Assignment 3: Convolutional network

DD2424 - Deep Learning
Svenja Räther

March 2020

1 Introduction

This assignment 3 in the course DD2424 Deep Learning in Data Science trains a ConvNet to predict the language of a surname from its spelling.

2 Comparison of numerical and analytical gradients

The analytical gradients are computed using function *passBackward*. Those follow the implementation given in the assignment instructions. To validate the analytical gradients, those were compared against the numerical computation of the gradients. The implementation for those was given as a Matlab version by the teacher. One of the students in this course provided a translation of those in python which I used to evaluate my implementation of the analytical gradients.

Due to speed issues, the test was called using a reduced number of names (in this case 10). The relative errors in in table 1 was given for the comparison. Those can be considered sufficiently small. Therefore, I continued with the assignment.

Gradient	**Maximum relative error**
W	1.182452676965648e-06
F1	2.833149193352273e-08
F2	4.0380142973248844e-08

Table 1: Relative error for comparing numerical and analytical gradients for 10 entries

3 Compensating for imbalanced dataset during training

To compensate for the imbalanced dataset, a balanced subset of the training set is chosen at each epoch. This subset includes the same amount of examples for each class (based on the number of examples in the least represented class, in this case class 18 with 59 examples). This method is usually described as undersampling since it results in an overall smaller dataset. The random sampling and construction of a dictionary holding all indexes for each is done using the functions *create_index_dict* and *createBalancedDataset*.

The following subsections show the results of training the convolutional network with an imbalanced and a balanced dataset. Both used the following setup: *n1 = 20, n2 = 20, k1 = 5, k2 = 3, n_batch = 100, eta = 0.01, rho = 0.9*. Due to the different sizes of the datasets, the imbalanced example was trained for *epochs = 100* while the balanced one used *epochs = 2000*. Both setups should apply *around 20000 update steps*. The results were plotted every 500 update steps.

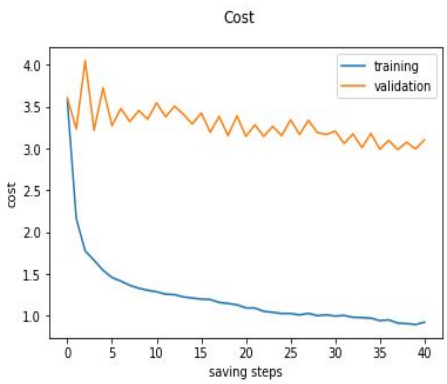
3.1 Imbalanced dataset

The given dataset for training is heavily imbalanced while the sample taken for validation represents a balanced set. The training dataset containes most examples for the classes 15 (9370 examples), 5 (3654 examples) and 1 (1986 examples). The most underrepresented classes are 18 (59 examples), 14 (60 examples), 12 (80 examples), and 16 (86 examples). The confusion matrix (shown in figure 1) shows that the model learned to predict those heavily represented classes over the underrepresented ones. This leads to the differences in accuracy and costs shown by the graphs in figure 1. While the model performs relatively good on the training data, it does relatively poor on the validation set. This is because it mostly predicts the above mentioned over-represented classes which often appear in the training set. Therefore, a large share of those examples are correctly classified. The validation set contains 14 examples from each class but the bias towards the strong training classes till applies in the predictions for this set. Therefore, most of the examples in this set are wrongly predicted. The resulting values from the last epoch are shown in table 2.

