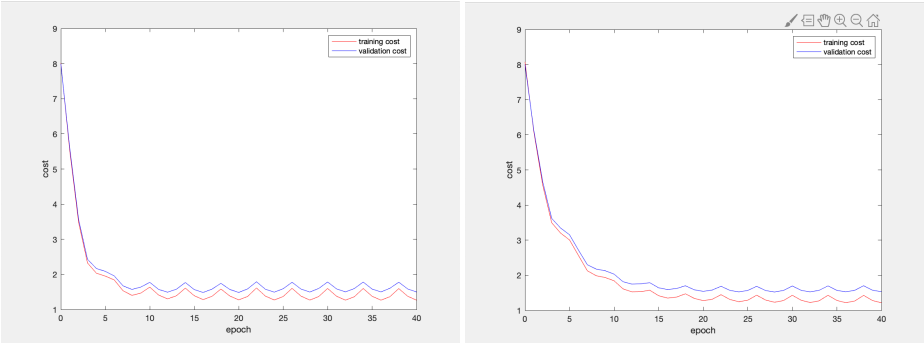
Network1:
$m = 500\_nodes, lambda = 0.005, size\_train = 49000$
$n\_batch = 100, n\_epoch = 40, n\_s = 980, eta\_min = 1e-5, eta\_max = 1e-1$
With random jitter $std = 0.005$ and He initialization activated.

Network2:
$m = 500\_nodes, lambda = 0.005, size\_train = 49000$
$n\_batch = 100, n\_epoch = 40, n\_s = 980, eta\_min = 1e-5, eta\_max = 0.05$
With random jitter $std = 0.005$ and He initialization activated

The performance of the networks after 10 cycles of training are:

$$Network1: \quad test\_accuracy = 0.5647$$

$$Network2: \quad test\_accuracy = 0.5723$$



(a) Cost plot of network 1(before) (b) Cost plot of network 2(after)

Figure 2: Curves of networks before/after setting LR range

From the results we may conclude that with newly found settings of eta_max and eta_min the performance our network with 500 hidden nodes improved. The accuracy on the test set increased $\sim 0.8\%$

Figure 2 shows the training cost plot for the network before/after using newly found settings. After I re-scaled the LR range, the convergence speed of my network became slower because learning rate decreased. But the curve became smoother after I used newly found settings because now the network has a lower LR upper bound so the parameters won't jump too much between updates, thus the degree of fluctuation decreases.