Kungliga Tekniska Högskolan

DD2424 Deep Learning in Data Science
Assignment 3

Quentin LEMAIRE

May 13, 2018

1 Gradient validation

In order to validate my computation of the gradient, I compared my values of the gradients with those of a numerically computed gradient function. To have a better estimate of the difference between both, I used the relative error. I also used the slower version of the numerical gradient to have more precise results.

The difference with the numeric one on the first batch is very good and I got the following values:

- gradW = 5.552617429659892e-10
- gradF1 = 5.402113545512228e-10
- gradF2 = 3.192067589829856e-10

On a second batch, I got the following results:

- gradW = 5.578825629390274e-10
- gradF1 = 5.734813851389323e-10
- gradF2 = 3.0999234203779523e-10

We can see that the gradients look very good and correct.

Finally, I tried to overfit the training data for a small subset of 100 images and for 200 epochs and a learning rate of 0.01, the loss on the training set went from 4 to 0.0008. It is an other proof that the gradients work correctly.

2 Imbalanced dataset

The confusion matrix shows us the consequence of the unbalanced dataset.

If we take a look at the dataset balance, we can see that some classes are over-represented. Here is the number of names by class from class 1 to 18: 1806, 236, 477, 260, 3312, 246, 641, 186, 203, 633, 880, 86, 123, 69, 8465, 91, 264 and 68.

We can see that the majority of the 18k names are in class 1, 5 and 15 where 15 is the biggest. If we print the confusion matrix after some iteration and for an accuracy of 5° % we have:

[388 0 0 0 1410 0 0 0 0 0 0 0 0 0 8 0 0 0]

[27 0 0 0 191 0 13 0 0 0 5 0 0 0 0 0 0 0]

[29 0 0 0 326 0 0 0 0 0 0 0 2 0 120 0 0 0]

[7 0 0 0 202 1 0 0 0 0 0 0 0 0 50 0 0 0]

[96 0 1 0 2985 0 2 0 0 0 27 0 3 0 198 0 0 0]

[11 0 1 0 199 0 0 0 0 0 0 0 0 0 35 0 0 0]

[35 0 0 0 546 0 1 0 0 0 2 0 2 0 55 0 0 0]

[9 0 1 0 108 0 0 0 0 0 3 0 0 0 65 0 0 0]

[4 0 1 0 159 0 0 0 0 0 0 0 0 0 39 0 0 0]

[69 0 1 0 495 0 2 0 0 1 5 0 0 0 60 0 0 0]

[100 0 3 0 693 0 0 0 0 0 19 0 0 0 65 0 0 0]

[9 0 0 0 72 3 0 0 0 0 2 0 0 0 0 0 0 0]

[5 0 0 0 58 0 0 0 0 0 0 0 0 0 60 0 0 0]

[8 0 0 0 58 0 0 0 0 0 0 0 0 0 3 0 0 0]

[169 0 3 0 2457 0 1 2 0 2 18 0 1 0 5812 0 0 0]

[2 0 0 0 87 0 0 0 0 0 0 0 0 0 2 0 0 0]

[21 0 0 0 222 1 0 0 0 0 4 0 0 0 16 0 0 0]

[16 0 1 0 44 0 3 0 0 1 3 0 0 0 0 0 0 0]

Where each row represents how the names of the corresponding class are classified. We can see that some classes are not well classified at all and that the 50% of accuracy comes from the overrepresented classes. Therefore, in order to increase the accuracy and to make the network classify correctly other classes, we need to overcome this unbalanced.

I decided to take a balanced subdataset balanced at the beginning of each epoch. We have the following confusion matrix with this improvement:

[1041 81 23 14 0 65 0 65 71 53 103 66 0 48 27 62 56 31]

[10 131 0 0 0 0 3 0 1 0 3 31 1 0 0 0 0 56]

[57 8 26 17 4 38 35 28 11 16 47 3 82 14 27 25 32 7]

[15 9 1 73 4 22 56 5 10 7 0 3 10 5 3 19 8 10]

[317 69 25 207 148 641 316 72 185 161 55 37 92 118 47 481 221 120]

[13 2 0 10 5 96 22 10 8 13 2 3 3 11 1 33 10 4]

[49 29 5 94 14 98 189 1 23 6 7 8 17 16 5 44 18 18]

[11 0 0 0 1 1 0 126 2 9 2 0 5 5 3 11 10 0]

[14 4 2 9 1 27 6 2 81 13 6 1 1 3 0 10 12 11]

[44 6 5 4 15 44 14 32 33 242 16 3 12 34 7 43 76 3]

