DD2424 - Assignment 3

Thi Thuy Nga Nguyen

1 Gradient computation check

The numerically computed gradients, g_n, and the analytically computed gradients, g_a, were compared by using relative error

$$error = \frac{|g_a - g_n|}{max(\epsilon, |g_a| + |g_n|)},$$

where $\epsilon = 10^{-30}$. The network has 2 convolutions layers in which the first one has n_1 filters of size $(d \times k_1)$ and the second one has n_2 filters of size $(n_1 \times k_1)$. He initialization was used for the model parameters, in particular, the filters were randomly drawn from Gaussian distribution with mean 0 and standard deviation $\sqrt{\frac{2}{10 \times k_1}}$ in layer 1 and $\sqrt{\frac{2}{n_1 \times k_2}}$ in layer 2; and the weight matrices W came from a Gaussian distribution with mean 0 and standard deviation $\sqrt{\frac{2}{\text{fsize}}}$, where fsize $= n_2(n_{len} - k_1 - k2 + 2)$, n_{len} is the number of columns of the input X. Since the input is sparse I set a bound 10 instead of $d = 28$ for std of the filters in layer 1.
Practically, I set the model hyper-parameter as the learning rate is 0.001, the number samples of a batch is 100, momentum term is 0.9 and the network structure $k_1 = k_2 = 5$, $n_1 = n_2 = 5$ and did the gradient computation check for first 6 names in dataset and $h = 1e - 5$, using mini-batch of size 1, I get

```
The number of error (relative error > 1e-6) of W: 0
The maximum of relative error of W: 4.940123577024272e-07
The number of errors (relative error > 1e-6) of filter 0: 0
The maximum of relative error of filter 0: 7.582209070388632e-07
The number of errors (relative error > 1e-6) of filter 1: 0
The maximum of relative error of filter 1: 8.724810501754798e-07.
```

When using mini-batch of size 2, I get

```
The number of error (relative error > 1e-6) of W: 2
The maximum of relative error of W: 2.9212056861892006e-06
The number of errors (relative error > 1e-6) of filter 0: 0
The maximum of relative error of filter 0: 2.9442325781691894e-07
The number of errors (relative error > 1e-6) of filter 1: 0
The maximum of relative error of filter 1: 3.92294992692926e-07.
```

And using mini-batch of size 3, I get

```
The number of error (relative error > 1e-6) of W: 0
The maximum of relative error of W: 9.539478143083176e-07
The number of errors (relative error > 1e-6) of filter 0: 1
The maximum of relative error of filter 0: 4.672309403308377e-06
The number of errors (relative error > 1e-6) of filter 1: 0
The maximum of relative error of filter 1: 5.108455966805446e-07.
```

The relative errors being almost very small show the precision for the gradients computation.

2 Extra measures for a more efficient implementation

To make the computations more efficient I used the equation (46) with matrix $M^{\text{input}}_{x_j,k_2}$ to compute equation (37) in backward pass. This change made training be faster. In particular, I trained a network of 10 filters of size 28×5 in layer 1 and 10 filters of size 10×3 in layer 2 with learning rate is 0.001, n_batch = 100, momentum term is 0.9 for 1 epoch ($\sim$200 updates). It took $\sim$50.5s in execution time before and $\sim$33.7s after the upgrade.

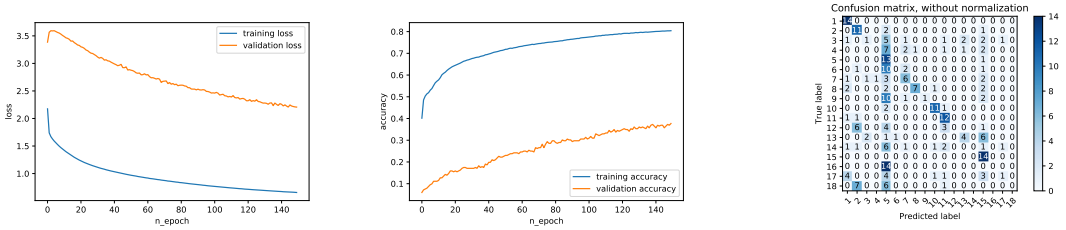
I tried using sparse matrix for $M^{\text{input}}_{x_j,k_1,n_1}$ as in construction but it did not bring the efficient gain.

3 Compensating for unbalanced training data

For compensating I use the second approach mentioned in the construction. Practically, for each epoch I randomly chose 18×59 samples from training set which has 59 samples for each class (59 is the number of samples of the smallest class). For training I used this set with batch of size 59 in mini-batch gradient descent.

