# DL Assignment 3

Meinan Gou meinang@kth.se

## 1 Introduction

In this assignment, I trained a 2-layer ConvNet to predict the language of a surname from its spelling using Mini-batch gradient descent.

## 2 Functions

I used two classes for data processing and convolution respectively.

Class **Data**: Read files and save all parameters in a dictionary needed in CNN.

Main functions in Class **ConvNet**:

- *__init__()* and *initialization()*: Initialize all parameters in training.
- *MFMatrix()* and *MXMatrix()*: Generate MF and MX.
- *evaluateClassifier()*, *computeCost()*, *cross_entropy()* and *computeAccuracy()*: Evaluate the current network performance.
- *testing()*: Test on friends' surnames.
- *confusion_matrix()*: Create confusion matrix.
- *train()*: Train the network with forward and backward pass.

## 3 Results

### 3.1 Compare analytic and numerical gradient computations

The function relative_error() was defined to calculate the relative error between analytical and numerical results. I used the first 100 samples in the training data. The formula is

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

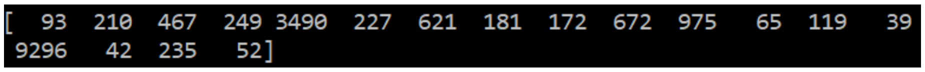
I set $eps$=1e-5. The maximum relative error of $W, grad\_F1$ and $grad\_F2$ are at level of 1e-8 or less, which shows that the analytic gradient computations have no bug.

## 3.2 Compensate the imbalanced dataset

I used the second option to compensate the data. The function was implemented in method *train()* in class *ConvNet*. I assign the number of smallest class to variable *min_class*. Then I randomly sampled *min_class* of samples from each class and built a numpy array *idx* which contains the shuffled index of the chosen samples. Then I take *batch_size* samples from *idx* to form the mini-batch of the current epoch. After all samples in *idx* has been trained, *idx* is updated with other randomly chosen indices.

## 3.3 Validation loss with and without compensating

The amount of samples in each class in training data is as follows:



```
[  93  210  467  249 3490  227  621  181  172  672  975   65  119   39
 9296   42  235   52]
```

Figure 1: Amount of samples in each class

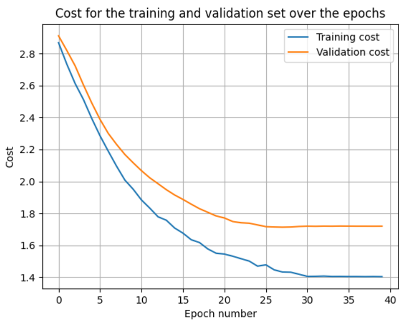
Each number corresponds to class

['Arabic', 'Chinese', 'Czech', 'Dutch', 'English', 'French', 'German', 'Greek', 'Irish', 'Italian', 'Japanese', 'Korean', 'Polish', 'Portuguese', 'Russian', 'Scottish', 'Spanish', 'Vietnamese'].

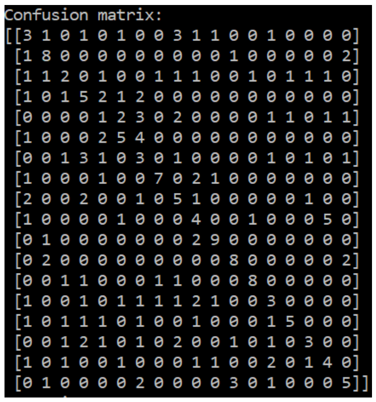
As is shown above, there are very large number compared to others, such as 3490 (English) and 9296 (Russian). This would affect heavily on the network performance if no compensating is applied.

### 3.3.1 With compensating

I set maximum iteration=20000, and $n_1, k_1, n_2, k_2 = 20, 5, 20, 3$ respectively. I checked validation loss and the confusion matrix every 500 iterations.