[135 23 9 6 4 10 2 34 14 51 417 14 59 20 23 8 44 7]

[0 31 0 0 0 0 0 0 1 0 2 30 0 0 0 0 0 22]

[5 3 1 2 1 5 2 7 1 4 10 1 73 2 2 2 2 0]

[2 0 0 0 0 2 1 2 1 6 2 0 4 26 1 6 16 0]

[423 54 224 295 138 448 266 849 277 699 449 33 532 518 2671 161 396 32]

[6 3 2 5 4 16 6 1 5 5 1 1 0 1 0 27 5 3]

[16 8 5 15 2 19 8 10 10 24 15 1 8 21 5 21 75 1]

[1 13 0 0 0 1 2 0 0 0 0 11 0 0 0 0 0 40]

We can see that the less dominant classes are way better classified but it is not the case for the most dominant ones. This could be explain by the fact that less example of these classes have gone through the network. Therefore, we might need more iterations to classify correctly the dominant classes. With more epochs, I could achieve a better classification with this modification and, therefore, to less suffer from the imbalance otherwise my accuracy would be block by a threshold caused by the classes that are almost never well classified.

3 Comparison between the two regimes

I trained a network longer on those two possibilities. I took 10 000 iterations, not 20 000 because 10 000 was already very long on my computer (more than 30mn). Here are the two graphs I got:

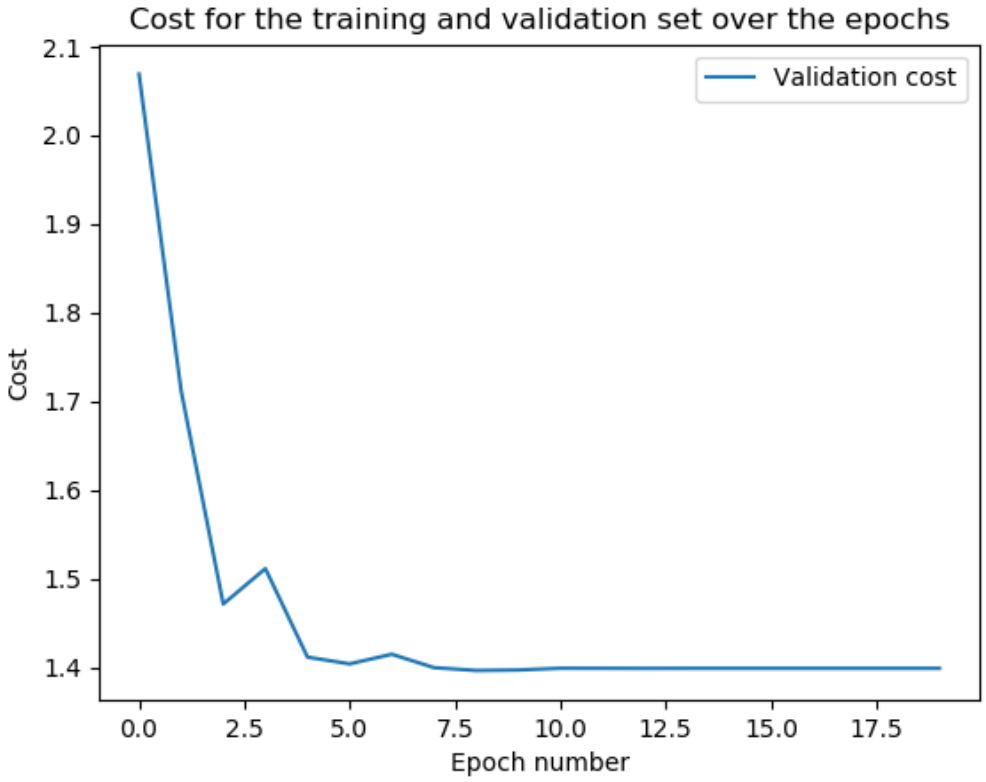

Figure 1: Validation cost over each update for the balanced train set. (edit: the actual legend is wrong)