Measure	**Training**	**Validation**
Cost	0.8929	2.9874
Accuracy	0.7536	0.2658

Table 2: Best results for imbalanced dataset

0.8929414416129362 0.7536114759066572 2.9874386748475197 0.26587301587301587



(a) Cost

(b) Accuracy

Predicted	1	2	3	4	5	7	8	9	10	11	15	All
Actual												
1	1655	0	0	0	59	0	0	0	0	103	169	1986
2	82	16	0	0	86	1	0	0	0	2	67	254
3	39	0	2	0	127	3	0	0	14	16	304	505
4	11	1	1	0	144	9	1	0	8	0	108	283
5	91	3	0	0	2555	22	2	1	31	8	941	3654
6	27	0	0	0	134	3	2	0	12	7	78	263
7	33	0	0	0	369	25	1	0	11	3	268	710
8	15	0	0	1	35	3	7	0	19	41	68	189
9	8	0	0	0	61	0	0	0	4	4	141	218
10	47	0	0	0	51	8	1	0	197	54	337	695
11	183	5	0	0	25	0	0	0	16	435	313	977
12	20	13	0	0	27	0	0	0	0	0	20	80
13	4	1	0	0	11	1	0	0	1	2	105	125
14	8	0	0	0	15	0	0	0	8	7	22	60
15	108	8	0	0	394	12	2	0	24	77	8745	9370
16	1	0	0	0	61	1	0	0	2	0	21	86
17	30	0	0	0	62	7	1	0	31	25	128	284
18	19	3	0	0	19	0	0	0	0	0	18	59
All	2381	50	3	1	4235	95	17	1	378	784	11853	19798

(c) Confusion Train

Predicted	1	3	5	10	11	15	All
Actual							
1	12	0	0	0	0	2	14
2	3	0	6	0	0	5	14
3	1	0	4	1	0	8	14
4	0	1	7	1	1	4	14
5	2	0	6	0	0	6	14
6	2	0	6	0	0	6	14
7	1	0	5	0	0	8	14
8	2	0	3	4	1	4	14
9	1	0	3	0	0	10	14
10	1	0	2	3	0	8	14
11	3	0	1	0	4	6	14
12	4	0	3	0	0	7	14
13	0	0	1	1	1	11	14
14	2	0	8	0	2	2	14
15	0	0	0	1	0	13	14
16	0	0	12	0	0	2	14
17	6	0	5	0	1	2	14
18	3	0	6	0	0	5	14
All	43	1	78	11	10	109	252

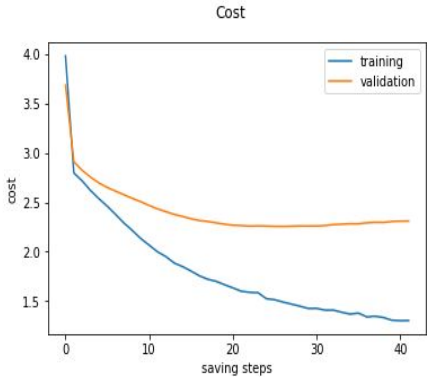
(d) Confusion Validate

Figure 1: Imbalanced dataset

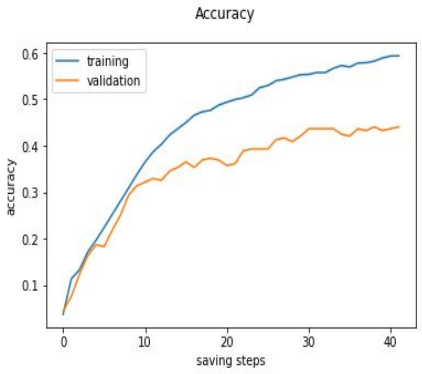
3.2 Balanced dataset

The result for the training with balanced data in table 3 and graphs in 3 show a decrease of *14.13%* in accuracy on Training and a an improvement of *22.12%* for the accuracy on Validation compared to the previous run. This is because given the balanced data, the network learns to predict all classes and not heavily focuses on the over-represented one as seen in the previous subsection. Therefore, it gets less examples in the unbalanced training set but shows a big improvement for the balanced Validation set. It is thus preferable to train the ConvNet on

balanced data since we want to train the network to predict all classes and not be biased on the distribution in the training set. The confusion matrices in 3 underline this observation b showing a more distributed prediction among the existing classes in both datasets.



