DD2424 Deep Learning in Data Science
Assignment 3

Rui Qu
rqu@kth.se

April 29, 2019

1 Introduction

Train a ConvNet to predict the language of a surname from its spelling follow the instruction of Assignment3. The code contains following components:

Preparing Data
By *Class Dataprocess*, read in training data *ascii_names*,*Validation_Inds* to get *n_len*, *d* and validation set. One-hot-encoding for all dataset: Training, Validation and Test random names.

Convolution as Matrix multiplication
By functions *MFMatrix, MXMatrix*, create filters and input matrix to help compute gradient.

Mini-batch gradient descent with momentum
Details in instruction of Assignment 2.

2 Gradient Check

First numerical derivative:

$$f'(x') = \frac{\partial f(x)}{\partial x}|_{x=x'} \tag{1}$$

First analytic derivative:

$$f'(x') \approx \frac{f(x'+\delta) - f(x'-\delta)}{2\delta} \tag{2}$$

Then compare the two gradient via relative error:

```
def rel_err(x, y):
 e=np.max(np.abs(x-y)/np.maximum(1e-8,np.abs(x)+np.abs(y)))
 return e
```

Differences and relative error between numerical and analytic gradient of Filter1, Fliter2, Weights for the first batch, we can see the error is less than 1e-6. I could say that I successfully managed to write the functions to correctly compute the gradient.

Gradient	Filter1	Filter2	Weights
Differences	6.8599e-07	2.3699e-07	7.6729e-07
Relative error	1.8208e-09	1.0594e-09	1.6891e-09

3 Algorithm

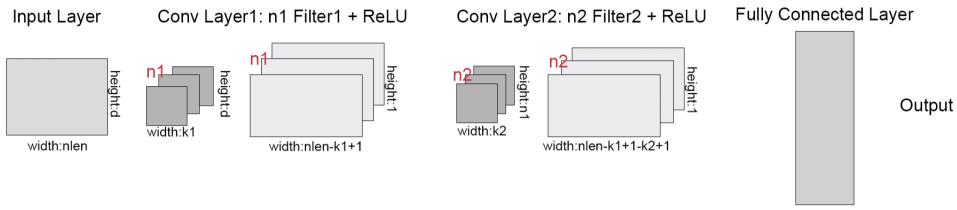
The code is written in Python. Pseudo-code as following:

Algorithm 1 ConvNet for text classifaction

1: Data preprocess: read in dataset and one-hot encoding
2: Generate random $F1(size : d \times k_1 \times n_1)$, $F2(size : n_1 \times k_2 \times n_2)$, $w(size : K, n_2 \times (n_len - k_1 + 1 - k_2 + 1))$ by He Initialization
3: Generate MFMatrix and MXMatrix
4: Generate mini-batch Xbatch for data and Ybatch for label
5: **repeat**
6: Take j_{th} batch in mini-batches, using $argmax(SOFTMAX)$ to predict the label Pbatch of each data in Xbatch
7: Compute Cross-Entropy Loss and minimize cost function
8: Compute gradient with momentum, update F1, F2, W and MFMatrix
9: **until** n-epochs
return Final Accuracy, Best epoch &cost; plotting of training &validation cost and confusion matrix

NB: Cross Entropy Loss with Softmax function and its derivative [1] $\frac{\partial L}{\partial w_i} = p_i - y_i$

Here I drew an image with photoshop to help me keep the structure of ConvNet in mind.



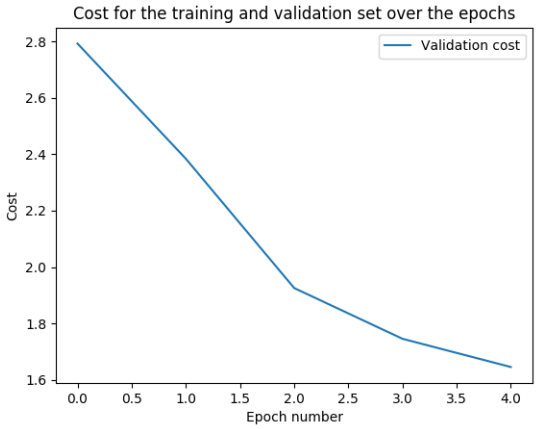
4 Compensating for unbalanced training data

The number of names in 18 different classes is 1806, 236, 477, 260, 3312, 246, 641, 186, 203, 633, 880, 86, 123, 69, 8465, 91, 264, 68, in which 15th class is the biggest,18th is the smallest.