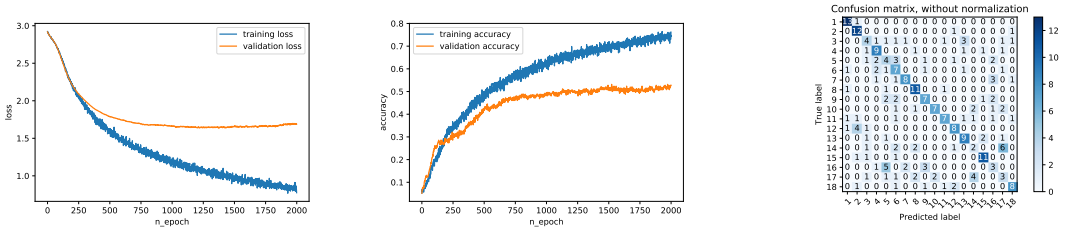
For comparison, I trained 2 convnets with the same $k_1 = 5, k_2 = 3$, $n_1 = n_2 = 20$ and hyper-parameter $\eta = 0.001$, momentum term $\rho = 0.9$.
The first network without compensating, n_batch=100 and after 150 epochs (corresponding to $\sim$30000 updates) gave 37.7% in the final validation accuracy. In this case, the loss, accuracy plots for training and validation set as well as the confusion matrix is shown in figure 1.
In the second network with compensating I trained with n_batch=59 and after 2000 epochs (corresponding to 36000 updates) gave 52.78% in the final validation accuracy. In this case, the loss, accuracy plots for training and validation set as well as the confusion matrix is shown in figure 2.

Without compensating for the unbalanced dataset (figure 1) the final accuracy is very different between training, $\sim$80% and validation 37.7%. From confusion matrix in



(a) (b) (c)

Figure 1: The loss (a), accuracy (b) after each epoch (∼200 updates) and the final confusion matrix (c) on validation set with $\eta = 0.001, n\_batch = 100$, and $n\_epochs = 150$.



(a) (b) (c)

Figure 2: The loss (a), accuracy (b) after each epoch (18 updates) and the final confusion matrix (c) on the validation set with $\eta = 0.001, n\_batch = 59$, and $n\_epochs = 2000$.

figure 1c we can see the accuracy is large (≈ 1) for classes 1, 5, 15 where there are a large number of samples. It seems that only large classes are learning.
In contrast, with compensating (figure 2) these final accuracies are closer than, $\sim 75\%$ in training and 52.78% in validation. The final confusion matrix on validation set in figure 2c shows that training is better in small class (for example, class 12, 18 that each of them has below 100 samples) and almost of classes are learning but the accuracy is decreasing in large classes as 5, 15.

4 Best performance

Using normal cost function (cross-entropy + softmax) for validation set I got 1.9846 with 54.76% in the accuracy. I trained a network with compensating unbalanced dataset as described above and $k_1 = 5, k_2 = 3$, $n_1 = 20, n_2 = 50$, hyper-parameter $\eta = 0.001$, momentum term $\rho = 0.9$, n_batch = 59 for 3000 epochs (corresponding to 54000 updates). The accuracy of each class is shown in table 1 and the final confusion matrix in validation set is shown in figure 3

Arabic	Chinese	Czech	Dutch	English	French	German	Greek	Irish	Italian
1.	0.9286	0.3571	0.5	0.4286	0.3571	0.7143	0.7143	0.5714	0.4286

Japanese	Korean	Polish	Portuguese	Russian	Scottish	Spanish	Vietnamese
0.8571	0.5714	0.5714	0.1429	0.7857	0.2143	0.3571	0.3571

Table 1: The accuracy of each class on the validation set after training 2 layers CNN with 20 filters 28×5 in layer 1 and 50 filters 20 in layer 2 and the hyper-parameter $\eta = 0.001, n\_batch = 59, \rho = 0.9$, and $n\_epochs = 3000$.

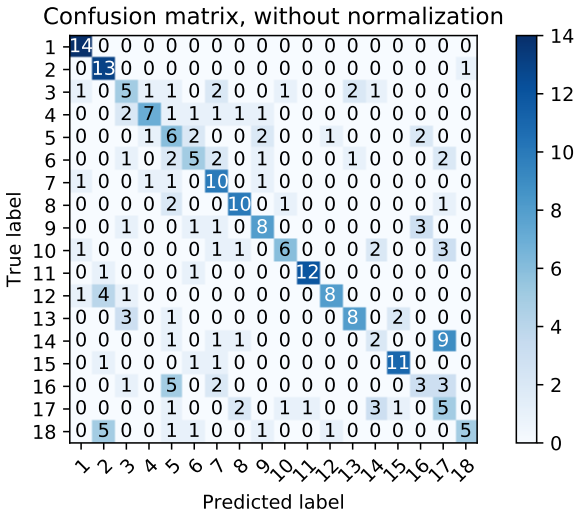


Figure 3: The final confusion matrix on the validation set after training 2 layers CNN with 20 filters 28×5 in layer 1 and 50 filters 20 in layer 2 and the hyper-parameter $\eta = 0.001, n\_batch = 59, \rho = 0.9$, and $n\_epochs = 3000$.