(a) loss



```
Confusion matrix:
[[3 1 0 1 0 1 0 0 3 1 1 0 0 1 0 0 0 0]
 [1 8 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 2]
 [1 1 2 0 1 0 0 1 1 1 0 0 1 0 1 1 1 0]
 [1 0 1 5 2 1 2 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 1 2 3 0 2 0 0 0 0 1 1 0 1 1]
 [1 0 0 0 2 5 4 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 1 3 1 0 3 0 1 0 0 0 0 1 0 1 0 1]
 [1 0 0 0 1 0 0 7 0 2 1 0 0 0 0 0 0 0]
 [2 0 0 2 0 0 1 0 5 1 0 0 0 0 0 1 0 0]
 [1 0 0 0 0 1 0 0 0 4 0 0 1 0 0 0 5 0]
 [0 1 0 0 0 0 0 0 0 2 9 0 0 0 0 0 0 0]
 [0 2 0 0 0 0 0 0 0 0 0 8 0 0 0 0 0 2]
 [0 0 1 1 0 0 0 1 1 0 0 0 8 0 0 0 0 0]
 [1 0 0 1 0 1 1 1 1 2 1 0 0 3 0 0 0 0]
 [1 0 1 1 1 0 1 0 0 1 0 0 0 1 5 0 0 0]
 [0 0 1 2 1 0 1 0 2 0 0 1 0 1 0 3 0 0]
 [1 0 1 0 0 1 0 0 0 1 1 0 0 2 0 1 4 0]
 [0 1 0 0 0 0 2 0 0 0 0 3 0 1 0 0 0 5]]
```

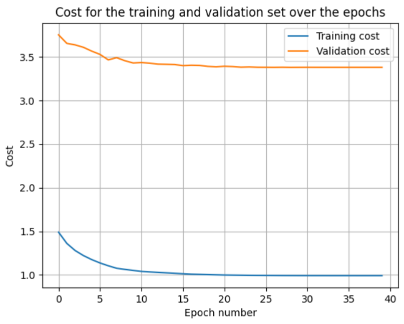
(b) Confusion matrix Validation

Figure 2: Balanced data

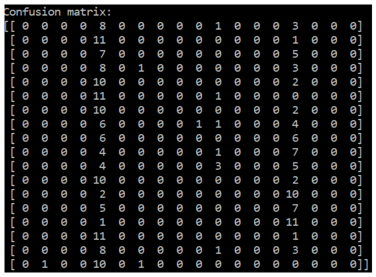
In Figure 2 (a), one epoch corresponds to 500 iterations, and one iteration corresponds to training with one mini-batch. The final validation loss is 1.72. The final validation accuracy is 39.35%.

### 3.3.2 Without compensating

The hyper-parameters settings are the same as in compensating.



(a) loss



```
Confusion matrix:
[[ 0  0  0  0  8  0  0  0  0  0  1  0  0  0  3  0  0  0]
 [ 0  0  0  0 11  0  0  0  0  0  0  0  0  0  1  0  0  0]
 [ 0  0  0  0  7  0  0  0  0  0  0  0  0  0  5  0  0  0]
 [ 0  0  0  0  8  0  1  0  0  0  0  0  0  0  3  0  0  0]
 [ 0  0  0  0 10  0  0  0  0  0  0  0  0  0  2  0  0  0]
 [ 0  0  0  0 11  0  0  0  0  0  1  0  0  0  0  0  0  0]
 [ 0  0  0  0 10  0  0  0  0  0  0  0  0  0  2  0  0  0]
 [ 0  0  0  0  6  0  0  0  0  1  1  0  0  0  4  0  0  0]
 [ 0  0  0  0  6  0  0  0  0  0  0  0  0  0  6  0  0  0]
 [ 0  0  0  0  4  0  0  0  0  0  1  0  0  0  7  0  0  0]
 [ 0  0  0  0  4  0  0  0  0  0  3  0  0  0  5  0  0  0]
 [ 0  0  0  0 10  0  0  0  0  0  0  0  0  0  2  0  0  0]
 [ 0  0  0  0  2  0  0  0  0  0  0  0  0  0 10  0  0  0]
 [ 0  0  0  0  5  0  0  0  0  0  0  0  0  0  7  0  0  0]
 [ 0  0  0  0  1  0  0  0  0  0  0  0  0  0 11  0  0  0]
 [ 0  0  0  0 11  0  0  0  0  0  0  0  0  0  1  0  0  0]
 [ 0  0  0  0  8  0  0  0  0  0  1  0  0  0  3  0  0  0]
 [ 0  1  0  0 10  0  1  0  0  0  0  0  0  0  0  0  0  0]]
```

(b) Confusion matrix validation

Figure 3: Unbalanced data

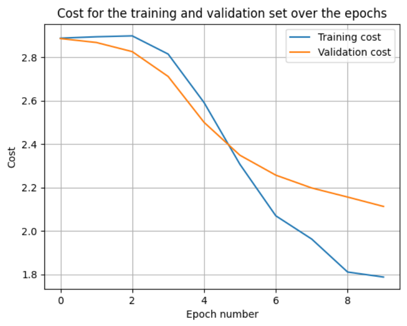
The final validation loss is 3.38. The final validation accuracy is 14.81%.