(a) Cost



(b) Accuracy

Predicted	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	All
Actual																		
1	1354	83	0	60	0	57	0	73	88	12	193	0	0	24	0	0	42	1986
2	5	177	0	0	0	2	2	0	0	1	2	63	0	0	0	2	0	254
3	30	8	116	39	22	16	36	12	27	15	30	4	75	8	27	20	20	505
4	3	7	12	152	11	21	33	2	9	0	0	7	5	3	9	4	5	283
5	111	89	108	380	642	381	416	51	369	43	21	58	56	94	120	626	89	3654
6	6	4	4	22	18	125	18	6	11	8	2	6	0	7	2	11	13	263
7	40	15	13	92	38	42	305	7	39	11	3	16	15	4	24	36	10	710
8	3	0	1	0	1	2	0	160	1	6	3	0	0	9	0	1	2	189
9	5	5	4	7	10	12	9	0	139	7	3	0	1	1	2	13	0	218
10	20	8	2	4	1	31	3	23	1	406	38	3	1	73	2	11	68	695
11	79	24	18	1	0	10	9	15	2	65	674	13	22	20	5	2	18	977
12	0	14	0	0	0	0	0	0	0	2	0	64	0	0	0	0	0	80
13	1	0	10	3	0	2	2	1	5	0	4	3	87	1	1	2	3	125
14	1	0	0	1	0	1	0	1	0	3	1	0	0	47	0	1	4	60
15	337	68	372	236	334	252	431	80	425	132	154	40	280	50	5891	197	91	9370
16	1	6	0	3	7	5	2	1	4	0	0	1	0	1	0	52	3	86
17	5	4	7	5	4	17	4	9	3	31	15	3	2	53	0	3	119	284
18	1	26	0	0	2	2	5	0	1	0	0	21	0	0	0	0	1	59
All	2002	538	667	1005	1090	978	1275	441	1124	742	1143	302	544	395	6083	981	488	19798

(c) Confusion Train

Predicted	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	All
Actual																		
1	7	1	0	1	0	0	0	1	2	0	2	0	0	0	0	0	0	14
2	0	11	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	14
3	1	0	2	0	3	0	1	0	0	0	1	0	5	0	0	0	1	14
4	0	0	1	4	0	3	0	1	1	1	0	0	0	0	0	3	0	14
5	0	0	0	3	4	2	0	0	1	0	0	1	0	0	0	3	0	14
6	0	0	0	2	0	4	2	0	0	0	0	1	1	1	0	1	2	14
7	1	0	1	3	1	1	4	0	0	0	0	0	0	0	0	1	2	14
8	1	0	0	0	0	0	0	10	0	1	0	0	1	0	0	1	0	14
9	0	1	0	1	0	0	0	0	9	1	0	0	0	0	0	2	0	14
10	0	0	0	0	0	0	0	1	0	8	1	0	0	2	0	2	0	14
11	2	1	0	0	0	0	0	1	0	0	9	0	0	0	0	0	1	14
12	1	5	0	0	0	0	0	0	0	0	0	8	0	0	0	0	0	14
13	0	0	3	0	1	0	0	0	0	0	1	0	9	0	0	0	0	14
14	0	0	1	0	0	2	0	3	0	2	0	0	0	2	0	0	4	14
15	0	1	1	1	0	1	0	0	1	0	0	0	0	0	9	0	0	14
16	0	0	0	2	2	1	1	0	3	0	0	0	0	0	0	5	0	14
17	1	0	0	0	1	2	0	2	0	1	1	0	0	5	0	0	1	14
18	1	7	0	0	0	0	1	0	2	0	0	3	0	0	0	0	0	14
All	15	27	9	17	12	16	9	19	19	14	15	16	16	10	9	18	11	252

(d) Confusion Validate

Figure 2: Balanced dataset

Measure	Training	Validation
Cost	1.3046	2.2562
Accuracy	0.5935	0.4404

Table 3: Best results for balanced dataset

4 Efficiency gains

To make the ConvNet more efficient, the following improvements were implemented.

- Improvement 1: Pre-computing the MX_intput matrix for the first layer and encode each point as a sparse vector.
- Improvement 2: Using the generalized equation for the following layers.

The efficiency test was done using the following parameters and running the basic and the improved version of the ConvNet: *eta = 0.01, rho = 0.9, balanced = True, n1 = 20, n2 = 20, k1 = 5, k2 = 3, n_batch = 100 n_update = 10, n_epochs = 10*

The test is done running the code in Google Colaboratory. Table 4 compares the execution time values for both executions. It shows that my implementation for improvement 1 might be more inefficient than the promised efficiency gains. Therefore, I decided to only keep improvement 2 in both cases. All relevant parts for improvement 1 are kept as comments in the code and marked as such.

