KTH Royal Institute of Technology
DD2424-VT19-1 Deep Learning in Data Science
Assignment 3

Marina Herrera Sarrias

August 8, 2019

3.1 Gradients Computation and Sanity Check

The maximum relative error was computed using Equation.(1) as a mean of comparing the numerical ($x_{numerical}$) and analytical ($x_{analytical}$) gradients, setting ε to 1e-30.

$$\frac{|x_{analytical} - x_{numerical}|}{\max(\varepsilon, |x_{analytical}| + |x_{numerical}|)} \tag{1}$$

The numerical estimations were computed using the finite difference formulation while the analytical gradients were computed using the back propagation algorithm and the ReLu activation function. Each layer has 5 filters of width 5. The results for the mini-batches of size one, two and three can be found in Table 1.

	Maximum Relative Error Finite Difference		
	Batch size =1	Batch size =2	Batch size =3
W	7.22e-06	2.24e-05	5.51e-06
F_1	1.31e-09	5.64e-09	1.00e-08
F_2	5.17e-07	1.63e-07	3.67e-07

Table 1: Maximum Relative error between numerical and analytical gradient vectors computations.

We can verify that the finite difference formula provides a "good" approximation to the analytical computations of the gradients.

3.2 Handling the imbalanced data set during training

Although the validation data set has the same number of examples per class, the training data set was very unbalanced, having a predominant class "Russian" with a $\sim 47\%$ relative frequency, of the total training data.

For balancing the data, I randomly under sampled the data set, selecting the same number of examples per class. I repeated this process at the beginning of each epoch and used this as the training set. The balanced training set had 1062 examples, containing 58 examples per class, as 58 corresponds to the class with the fewer number of examples. The unbalanced

data set has 19798 examples, its relative frequency and cumulative relative frequency can be found in Table 2.

Class	Relative Frequency	Cumulative Relative Frequency
Russian	47.33 %	47.33 %
English	18.46 %	65.79 %
Arabic	10.03 %	75.82 %
Japanese	4.94 %	80.76 %
German	3.59 %	84.35 %
Italian	3.51 %	87.86 %
Czech	2.55 %	90.41 %
Dutch	1.43 %	91.84 %
Spanish	1.43 %	93.27 %
French	1.33 %	94.60 %
Chinese	1.28 %	95.88 %
Irish	1.10 %	96.98 %
Greek	0.96 %	97.94 %
Polish	0.63 %	98.57 %
Scottish	0.43 %	99.00 %
Korean	0.40 %	99.40 %
Portuguese	0.30 %	99.70 %
Vietnamese	0.30 %	100.00 %

Table 2: Relative frequency and cumulative frequency table for the training data set.

3.3 Validation loss function for a longer training run

As a mean of comparing the effect of balancing the training data set, I trained two models using the following training parameters: `n_1`=20, `k_1`=5, `n_2`=20 `k_2`=3, `eta`=0.001, `rho`=0.9.

The unbalanced model was trained for `epochs`=135 and with a `batch_size`=100 which results in `update_steps`=26595 while the balanced model was trained for `epochs`=2516 and with a `batch_size`=59 which results in `update_steps`=45288, this as the balanced training data set has a much smaller size than the unbalanced training data set. I computed the number of epochs for the balanced data set as : $epochs$ $(\frac{Npts}{Balanced\_Npts})$.

Results after training with unbalanced Data:

After training the first model without balancing the training data the best accuracy obtained after 135 epochs in the validation set was of 26.59%, while on the training set was of 74.80%. Graphs for the loss function and accuracy during training can be verified in Figure 1, while the confusion matrix corresponding to the last epoch of training can be verified in Figure 2.

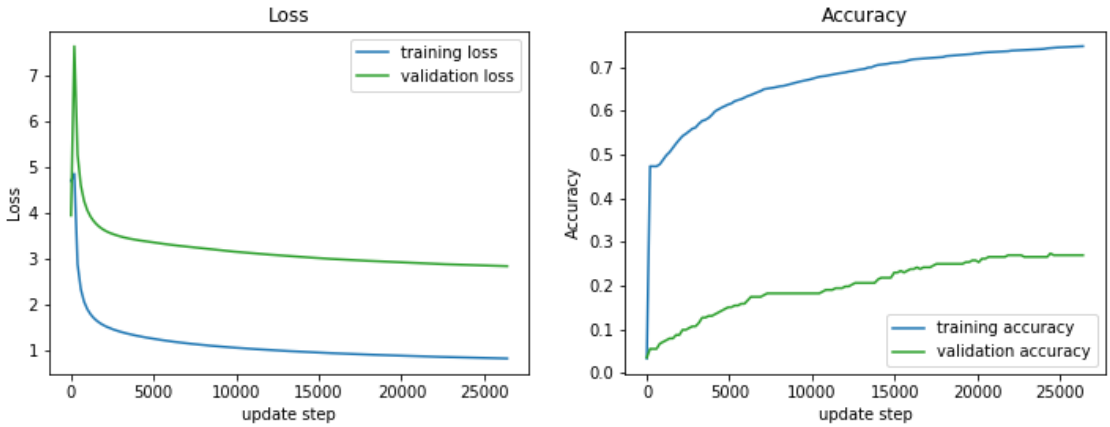

Figure 1: Loss and accuracy for 135 epochs of training using unbalanced data.