Hyper-parameter setting: (n1, k1, n2, k2)=(50, 5, 50, 3), eta=0.01(decaying rate=0.1), rho=0.9, batch size=100, ite=1000

4.1 Non-compensated

Validation loss:



Final Accuracy: 0.5449

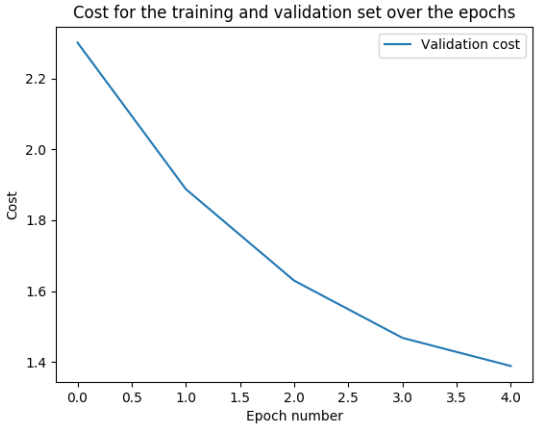
Confusion Matrix[1]:

```
[[1100    0    0    0  595    0   19    0    0   19   63    0    0    0   10    0    0    0]
 [  28   27    7   12   51   13   50    0    8    2   22    0    0    0   13    0    2    1]
 [  66    0    4    2  206    1   49    1    1    5   62    0    0    0   77    1    2    0]
 [  17    1    0    7  140    3   29    4    2    3   21    0    0    0   32    0    0    1]
 [ 119   14    5    6 2643    7  250    8   14   13   75    1    6    0  145    0    4    2]
 [  25    2    0    0  125    4   38    3    1   11   12    0    0    0   24    0    1    0]
 [  54    1    3    0  322    2  159    0    2    3   32    1    0    0   61    0    1    0]
 [   8    0    1    4   82    0    6   22    2   18   19    0    0    0   20    0    4    0]
 [  16    1    0    0  129    3   10    0    0    3   13    0    0    0   27    0    1    0]
 [  85    1    8    3  234    7   42    0    2   76  110    1    2    4   56    0    2    0]
 [ 132    3    8    4  214    7   25    2    2   48  369    0    0    2   56    0    7    1]
 [   9    4    0    1   23    7   20    0    1    1   13    0    0    0    6    0    1    0]
 [  11    0    2    1   49    1   11    0    1    5   17    0    1    0   23    0    1    0]
 [   9    0    0    2   36    1    3    3    0    1   11    0    0    0    3    0    0    0]
 [ 424   10   19   12 1724   12  236   19   16   48  218    2    3    0 5702    3   16    1]
 [   0    2    0    0   74    0    8    0    0    0    3    0    0    0    4    0    0    0]
 [  33    1    2    2  134    1   28    1    1    5   39    0    1    1   13    0    2    0]
 [  11    5    3    1   17    3    7    0    4    0   13    0    0    0    2    1    1    0]]
```

[1]In the code I print confusion matrix every 200 iterations. The screencuts of confusion matrix in this report are all taken from the last print of matrices .

4.2 Compensated

If don't compensate for training data, the ConvNet will just learn to classify the dominating classes correctly. Thus I randomly sample the same number (i.e. the number of examples from the smallest class) to compensate for the unbalanced data. Validation loss:



Final Accuracy: 0.5908

Confusion Matrix:

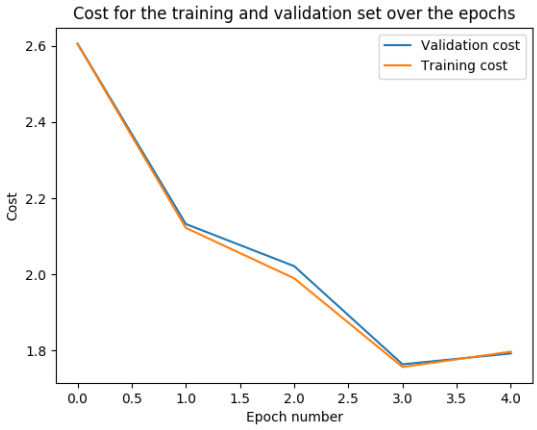
```
[[1442   29    0    0    0    0   19   66   20   13   89   32    0   31   27   11   18    9]
 [   2  170    1    0    1    0    1    0    0    0    2   32    0    0    0    0    1   26]
 [  10    4  174   18   23    9   37    8   15    9   31    4   30   14   44   16   26    5]
 [   3    3    6  177   10    6   21    2    6    1    1    3    2    2    4    3    3    7]
 [ 104   50  123  195  925  226  357   53  331   81   25   41   58   71  164  381   90   37]
 [   5    0    4    6   13  152   12   11   10    6    1    5    3    3    4    5    2    4]
 [  25   10   26   53   46   27  341    6   19    8    5    8    8    1   15   23    6   14]
 [   1    0    1    0    1    1    0  174    0    1    3    0    0    2    0    0    2    0]
 [   1    1    1    1   12    3    5    0  156    1    2    1    2    2    7    7    1    0]
 [  20    3   13    9    7   19    8    8    6  394   23    1    3   32    7    6   69    5]
 [  36   18   12    6    6    4    3   11    5   44  641   14   19   13   12    1   27    8]
 [   0   10    0    0    0    0    0    0    0    0    0   67    0    0    0    0    0    9]
 [   2    1    3    4    1    0    1    1    2    0    3    1  102    0    1    0    0    1]
 [   0    0    0    0    0    0    0    0    0    2    2    0    0   64    0    0    1    0]
 [ 137   26  318  142  263  119  233  106  165  175  201   29  135   47 6111  107  111   40]
 [   1    2    1    0    3    1    2    0    2    0    0    1    0    1    0   77    0    0]
 [   6    2    6    3    3    7    3    6    2   18   14    0    3   38    1    4  146    2]
 [   0    9    1    0    0    0    0    0    0    0    0    4    0    0    0    0    0   54]]
```

5 Differences between two training regimes

Hyper-parameter setting: (n1, k1, n2, k2)=(20, 5, 20, 3), eta=0.01(decaying rate=0.1), rho=0.9, batch size=100, ite=1000

5.1 Non-compensated

Train Validation cost:



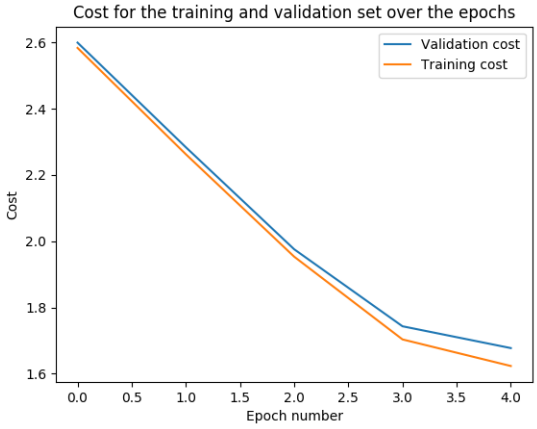
Final Accuracy: 0.4755

Confusion Matrix:

```
[[ 400   14    0    0 1136    0   37    0    0   30  154    0    0    0   35    0    0    0]
 [  13    1    0    0  186    0    5    0    0    0   16    0    0    0   12    0    3    0]
 [  31    1   11    0  258    3   48    0    0    9   31    0    0    0   85    0    0    0]
 [   2    0    4    0  169    6   42    0    0    4    5    0    0    0   28    0    0    0]
 [  84    0   16    0 2625    9  338    0    0   44   45    0    0    0  149    0    2    0]
 [   9    2    0    0  156    5   41    0    1    4    7    0    0    0   21    0    0    0]
 [   8    2    8    0  372    7  186    0    0    8    7    0    0    0   42    0    1    0]
 [  12    0    8    0   89    0   23    1    0    2   11    0    1    0   37    0    2    0]
 [   6    0    0    0  145    2   11    1    0    4    9    0    0    0   25    0    0    0]
 [  27    2    8    0  387    3   89    2    0   43   56    0    0    0   10    0    6    0]
 [  67    1    6    0  473    1   56    1    0   29  219    0    0    0   27    0    0    0]
 [   3    0    0    0   66    0    2    0    0    1   12    0    0    0    2    0    0    0]
 [  13    0   13    0   56    0   10    2    0    1    5    0    0    0   23    0    0    0]
 [   4    0    0    0   46    0   11    0    1    2    3    0    0    0    2    0    0    0]
 [ 246    2   75    0 2213   10  275    1    1   44  108    0    0    0 5488    0    2    0]
 [   2    0    0    0   76    0    8    0    0    1    2    0    0    0    2    0    0    0]
 [  15    1    0    0  172    0   42    0    0    5   23    0    0    0    6    0    0    0]
 [   4    0    0    0   41    0    4    0    0    0   12    0    0    0    4    0    3    0]]
```

5.2 Compensated

Train Validation cost:



Final Accuracy:0.4810

Confusion Matrix:

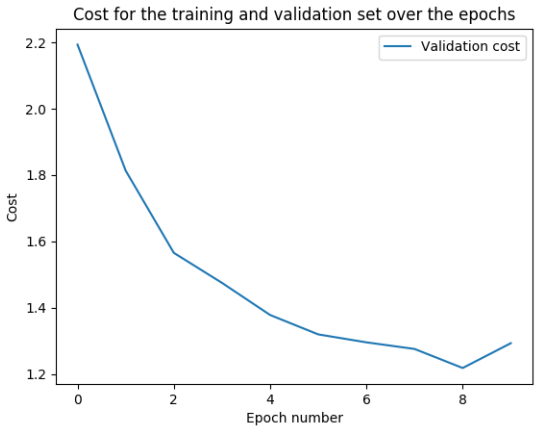
```
[[1107   29   36    0    0    0   53   62   65    0  223   29   11   47   15   37   15   77]
 [   1  147    1    1    0    0    0    0    2    0    2   50    0    1    0    0    0   31]
 [  34    1  120   19   30   13   35    6   22   18   20    6   52   18   42   18   22    1]
 [   3   11   10  109    6    6   31    3    8    1    6    8    6    5   20   16    9    2]
 [ 127   66  157  222  734  225  391   94  274   74   34   41   53  140  100  406  120   54]
 [  12    6    0    9   16   80   28    9   16   16    5    1    3   17    2   11   10    5]
 [  26   14   42   58   55   37  241    7   30   11    6    8   12    9   19   26   16   24]
 [   3    0    0    1    2    1    0  153    0    7    4    0    0    8    1    1    5    0]
 [   8    6    3    4   10   13   13    2  108    1    8    3    2    5    3    9    2    3]
 [  30    8    8   13    8   20   10   12   12  304   57    4    6   58    4   11   66    2]
 [  70   14    6    8    1    4    2   10   10   30  640   15   10   23    8    2   18    9]
 [   1   10    0    0    0    0    0    0    0    0    1   65    0    0    0    0    0    9]
 [   4    2    8    2    0    1    7    0    4    0    5    1   85    1    1    0    2    0]
 [   2    0    0    1    1    0    1    3    2    3    2    0    0   50    0    0    3    1]
 [ 227   34  545  324  294  140  194  158  398  151  317   38  302   41 4898  229  125   50]
 [   3    4    3    1    6    3    7    2    8    5    0    1    0    4    0   41    3    0]
 [  18    7    7    8    7    9    8   10    4   25   15    0    6   42    3    6   88    1]
 [   0   13    0    1    0    0    1    0    0    0    0   15    0    0    0    0    0   38]]
```

The confusion matrix on the compensated dataset shows both the dominating classes and the others are well classified. More mistakes came from the dominating classes due to the large number of data inside. For the confusion matrix on the non-compensated one, the dominating classes are well classified but the others are not well predicted. Besides, error of the uncompensated one isn't much higher even if some classes have almost 0 accuracy because the dominating ones are far more dominant than the others. What's more, compensation of data could fix overfitting problem in comparison of two validation cost plots.