Version	Time
Basic	37 s
Improvement 1	3min 16s
Improvement 2	**28.8 s**
Improvement 1 + 2	2min 55s

Table 4: Execution times for basic and improved ConvNet

5 Improving performance

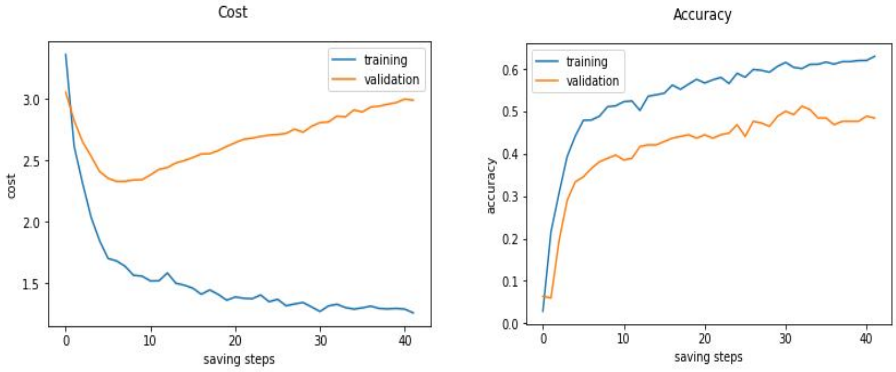
Due to the relatively long execution times, only two tests on different parameters for the filters were executed to elaborate their impact. The results compared to the setup from section 3.2 are stated in the following subsections. No excessive search on values for eta and rho was done. Finding good values for those could improve the prediction performance.

5.1 Setting the learning rate to .005

Setting the learning rate to .005 resulted in better values compared to the network compared to the setup shown in section 3.2. It might have been learning faster. For a good comparison, the results, including the graphs are shown below. To further improve this, I assume that a decaying learning rate might be useful.

Measure	**Training**	**Validation**
Cost	1.2547	2.3257
Accuracy	0.6295	0.5119

Table 5: Best results for learning rate .005



(a) Cost (b) Accuracy

Figure 3: Balanced dataset: results for learning rate .005

5.2 Increasing the number of filters for each layer

Instead of having 20 filters in each layers, those were increased to 30. This results in an improvement for Train and Validation.

Measure	Training	Validation
Cost	1.1016	2.1741
Accuracy	0.6699	0.4523

Table 6: Best results for 30 filters at each layer

5.3 Increasing the width of the filters

Instead of having k1 = 5 and k2 = 3, the ConvNet was trained on k1 = 10 and k2 = 6. Comparing this to the result in section 3.2, again improved performance for Train and Validation is visible.

Measure	Training	Validation
Cost	1.2739	2.2109
Accuracy	0.6119	0.4603

Table 7: Best results for increased width for the filters

5.4 Combining increased number of filters and width

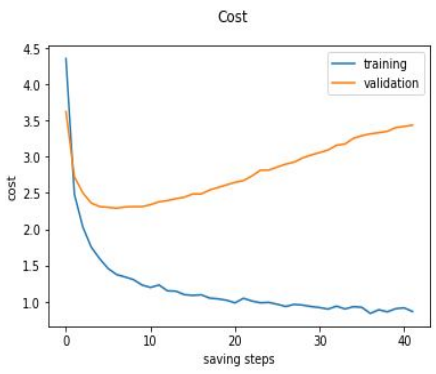
Combining both improvements (increased number of filters and width) shows a lower performance than the individual changes do.

Measure	Training	Validation
Cost	1.1943	2.29965
Accuracy	0.6410	0.4087

Table 8: Results for increased number of filters and width

5.5 Best performing ConvNet

Given the results from the test runs shown above, the setup in 5.2 performs best on Validation accuracy. More insight on this are therefore shown in figure 4. (However, setup 5.1 might be considered as better since its Validation accuracy is very close to 5.2 and it has lower costs.) The accuracy for each class can be calculated from the confusion matrix and is shown in table 9.