The confusion matrix for the validation data in Figure 2, shows that the model had a preference to incorrectly classify examples in the three most dominant classes 14, 4 and 1, which correspond to "Russian", "English" and "Arabic". It also properly classified almost all examples corresponding to the same dominant classes.

Referring now to the training data in Figure 2 the model succeeded in classifying most of the examples of the dominant classes, but it also showed a preference for wrongly classifying examples in these classes more than others.

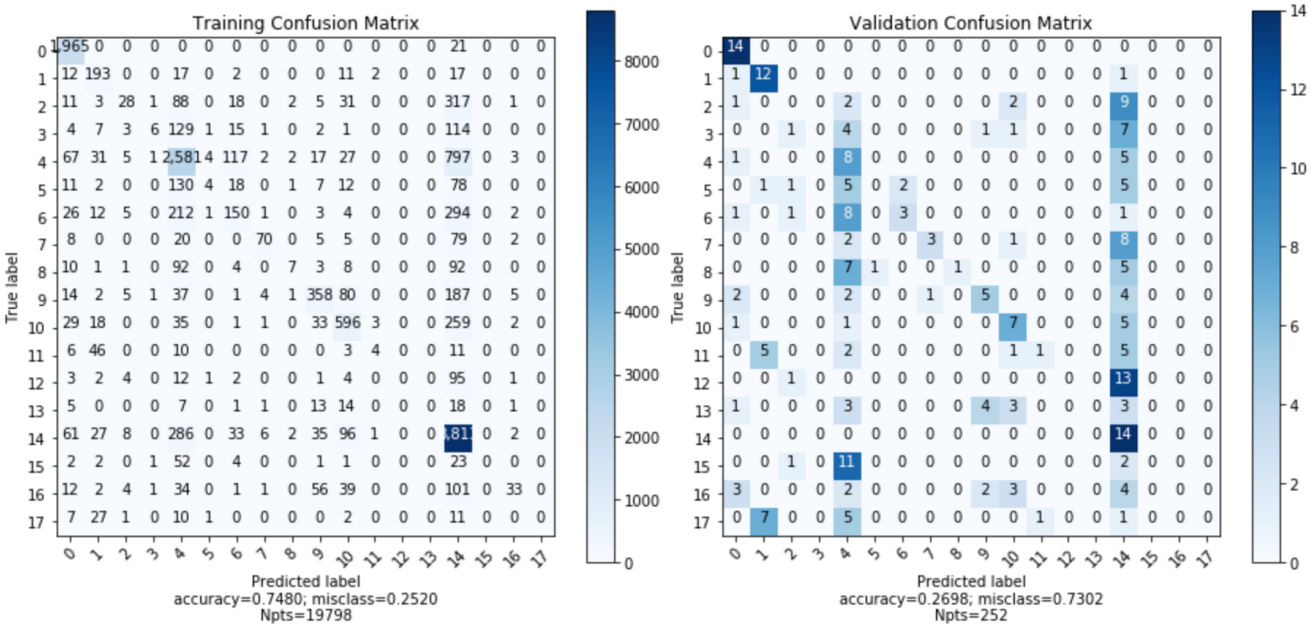

Figure 2: Confusion Matrix without normalization corresponding to the last epoch of training using unbalanced data.

Results after training with Unbalanced Data:

Figure 3 shows the loss function and accuracy for the model trained with balanced data. After training the model for 2516 epochs it achieved a final validation accuracy of 50.00% which is much higher than in the unbalanced case. Most of the learning and accuracy gain was achieved on the first epochs. For the rest of the training period the accuracy and loss remained constant or slightly varied, showing possibly over-fitting due to its behavior on the last updates, as the validation loss starts increasing while the accuracy remains constant or slightly increases. I also noticed that the Network was very sensitive to the weights initialization.

Referring to the confusion matrix in Figure 4 it is clear that under sampling the training data had a notable result in improving the accuracy for both, the training and validation data (although the training confusion matrix is not of the full data set, if not of the balanced data set). Furthermore, it seems not to be a preference for the incorrectly classified examples.