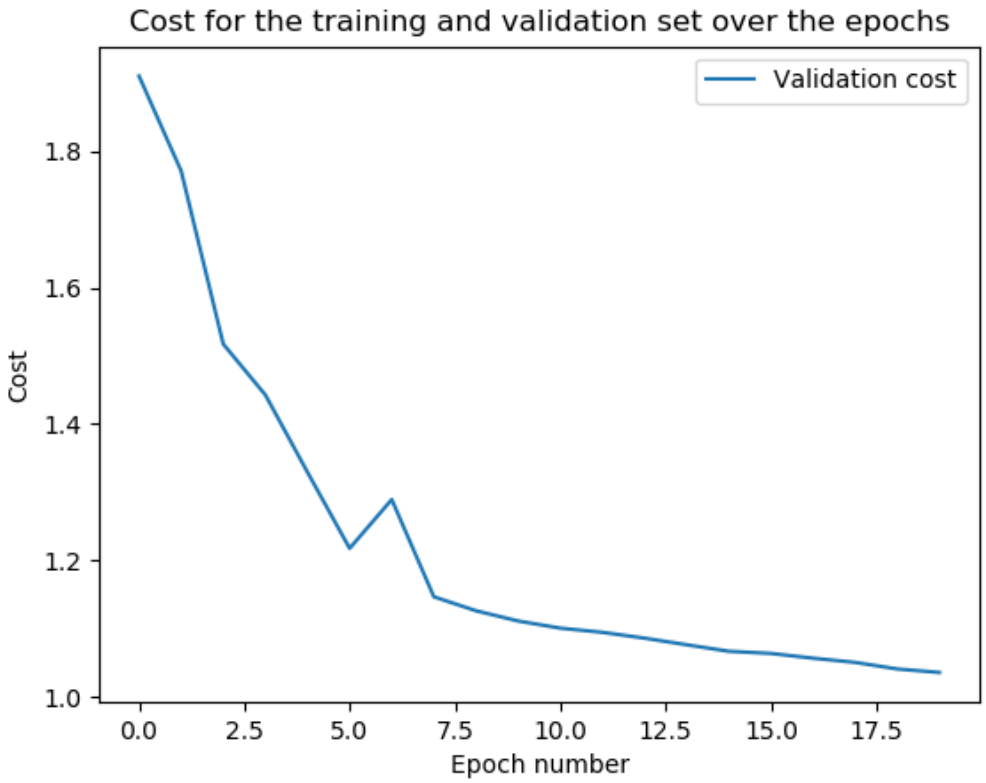

Figure 2: Validation cost over each update for the unbalanced train set. (edit: the actual legend is wrong)

I also got the following confusion matrices:

For the balanced train set :

[[1432 29 15 0 0 0 37 42 0 13 108 32 0 57 0 0 12 29]

[3 163 1 0 0 0 6 0 0 0 1 33 0 1 0 0 0 28]

[21 4 187 21 29 10 50 9 15 7 13 3 30 9 27 14 27 1]

[1 4 5 162 7 13 27 2 8 2 1 5 2 0 8 9 3 1]

[105 42 161 209 816 340 370 60 340 58 23 31 84 68 68 425 88 24]

[6 1 6 10 13 153 10 6 9 4 1 3 3 6 3 5 5 2]

[21 6 31 40 49 29 367 3 27 10 3 5 9 2 9 16 10 4]

[4 0 1 0 1 3 0 170 0 0 1 0 0 1 0 0 5 0]

[2 3 2 0 12 6 5 1 152 1 0 1 1 3 4 8 0 2]

[20 4 8 6 3 24 3 9 7 391 26 1 4 32 5 6 79 5]

[51 18 11 4 3 4 1 1 3 35 682 10 11 13 7 3 16 7]

[0 9 0 0 0 0 0 0 0 0 0 68 0 0 0 0 0 9]

[1 0 3 2 2 2 3 0 1 0 0 0 104 1 1 0 2 1]

[0 0 0 0 0 1 0 1 0 1 0 0 0 64 0 0 2 0]

[178 36 443 200 231 137 286 95 257 130 179 30 155 32 5859 89 94 34]

[0 3 0 0 2 0 1 0 2 0 0 1 0 0 0 81 1 0]

[7 3 2 2 6 9 3 8 2 20 8 0 2 40 1 2 148 1]

[0 11 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 55]]

For the unbalanced dataset:

[[1703 0 0 0 67 0 0 0 0 0 26 0 0 0 10 0 0 0]

[29 106 0 0 61 0 0 0 0 0 15 0 0 0 25 0 0 0]

[53 0 20 0 153 0 24 0 0 11 23 0 0 0 193 0 0 0]

[1 5 1 0 179 1 15 0 0 9 1 0 0 0 48 0 0 0]

[67 14 0 0 2868 0 54 1 0 18 8 0 0 0 282 0 0 0]

[13 1 1 0 165 2 7 0 0 10 11 0 0 0 36 0 0 0]

[23 5 4 0 340 1 141 0 0 2 3 0 0 0 122 0 0 0]

