Deep Learning Assignment2

Mengdi Xue, mengdix@kth.se

April 2018

1 Computing Gradients

First, I modified function "EvaluateClassifier" and function "ComputeCost" to realize the new cost function as following:

$$J = \frac{1}{|D|} \sum_{(x,y)\in D} -log(y^T p) + \lambda \sum_{l=1}^{2} \sum_{i,j} W_{l,ij}^2$$

And here "p" is calculated differently according to the structure of two layer networks:

$$s_1 = W_1 x + b_1$$

$$h = max(0, s_1)$$

$$s = W_2 h + b_2$$

$$p = softmax(s)$$

Then, the steps to compute the gradients were also modified as following:

1. Set all entries in $\frac{\partial L}{\partial b_1}$, $\frac{\partial L}{\partial b_2}$, $\frac{\partial L}{\partial W_1}$ and $\frac{\partial L}{\partial W_2}$ to zero.
2. for each $(x, y) \in D$, do the following computations in order:

$$g = -(y - p)^T$$

$$\frac{\partial L}{\partial b_2} += g, \frac{\partial L}{\partial W_2} += g^T h^T$$

$$g = gW_2$$

$$g = g\text{diag}(\text{Ind}(s_1 > 0))$$

$$\frac{\partial L}{\partial b_1} += g, \frac{\partial L}{\partial W_1} += g^T x^T$$

3. Divide all entries by $|D|$.
4. $\frac{\partial J}{\partial W_i} = \frac{\partial L}{\partial W_i} + 2\lambda W_i$, $\frac{\partial J}{\partial b_i} = \frac{\partial L}{\partial b_i}$

After the above implementation, I compared the result of "ComputeGradients"

with the result of the numerically computed gradient and got their relative error according to the formula:

$$\frac{|g_a - g_n|}{max(eps, |g_a| + |g_n|)}$$

The result were as in Figure 1. Since they are very small numbers, I think the gradient computations are bug free.

error_b1	4.7324e-07
error_b2	1.6666e-07
error_W1	1.8567e-06
error_W2	7.7897e-08

Figure 1: Relative error between g_a and g_n

Then I tried to overfit to the training data with 100 examples and 200 epoches. The parameter settings are $lambda = 0; n\_epochs = 200; n\_batch = 10; eta = .01$. The result loss figure is as Figure 2.

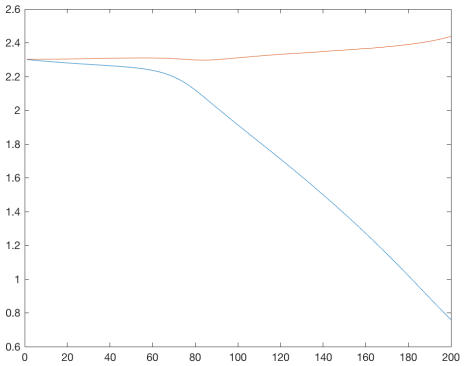

Figure 2: Loss: $lambda = 0; n\_epochs = 200; n\_batch = 10; eta = .01$

2 Momentum Term

After add momentum term, the cost after 200 epoches with the same parameters before are as in Figure 3. We can see from Figure 2, it takes about 180 epoches to reach $cost = 1$. But with momentum in Figure 3, when $\rho = 0.9$, it needs only about 50 epoches to reach the same cost. So it can speed the training by at least 2 times. Also, for different ρ, we can conclude that the bigger ρ is, the faster the training is. But too large ρ may make the training too fast, so I chose $\rho = 0.9$ to apply in the following steps.

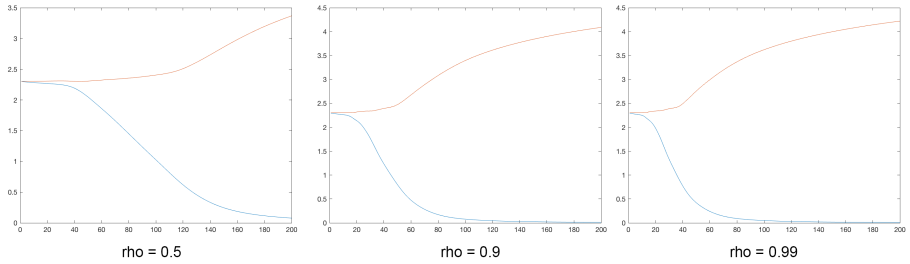

Figure 3: Loss with momentum term

3 Coarse Search

The search range of λ and η (under log scale) is as following:

$$lmin = -6; lmax = -1; emin = log10(0.003); emax = log10(0.5)$$

The number of epoches used for training is 10. The hyper-parameter settings for the 3 best performing networks are as in Table 1.