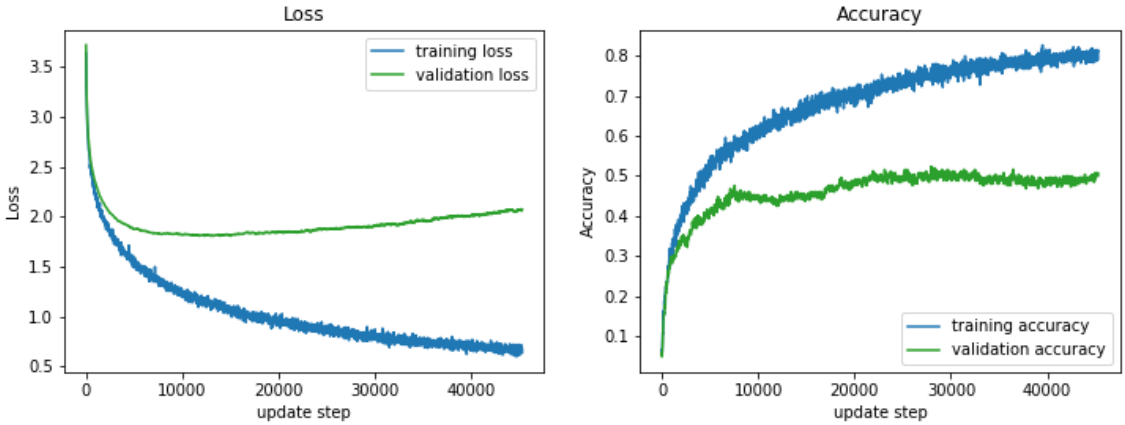

Figure 3: Loss and accuracy for 2516 epochs of training using balanced data.

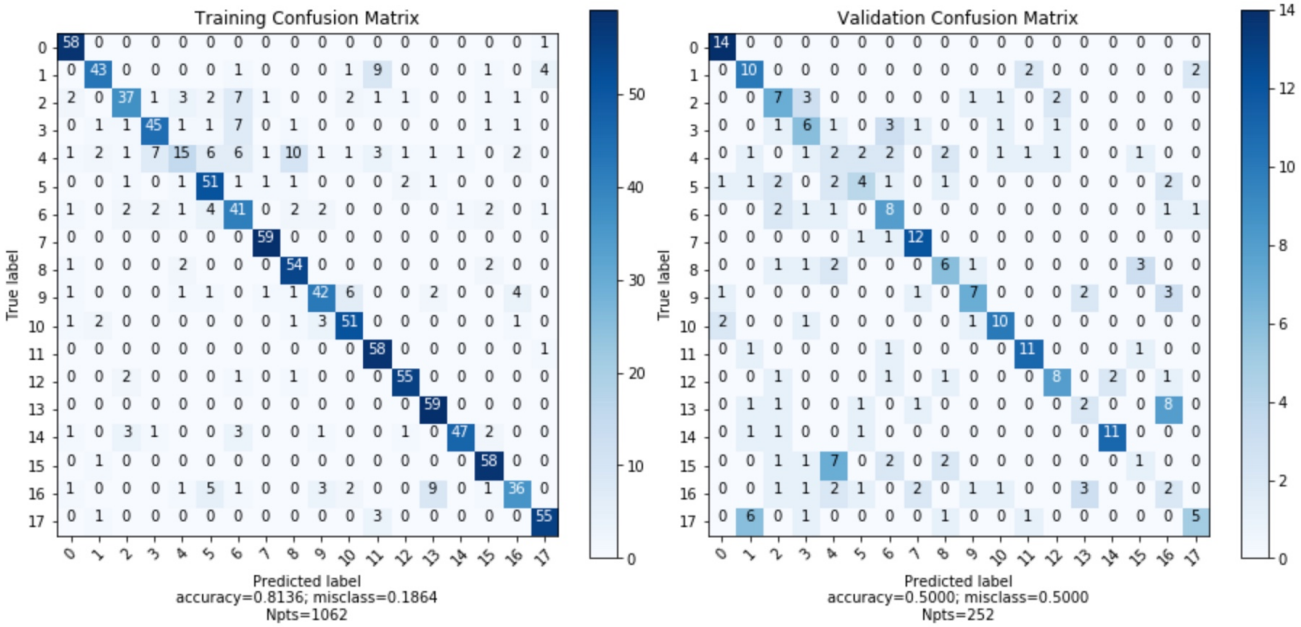

Figure 4: Confusion Matrix without normalization corresponding to the last epoch of training using balanced data.

3.4 Best performing ConvNet

The cross-entropy loss function in Equation 2 was the validation loss function used for training the models.

$$l_{cross}(\mathbf{x}, \mathbf{y}, \boldsymbol{F_1}, \boldsymbol{F_2}, W) = -\log(\mathbf{y}^T \mathbf{p}) \tag{2}$$

The best model achieved on the validation set an accuracy of 52.38% and a minimum loss of 1.88. The training data was balanced for training the model, which was trained for 45288 update steps, with the following settings: `n_1`=20, `k_1`=5, `n_2`=20 `k_2`=3, `eta`=0.001, `rho`=0.9, `batch_size`=57.