Compared with Figure 2, the loss reduction in Figure 3 is much slower. The confusion matrices also show that the the samples cluster to the classes which contain the large number of samples. The final accuracy is much lower than compensating data. This is because the samples in these classes have much larger chance to be chosen during training, whereas other classes cannot get enough training. That is why the result shows a relatively good performance on these classes, while other classes have poor result.

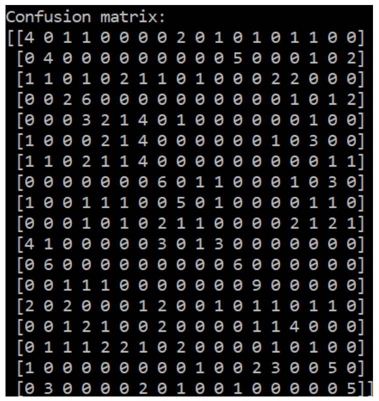
## 3.4 Validation loss for best performing ConvNet

I experimented with different number of filters per layer and different width of filters with a grid search. I found that the more filters applied per layer, the better performance the network can achieve. During my search, I found that the following hyper-parameter settings behave well:

(n1, k1, n2, k2)=(40, 5, 40, 3), eta=0.01 (decrease to half once validation loss increases), rho=0.9, batch_size=100, n_iter=3000.



(a) loss



```
Confusion matrix:
[[4 0 1 1 0 0 0 0 2 0 1 0 1 0 1 1 0 0]
 [0 4 0 0 0 0 0 0 0 0 0 5 0 0 0 1 0 2]
 [1 1 0 1 0 2 1 1 0 1 0 0 0 2 2 0 0 0]
 [0 0 2 6 0 0 0 0 0 0 0 0 0 0 1 0 1 2]
 [0 0 0 3 2 1 4 0 1 0 0 0 0 0 0 1 0 0]
 [1 0 0 0 2 1 4 0 0 0 0 0 0 1 0 3 0 0]
 [1 1 0 2 1 1 4 0 0 0 0 0 0 0 0 0 1 1]
 [0 0 0 0 0 0 0 6 0 1 1 0 0 0 1 0 3 0]
 [1 0 0 1 1 1 0 0 5 0 1 0 0 0 0 1 1 0]
 [0 0 0 1 0 1 0 2 1 1 0 0 0 0 2 1 2 1]
 [4 1 0 0 0 0 0 3 0 1 3 0 0 0 0 0 0 0]
 [0 6 0 0 0 0 0 0 0 0 0 6 0 0 0 0 0 0]
 [0 0 1 1 1 0 0 0 0 0 0 0 9 0 0 0 0 0]
 [2 0 2 0 0 0 1 2 0 0 1 0 1 1 0 1 1 0]
 [0 0 1 2 1 0 0 2 0 0 0 0 1 1 4 0 0 0]
 [0 1 1 1 2 2 1 0 2 0 0 0 0 1 0 1 0 0]
 [1 0 0 0 0 0 0 0 0 1 0 0 2 3 0 0 5 0]
 [0 3 0 0 0 0 2 0 1 0 0 1 0 0 0 0 0 5]]
```

(b) Confusion matrix validation

Figure 4: Good hyper-parameter setting curves

As shown in Figure 4, after 3000 iterations, the loss is 2.11 and the validation accuracy achieved 31.02%. From Figure 4 (a), I see that with larger number of iterations, the validation loss can decrease more until converge. However, due to time limit, I did not manage to train for a longer time, but I can say that the hyper-parameter settings are appropriate, and decrease learning rate during training is a good option.

## 3.5 Improvement on efficiency

According to background 5, before entering the training loop, I computed the first layer of $M^{input}_{x_j,k_1,n_1}$ and made each one sparse using *scipy.sparse.csr_matrix()*. Then I placed all of them in a list, which was converted to numpy array after appending done.