[15 0 2 1 44 0 1 27 0 40 17 0 0 0 39 0 0 0]

[4 1 0 0 121 0 1 3 0 4 7 0 0 0 62 0 0 0]

[90 0 0 0 119 0 2 1 0 296 76 0 0 0 49 0 0 0]

[152 11 0 0 58 0 0 0 0 21 582 0 0 0 56 0 0 0]

[13 24 0 0 32 0 0 0 0 1 6 0 0 0 10 0 0 0]

[5 0 3 0 24 0 1 0 0 5 11 0 0 0 74 0 0 0]

[10 0 0 0 27 0 1 0 0 10 14 0 0 0 7 0 0 0]

[138 17 7 1 580 0 28 3 0 75 87 0 0 0 7529 0 0 0]

[4 1 0 0 80 0 2 0 0 0 0 0 0 0 4 0 0 0]

[36 1 2 0 86 0 4 2 0 45 34 0 0 0 54 0 0 0]

[14 12 1 0 27 0 0 0 0 0 4 0 0 0 10 0 0 0]]

We can see that there is a big difference between the two confusion matrices. The one on the balanced dataset shows something quite equitable, both the dominant classes and the others are more or less well classified. Therefore, more mistakes coming from the dominant classes because or the number of data in those classes.

On the unbalanced one, this is totally the opposite. The dominant classes are very well classified but the other ones are almost never well classified.

We can see something quite problematic, the unbalanced one has less error even if some classes have almost 0% accuracy because the dominant ones are really too much dominant over the other one. The balance version will have a better accuracy on average over each class but it will have a worse overall.

4 Best performances

I have my best overall accuracy on a network with an unbalanced dataset. I can get more than 72% of accuracy in this case. But as we have seen just before, it is a "false good performance" because it totally forgets the non-dominant classes. Therefore, I have chosen to use a balance dataset and so to have a worse overall accuracy for a better average accuracy on all the classes.

I have trained a network on 2000 iterations and with the parameters k1 = 5, n1 = 50, k2 = 3, n2 = 50. I have also done some modification to the preprocessing and to the learning to have something better. First, I put all the names in lower case to reduce the dimensionality because I don't think we loss any information in this reduction. Secondly, I implemented the following weight decay: when the accuracy on the validation set goes up, the learning rate is divided by 10.

I have restrained myself to those parameters in order to have a reasonable computation time on my computer with my python implementation. I could have get better performances with more iteration and with bigger filters.

I got an overall accuracy of 59%. And on each class:

- 1 Arabic 87%
- 2 Chinese 73%
- 3 Czech 45%
- 4 Dutch 70%
- 5 English 33%
- 6 French 60%
- 7 German 49%
- 8 Greek 91%
- 9 Irish 76%
- 10 Italian 60%
- 11 Japanese 75%
- 12 Korean 77%
- 13 Polish 84%
- 14 Portuguese 93%
- 15 Russian 73%
- 16 Scottish 82%

- 17 Spanish 58%
- 18 Vietnamese 77%

We can see that the overall accuracy is lowered by some dominant classes that are not well classified like British.

5 Efficiency gains

I implemented the two improvements to compute more efficiently my gradients. Here is the time to compute the gradient of on mini batch before and after the improvement.

- With improvements: 0.03560924530029297 seconds.
- Without improvements: 0.05625128746032715

We can see that the improvement in pretty signification because it speed up the computation time by around 60%. Nevertheless, it is still quite slow to compute like I said in the section "Comparison between the two regimes".

6 Test on real names

I then tried the network on my names and the one of some of my friends, here are the name, the origin and the prediction.

- Lemaire, french origin and the prediction is french
- Yeramian, armenian origin and the prediction is jananese
- Vecchio, italian origin and the prediction is italian
- Del Castillo, spanish origin and the prediction is greek
- Larsson, swedish origin and the prediction is scottish
- Blain, french origin and the prediction is english
- Vinesse, french origin and the prediction is dutch
- Thiron, french origin and the prediction is english
- Blanchard, french origin and the prediction is german
- Gogoulou, greek origin and the prediction is portugese
- Sebahi, arabic origin and the prediction is italian

- Do, vietnamese origin and the prediction is korean

We can see that it does not work so well on the names I have chosen. Some can't work like Larsson because Swedish in not in the list but some really don't work like Del Castillo which is really Spanish. Some are quite close like Do.

It seems that the generalization is not that good. It can come from the data which have too few examples in some classes and, therefore, the network may not be able to really extract the important features of those classes.