Table 3 shows the accuracy obtained per class by the model.

Class	Class Accuracy
Arabic	100.00 %
Chinese	85.71 %
Greek	85.71 %
Korean	85.71 %
Japanese	78.57 %
Russian	78.57 %
Polish	71.43 %
German	50.00 %
Irish	50.00 %
Vietnamese	50.00 %
Czech	42.86 %
Italian	42.86 %
Dutch	35.71%
French	35.71 %
English	14.29 %
Portuguese	14.29 %
Scottish	14.29 %
Spanish	7.14 %

Table 3: Best performing ConvNet validation accuracy per class.

3.5 Efficiency gains

With the aim of speeding up the training process I computed the sparse input matrix $\mathbf{M}^{input}_{\mathbf{x}_j,k_1,n_1}$ for the first layer at the beginning of the training session, this as $\mathbf{M}^{input}_{\mathbf{x}_j,k_1,n_1}$ only depends on the input data and remains constant through all the training.

For testing the efficiency gain I trained two models for 5 epochs ($\sim$ 985 updates) with the following parameter settings: `n_1`=5, `k_1`=5, `n_2`=5 `k_2`=5, `eta`=0.001, `rho`=0.9, and `batch_size`= 100. Both models were trained with the unbalanced training data.

The model trained without the changes on the input matrix of the first layer took $\sim$ 37.63 seconds, while the model trained with the update took $\sim$ 28.50 seconds. Therefore we can verify that this extra measure brings more efficiency during training.

3.6 Best network probability vector output

For testing the best performing network I applied the surnames listed bellow, and obtained the surname's origin that follows each listed surname. All predictions were correct. Table 4 shows probability output vector for each surname.

```
Herrera: Spanish
Nguyen: Vietnamese
Souza: Portuguese
Vilanova: Spanish
De Fenza: Italian
Claramunt: French
```

	Herrera	Nguyen	Souza	Vilanova	De Fenza	Claramunt
Arabic	3.09328892e-08	1.49923689e-07	1.72106003e-02	6.48541112e-15	5.44748054e-08	4.28996584e-07
Chinese	9.11485344e-14	3.42957384e-10	1.71346028e-05	1.84662065e-12	7.01570222e-09	1.78886837e-10
Czech	5.37040779e-02	1.14539620e-03	3.13104525e-03	1.83838147e-01	1.43566821e-02	9.58435635e-04
Dutch	6.77519434e-02	3.00235661e-05	7.65958019e-05	6.10754718e-07	3.96583086e-03	3.98506942e-05
English	1.03629306e-01	9.43627004e-03	1.11854222e-03	3.83045922e-04	1.72902601e-02	1.83080270e-03
French	7.30066229e-03	4.86062334e-03	6.37419059e-04	1.00021996e-07	8.19688431e-04	9.82831454e-01
German	1.53925285e-02	2.29891856e-03	1.42663158e-04	4.19561581e-07	8.59956624e-03	1.31885068e-03
Greek	3.28955464e-04	1.80669322e-07	2.89894372e-05	9.92269401e-04	4.52898282e-05	7.63117769e-08
Irish	7.08894164e-03	4.95105083e-03	1.80533689e-04	3.72856072e-05	8.16784385e-02	1.87749805e-04
Italian	5.37339092e-02	3.78213685e-04	2.23122039e-02	1.27012738e-01	8.46583830e-01	6.24862454e-03
Japanese	1.11672913e-05	1.85089355e-06	6.16032549e-02	1.81119010e-03	3.38698215e-03	3.72810989e-05
Korean	3.11175442e-10	1.05233560e-09	4.18914883e-10	3.30499315e-18	8.16732807e-12	2.71952940e-11
Polish	2.19297243e-04	1.99955994e-06	1.95074358e-03	1.02820493e-05	4.20369747e-04	5.06456549e-09
Portuguese	4.61645679e-04	1.23163603e-09	8.44576711e-01	8.78807627e-04	5.12395184e-05	1.17203246e-05
Russian	1.21049315e-03	7.65119765e-04	2.27387216e-04	1.40397818e-02	1.82163000e-03	5.95823779e-03
Scottish	2.26286528e-02	2.28867451e-04	7.27083449e-06	3.90082841e-09	1.04672515e-07	6.15757082e-05
Spanish	6.66538388e-01	1.22001369e-05	4.66800684e-02	6.70995318e-01	2.09787093e-02	5.14906223e-04
Vietnamese	3.13197381e-10	9.75889133e-01	9.88358073e-05	1.05612071e-13	1.31690260e-06	2.29775943e-12

Table 4: Probability output vector for CNN