For another improvement, I created generalization $M^{input}_{x_j,k_2}$.

I tested the network with (n1, k1, n2, k2)=(20, 5, 20, 3), eta=0.001, rho=0.9, batch_size=100, n_iter=1000, n_update=200 in each setting, respectively.

Without these improvements, training used 1014.21 seconds.

With these improvements, training used 631.96 seconds.

## 3.6 Test on friends' names

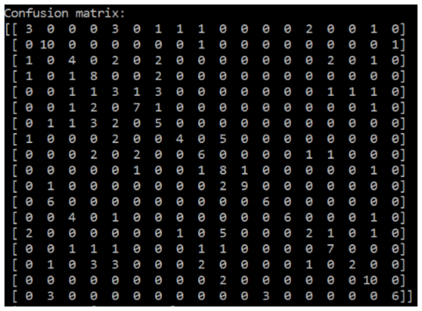
I set a list containing my and my friends' surnames:

["gou", "sun", "bin", "yusefi", "bianchi", "maki", "dimitriou"]

Their ground truth labels are:

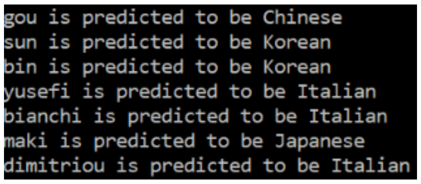
["Chinese", "Chinese", "Chinese", "Arabic", "Italian", "Japanese", "Greek"]

With my best settings with iteration 20000, my validation accuracy achieved 44.29%.



```
Confusion matrix:
[[ 3  0  0  0  3  0  1  1  1  0  0  0  0  2  0  0  1  0]
 [ 0 10  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0  1]
 [ 1  0  4  0  2  0  2  0  0  0  0  0  0  0  2  0  1  0]
 [ 1  0  1  8  0  0  2  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  1  1  3  1  3  0  0  0  0  0  0  0  1  1  1  0]
 [ 0  0  1  2  0  7  1  0  0  0  0  0  0  0  0  0  1  0]
 [ 0  1  1  3  2  0  5  0  0  0  0  0  0  0  0  0  0  0]
 [ 1  0  0  0  2  0  0  4  0  5  0  0  0  0  0  0  0  0]
 [ 0  0  0  2  0  2  0  0  6  0  0  0  0  1  1  0  0  0]
 [ 0  0  0  0  0  1  0  0  1  8  1  0  0  0  0  0  1  0]
 [ 0  1  0  0  0  0  0  0  0  2  9  0  0  0  0  0  0  0]
 [ 0  6  0  0  0  0  0  0  0  0  0  6  0  0  0  0  0  0]
 [ 0  0  4  0  1  0  0  0  0  0  0  0  6  0  0  0  1  0]
 [ 2  0  0  0  0  0  0  1  0  5  0  0  0  2  1  0  1  0]
 [ 0  0  1  1  1  0  0  0  1  1  0  0  0  0  7  0  0  0]
 [ 0  1  0  3  3  0  0  0  2  0  0  0  0  1  0  2  0  0]
 [ 0  0  0  0  0  0  0  0  0  2  0  0  0  0  0  0 10  0]
 [ 0  3  0  0  0  0  0  0  0  0  0  3  0  0  0  0  0  6]]
```

(a) Confusion matrix validation



```
gou is predicted to be Chinese
sun is predicted to be Korean
bin is predicted to be Korean
yusefi is predicted to be Italian
bianchi is predicted to be Italian
maki is predicted to be Japanese
dimitriou is predicted to be Italian
```

(b) Prediction result

Figure 5: Results on testing on friends' names

As is shown in Figure 5, 3 of 7 surnames were predicted correctly. The testing accuracy is 42.86%.

The test accuracy is similar to validation accuracy. There are two Chinese surnames predicted to be Korean, which may be because there are some similarities between Chinese and Korean surnames. The result can be improved by increasing training iterations or increase CNN layers or the number of filters.

## 4 Conclusions

This is the very first time to use a CNN to process text. Although it is a little complicated when implementing MF and MX, the tasks are interesting. I also tried to get a better performance in optional part by implementing more layers and max-pooling.