(a) Cost

(b) Accuracy

Predicted / Actual	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1	1986	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	223	0	0	1	0	1	0	0	0	1	27	0	0	0	1	0
3	0	3	350	3	30	8	34	2	9	9	11	0	8	2	20	3	13
4	1	2	2	245	4	4	19	0	1	0	0	0	1	0	0	1	3
5	59	29	132	137	1629	336	394	36	315	47	33	18	48	24	83	263	71
6	0	1	0	2	8	231	7	0	1	1	0	1	4	0	0	2	5
7	9	6	30	25	41	29	529	0	7	3	4	6	6	1	3	4	7
8	0	0	0	0	0	0	0	189	0	0	0	0	0	0	0	0	0
9	0	0	0	0	5	3	2	0	204	0	0	0	0	0	0	4	0
10	2	2	6	2	9	11	2	0	5	589	8	1	1	11	4	1	41
11	12	5	13	1	4	3	2	1	2	28	868	11	4	0	7	1	15
12	0	0	0	0	0	0	0	0	0	0	0	80	0	0	0	0	0
13	0	0	0	0	0	1	0	0	0	0	0	0	124	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	60	0	0	0
15	78	33	450	89	268	133	260	40	171	120	181	21	76	20	7323	39	68
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	86	0
17	1	0	0	0	6	5	3	1	2	20	2	0	1	25	0	0	218
18	1	23	0	5	0	1	4	0	3	0	1	18	1	0	1	0	1

(c) Confusion Train

Predicted / Actual	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1	14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	12	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0
3	0	0	7	1	0	1	2	0	0	0	0	0	2	0	0	1	0
4	0	0	1	8	1	1	0	0	0	0	1	0	1	0	0	0	1
5	0	0	0	0	8	2	2	0	0	0	0	1	0	0	0	1	0
6	0	0	1	1	1	5	1	1	0	0	1	0	0	0	0	0	3
7	0	1	0	3	2	1	7	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	1	1	0	9	0	2	0	0	0	1	0	0	0
9	0	0	0	0	1	2	1	0	6	1	0	0	0	0	1	2	0
10	0	0	0	0	0	0	0	0	1	0	7	2	0	0	2	0	0	2
11	2	1	0	0	0	0	0	0	0	0	0	9	1	0	1	0	0	0
12	1	5	1	0	0	0	0	0	0	0	0	0	7	0	0	0	0	0
13	0	0	3	0	0	0	0	0	0	1	0	0	0	7	0	3	0	0
14	0	0	1	0	1	0	0	1	1	0	0	0	0	0	0	0	0	10
15	0	1	1	0	0	0	1	0	0	0	0	0	0	0	0	11	0	0
16	0	0	1	0	9	0	1	0	1	0	0	0	0	0	0	0	2	0
17	0	0	1	0	1	1	0	2	0	2	0	0	0	3	1	0	3
18	0	7	0	0	0	0	1	0	3	0	1	1	0	0	0	1	0

(d) Confusion Validate

Figure 4: Setup from 5.2

Class	Country	Accurcay
1	Arabic	1
2	Chinese	1
3	Czech	0.4285
4	Dutch	0.5714
5	English	0.2142
6	French	0.3571
7	German	0.4285
8	Greek	0.6428
9	Irish	0.4285
10	Italian	0.6428
11	Japanese	0.6428
12	Korean	0.7142
13	Polish	0.5714
14	Portuguese	0.4285
15	Russian	0.1428
16	Scottish	0.7142
17	Spanish	0
18	Vietnamese	0.14285

Table 9: Accuracy of names from Validation

6 Applying the network to surnames

The surnames shown in table 10 are taken from contacts in my Facebook account. Special characters were removed the way it is described in the assignment instructions. The best performing ConvNet categorized those names as shown in the table. It has to be added that those names are taken from friends an categorized by their nationality. This does however not mean that the surname is of this origin, so the labels might not be 100% correct. The network achieves an accuracy of *53.84%* on the list of friends.

Surname	**Country**	**Prediction**
Rather	German	English
Pfeiffer	German	German
Lorenzo	Italian	Italian
Abdelhafez	Arabic	German
Shunnar	Arabic	German
Min	Korean	Chinese
Fernandez	Spanish	Spanish
Lusby	English	English
Sykorova	Czech	Czech
Kruglov	Russian	Russian
Nguyen	Vietnamese	Dutch
Allard	French	French
Charitaki	Greek	Japanese

Table 10: Execution times for basic and improved ConvNet