DD2424 Assignment3 Optional Part

Shuyuan Zhang shuyuanz@kth.se

April 2019

1 Brief Introduction

1.1 Methods

In this part I choose to use 5 methods to optimize my network.

(a) Do a more exhaustive random search for the amount of regularization.

(b) Do a more thorough search to find a good network architecture.(Adjusting number of layers and number of nodes)

(c) Apply batch normalization to the scores after the non-linear activation function has been applied.

(d) Augment training data by applying random jitters.

(e) Use Leaky ReLu as activation function.

1.2 Baseline networks

In some of the tests, I will use the following 3 networks as baseline networks to test the effectiveness of optimization methods.

3-layer network:
$train\_size = 45000,\ n\_batch = 100,\ eta\_min = 1e-5,\ eta\_max = 1e-1,\ lambda = .005,\ n\_cycles = 2,\ n\_s = 5 * 45,000/n\_batch,\ num\_layers = 3,\ hidden\_node = [50,\ 50]$ with He initialization and BN.

6-layer network:
$train\_size = 45000,\ n\_batch = 100,\ eta\_min = 1e-5,\ eta\_max = 1e-1,\ lambda = .005,\ n\_cycles = 2,\ n\_s = 5 * 45,000/n\_batch,\ num\_layers = 6,\ hidden\_node = [50,\ 30,\ 20,\ 20,\ 10]$ with He initialization and BN.

9-layer network:
$train\_size = 45000,\ n\_batch = 100,\ eta\_min = 1e-5,\ eta\_max = 1e-$

1, $lambda = .005$, $n\_cycles = 2$, $n\_s = 5 * 45,000/n\_batch$, $num\_layers = 9$, $hidden\_node = [50, 30, 20, 20, 10, 10, 10, 10]$ with He initialization and BN.

2 Tests

2.1 Random search for the amount of regularization.

I did two rounds of random search to find the best amount of regularization of the 3 baseline networks.

Coarse search: search lambda randomly from $1e-05$ to $1e-01$

Fine search: adjust search range according to the results of coarse search

3-layer network: from $1e-03$ to $1e-02$.
6-layer network: from $1e-03$ to $1e-02$.
9-layer network: from $1e-03$ to $1e-02$.

After around 25 trials in each of these search, the best test accuracy and corresponding lambda are shown in table below:

Network	lambda	Test accuracy
3-layer network	0.0029	0.5316
6-layer network	0.0020	0.5179
9-layer network	0.0074	0.502

Table 1: Tuning amount of regularization

2.2 Search for a good network architecture

First, let's try the 3 baseline networks and test their performance:

Network	Test accuracy
3-layer network	0.5305
6-layer network	0.5071
9-layer network	0.4871

Table 2: Baseline network performance

From the table we can notice that a deeper network doesn't necessarily mean a higher performance. A more complex structure may lead to a worse accuracy on the test set.

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.
So I changed the number of hidden nodes of the 3 baseline network, while all other parameters remain the same.
3-layer:[500, 100]
6-layer:[500, 200, 100, 50, 20]
9-layer:[500, 250, 100, 50, 50, 30, 20, 10]

Network	Test accuracy
3-layer network	0.5703
6-layer network	0.5795
9-layer network	0.5716

Table 3: Baseline network performance after adding nodes

Table 3 shows that by adjusting the number of nodes carefully, a deeper network can have better results than a shallower network. Based on this, I tested many other combinations of parameters and achieved the best performance of:

$$Test\_accuracy = 0.5926$$

With a 6-layer network [1000, 500, 250, 100, 50] and all other parameters remain the same as the baseline 6-layer network.

Keep increasing the number of layers or nodes may improve the accuracy more. But because of the limitation of my computing device, I cannot add more layers or nodes and my final accuracy is 0.5926 by tuning number of layers and nodes.

2.3 Apply BN to the scores after the non-linear activation function

As suggested in the assignment description, it has been empirically reported in several works that you get better performance by the final network if you apply batch normalization to the scores after the non-linear activation function has been applied. So I changed the order of computations in functions *intervalues* and *ComputeGradients* and put them into new functions *intervalues1* and *ComputeGradients1*. Details can be seen in file *Assignment3_bonus*.
I swapped the order of ReLu and BN in forward and backward pass.
I used the 3 baseline network settings. Some test accuracy are shown below:

std	3-layer	6-layer	9-layer
BN-ReLu(original)	0.5278	0.5164	0.492
ReLu-BN	0.5382	0.5267	0.5091

Table 4: Some test results: Change order of BN and ReLu layer

The table tells us that applying batch normalization to the scores after the non-linear activation function indeed brings about some improvements. On average it can improve the accuracy by $0.5\% \sim 1.5\%$.

2.4 Random jitters

By adding random jitters to the training data, the network can be more robust and become less sensitive so they can have a better result.
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:

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

I used the 3 baseline network settings and some test accuracy with different *std* settings are shown below:

std	3-layer	6-layer	9-layer
0 (no jitter)	0.5265	0.5134	0.5018
0.001	0.5293	0.5146	0.5061
0.005	0.5276	0.5175	0.4955
0.01	0.5344	0.5243	0.5136
0.05	0.5326	0.5257	0.5048
0.1	0.524	0.5203	0.4921

Table 5: Some test results: Jitter with different std

From the table we can conclude that adding jitter to images can slightly improve the performance of our network. Sometimes because of the random numbers, adding jitter even has a worse performance than the original version. On average I achieved $0.5\% \sim 1\%$ improvement on test set by using augmented data set when $std = 0.01$.

2.5 Leaky ReLu

Relu is a popular activation function, but it may cause the "Dying ReLu problem". So in this part I tried to use Leaky ReLu as the activation function. Leaky ReLu activation function can be expressed by:

$$f(x) = \max(\alpha * x, x),\ 0 < \alpha < 1$$

When $x > 0$, $f(x) = x$ and when $x < 0$, $f(x) = \alpha * x$.
Then the back propagation process of the function becomes:

$$G = G. * ((h\{l-1\} > 0) + \alpha * (h\{l-1\} < 0))$$

I tested two Leaky ReLu activation functions with different α. I used the 3 baseline network settings and some test accuracy are shown below:

α	3-layer	6-layer	9-layer
0 (Relu)	0.5249	0.5122	0.5003
0.01	0.5242	0.527	0.5036
0.1	0.5294	0.5199	0.4974

Table 6: Some test results: Leaky ReLu with different α

Sometimes Leaky ReLu outperforms Relu, while sometimes ReLu is better than Leaky ReLu. So at least in my test cases there's no much difference between these two methods, but we may say that it is still worth to try Leaky ReLu when you want to optimize your network.

3 Conclusion

Among all the methods applied, do a more thorough search to find a good network architecture brought the largest gains. By tuning the number of layers and the number of hidden nodes in each hidden layer, I can reach $\sim 60\%$ test accuracy by using a 6-layer network with [1000, 500, 250, 100, 50] hidden nodes.

Also, I tried to combine methods mentioned in this report as much as possible. I can achieve the best test accuracy of 0.6010 by using the 6-layer network with [1000, 500, 250, 100, 50] hidden nodes with random jitters and using Leaky ReLu activation after 3 cycles of training(30 epochs).