6 Best performing ConvNet

I took compensated data to get the best performance. With observation of cost plots, it's still converging, so I increased the number of iterations. In the past assignments, we've done the research of learning rate, momentum term, batch size, etc. Thus, hyper-parameter setting: (n1, k1, n2, k2)=(50, 5, 50, 3), eta=0.01(decay rate=0.1), rho=0.9, batch size=100, ite=2000

Validation cost:



Final Accuracy:0.6582

Confusion Matrix:

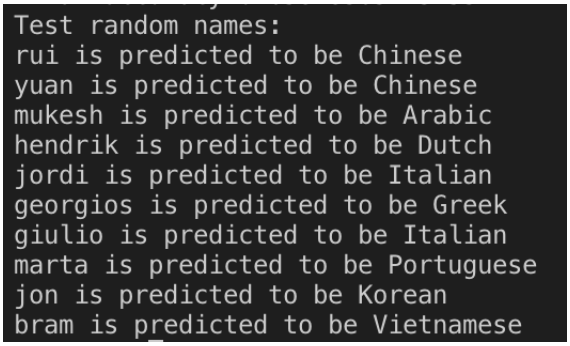
```
[[1762   14    0   11    0    0    0    0    0    0    0   10    0    0    0    0    0    9]
 [   2  181    0    0    0    0    0    0    0    0    0    2   34    0    1    0    1    0   15]
 [   4    4  312   10   26    5   46    3    6   10    8    1   12    8    9    5    8    0]
 [   0    5    0  236    1    3    6    1    0    0    0    3    1    0    0    2    2    0]
 [  70   49  195  226 1305  232  357   43  226   51   19   29   35   22   63  300   69   21]
 [   0    2    0    4    9  204    8    1    2    3    0    4    6    0    0    1    2    0]
 [  11    8   14   45   30   25  464    1    4    3    1    4    3    0    9   11    3    5]
 [   0    0    1    0    0    1    0  181    0    0    1    0    0    0    0    0    2    0]
 [   0    1    0    0   10    1    4    0  180    0    0    0    1    0    0    6    0    0]
 [   8    2   12    4    4   23    2    1    3  492   14    1    7   13    4    2   40    1]
 [  31   22   20   11    2    5    4    2    3   44  691    9   10    6    4    1   12    3]
 [   0    5    0    0    0    0    0    0    0    0    0   77    0    0    0    0    0    4]
 [   0    1    2    1    0    1    1    0    0    0    0    0  117    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0    0    0    0    0    0   66    0    0    3    0]
 [ 132   39  546  133  248  127  303   56  174  153  100   23   87   14 6166   42   88   34]
 [   0    2    0    0    0    0    1    0    0    0    0    0    0    0    0   87    0    1]
 [   2    3    3    1    2    9    3    1    2    9    5    1    3   26    0    2  191    1]
 [   0    2    0    0    0    0    0    0    0    0    0    2    0    0    0    0    0   64]]
```

7 Test on random friends' name

Here I chose my KTH friends' name to test the function of my code.

Testfriends = [”rui”, ”yuan”, ”mukesh”, ”hendrik”, ”jordi”, ”georgios”, ”giulio”, ”marta”, ”jon”, ”bram”]

where ”rui” is Chinese, ”yuan” is Chinese, ”mukesh” is Indian, ”hendrik” is German, ”jordi” is German, ”georgios” is Greek, ”giulio” is Itlian, ”marta” is Portuguese, ”jon” is Spanish, ”bram” is Belgium. The screencut of predictions as following:



The accuracy of test is 5/10 which is close to the final accuracy 0.6582 given above. We could see that ConvNet is well performed in Chinese. What’s worth mentioning, I took names that don’t belong to any class in the training dataset as noisy data, like ”mukesh” is Indian, ”bram” is Belgium. It’s interesting and reasonable to find that mukesh is predicted to be Arabic. Besides, Dutch and German are quite similar to some extent, that’s why ”hendrik” is German but is predicted to be Dutch.

References

[1] Eli Bendersky. The softmax function and its derivative, October 2016.