4.1 Test in real

I tested this network for the following names:

```
name_test = ['Nguyen','Herrera','Martinez','Bergstrom','Batakis','
    Fontbona']
```

and got the predict

```
Nguyen: Vietnamese
Herrera: Spanish
Martinez: Spanish
Bergstrom: English
Batakis: Greek
Fontbona: Italian
```

Three in six names are correct (Nguyen, Herrera, Batakis). Three of them can not be sure, Martinez comes from Mexico where they speak Spanish, Bergstrom is a Swedish

which there is not in the category and Fontbona is a Chilean who speaks Spanish but can be not a Spanish.

The probability vector output is shown below and in figure 4.

```
                 Nguyen       Herrera      Martinez     Bergstrom  \
Arabic     1.755429e-06  2.935361e-09  1.103962e-14  1.517484e-13
Chinese    1.248380e-07  2.353268e-12  2.736419e-13  4.095259e-13
Czech      2.085217e-02  2.527341e-03  7.830945e-06  2.509641e-02
Dutch      1.450255e-02  1.549228e-03  1.830929e-06  5.618185e-05
English    4.312408e-02  1.441998e-03  6.285375e-06  6.603984e-01
French     1.233186e-02  4.406504e-05  3.524951e-05  2.216746e-03
German     8.490800e-02  5.443868e-03  3.511246e-05  2.939586e-01
Greek      1.373575e-04  7.943437e-08  2.299047e-09  6.665894e-06
Irish      6.915342e-03  2.448532e-05  6.913226e-08  5.308482e-05
Italian    1.002758e-05  5.933148e-02  4.303584e-03  5.937652e-06
Japanese   4.430237e-05  3.654550e-07  5.195473e-08  3.957482e-08
Korean     9.428184e-06  1.280309e-14  3.886511e-18  8.328527e-17
Polish     5.652116e-03  3.442742e-03  2.327683e-07  1.003469e-05
Portuguese 1.613348e-09  2.637562e-02  2.728391e-07  1.429410e-07
Russian    8.934925e-02  1.210893e-04  6.744447e-06  1.819025e-02
Scottish   1.018798e-02  2.196759e-05  8.895319e-08  1.083571e-06
Spanish    6.776809e-06  8.996757e-01  9.956026e-01  6.423511e-06
Vietnamese 7.119669e-01  2.405120e-19  6.652285e-19  9.049097e-19

                 Batakis      Fontbona
Arabic     8.682465e-04  1.267745e-09
Chinese    3.195545e-11  1.044664e-17
Czech      1.182259e-02  8.295815e-03
Dutch      1.387131e-03  2.439728e-04
English    5.461334e-04  2.295327e-03
French     4.444219e-08  3.256630e-05
German     7.023130e-06  5.585226e-03
Greek      9.744414e-01  4.661474e-05
Irish      2.677979e-09  2.353190e-05
Italian    1.955839e-07  8.812227e-01
Japanese   2.516131e-03  4.901186e-03
Korean     2.876879e-17  9.465274e-18
Polish     4.828730e-05  1.142154e-03
Portuguese 6.096890e-07  5.137021e-04
Russian    8.362179e-03  2.043457e-03
Scottish   3.056144e-12  2.183848e-06
Spanish    9.123937e-09  9.365152e-02
Vietnamese 1.650747e-19  5.633498e-19
```

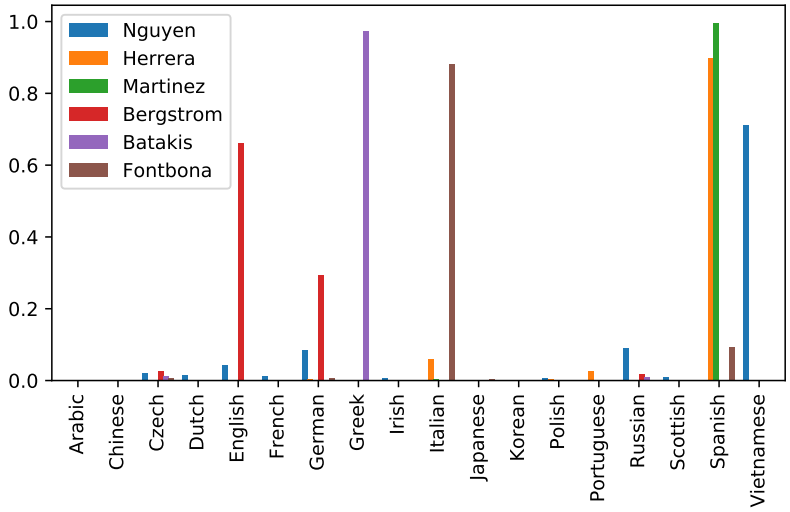


Figure 4: The probabilities vector output of testing CNN network with 2 layers, 20 filters 28×5 in layer 1 and 50 filters 20 in layer 2, and the hyper-parameter $\eta = 0.001, n\_batch = 59, \rho = 0.9$, and $n\_epochs = 3000$.