Table 1: 3 Best parameters for coarse search

	λ	η
1	0.00068687	0.0968951
2	0.00028498	0.106499
3	0.000558383	0.0961255

4 Fine Search

The search range of λ and η (under log scale) is as following:

$lmin = log10(4e{-}6); lmax = log10(4e{-}5); emin = log10(0.08); emax = log10(0.12)$

Table 2: 3 Best parameters for find search

	λ	η
1	2.24e-05	0.108993
2	1.24e-05	0.118811
3	3.91e-05	0.119100

I modified and chose this lambda range after several rounds of fine search because I found this range has the best performance and this was the last round

before I reach the 44% accuracy.
The number of epoches used for training is 15. The hyper-parameter settings for the 3 best performing networks are as in Table 2.
The best found setting here is the first setting. All the data in batch 1 were used to train it. It gets the accuracy of 44.42%, as in Figure 4. The loss is as in Figure 5.

```
18      % set training parameters
19 -    n_epochs=10; n_batch=230;  % 0.4442
20
21 -    lambda = 2.24e-05; eta = 0.108993;
22      % lambda = 3.81e-05; eta = 0.108799;  %0.4008
23      % lambda = 4.42e-06; eta = 0.119997;  %0.3988
```

命令行窗口

```
>> training
    0.4442
```

Figure 4: Best Found setting on batch 1 with 10 epoches

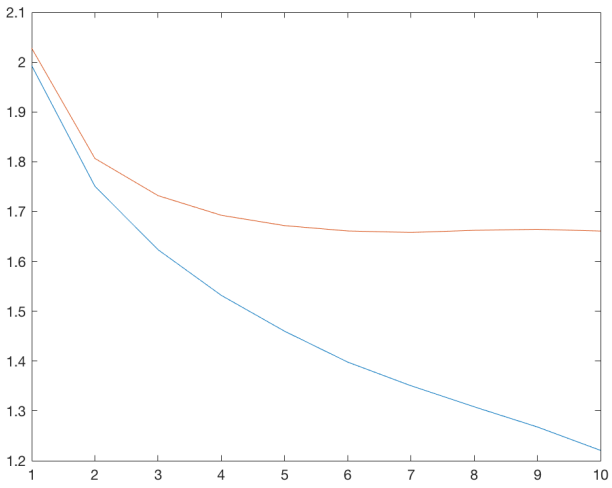

Figure 5: Train and validation loss

5 Best Found Setting on All Data

For the best found hyper-parameter setting, I trained the network on all the batch data, except for 1000 examples in a validation set, for 30 epochs.

It gets the accuracy of 50.36% on the test data, as in Figure 6. The training and validation cost after each epoch of training is as in Figure 7.

```
38      % set training parameters
39 -    n_epochs=30; n_batch=230;  % 0.4442
40
41 -    lambda = 2.24e-05; eta = 0.108993;
```

命令行窗口

```
>> training
    0.5036
```

Figure 6: Train and validation loss

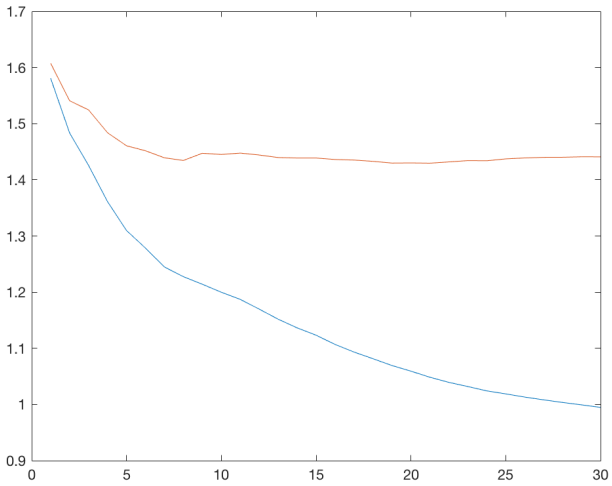

Figure 7: Train and validation loss