Assignment 3 - Deep Learning DD2424

Erifili Ichtiaroglou

June 2018

1 Exercise 1

For the third assignment (2nd option) we had to implement a 2-layer convolutional neural network and train it. The first step was to calculate the gradients function for each layer. The calculation of the gradients was correct and checked with the help of the function NumericalGradients(). The error we got after comparing the numerical and analytical values of the gradients for the 2 convolutional layers and the fully connected layer was 2.396914104718218e-09. This error is adequately small and we can consider the implementation of the analytical function, backward_pass(), correct.

After that, we implemented the mini batch and we added the momentum term to speed up the training and obtain better results. The training was quite faster when using the momentum and the accuracy increased significantly.

In this lab we encountered for the first time unbalanced data, meaning that the number of samples for certain classes, was much bigger than the other classes. Thus, the ConvNet will most of the times predict the prevailing classes and still gain high accuracy. To compensate for this fact, we implemented the second option proposed in the assignment description and for every epoch of training we randomly sampled the same amount of data from each class. Below in Figures 1 and 2, we present the graphs of the validation loss and the accuracy for a network with k1=5, k2=3, n1=20 and n2=20 for both the cases of compensating and not for the unbalanced data.

Figure 1: Mini-batch training of ConvNet with 25000 update steps without compensating for the unbalanced data. $\eta = 0.001$, $batch = 100$,
accuracy at the validation set 70.9%

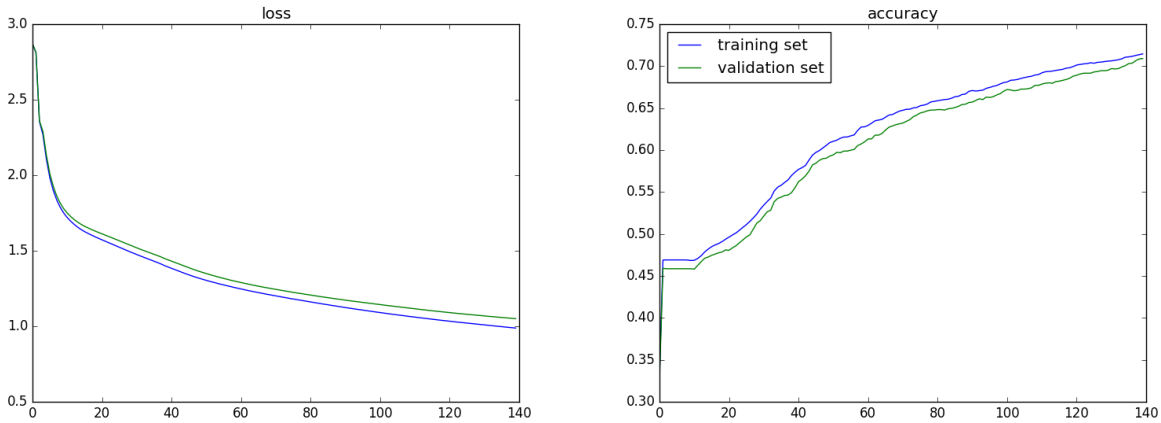
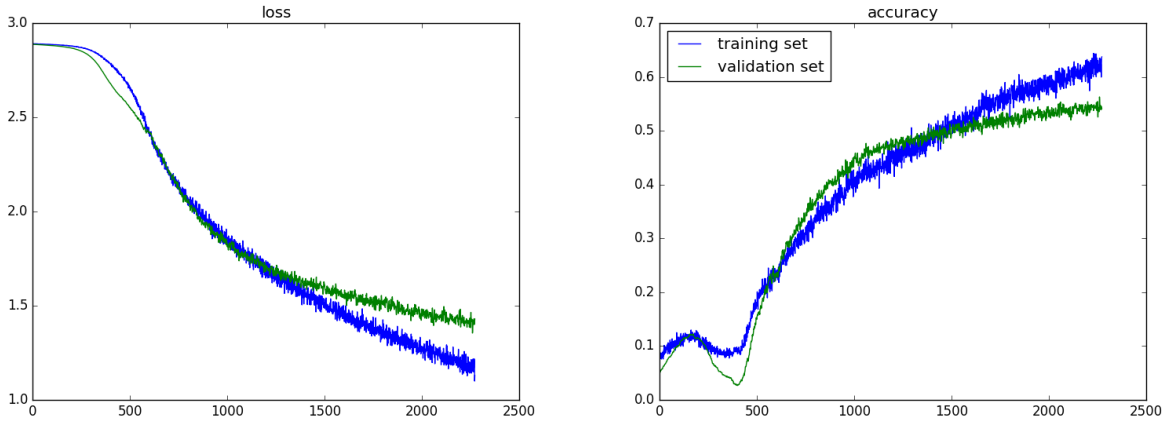


Figure 2: Mini-batch training of ConvNet with 25000 update steps and compensating for the unbalanced data. $\eta = 0.001$, $batch = 100$,
accuracy at the validation set 56%



From Figure 1 and Figure 2 we can see that the validation accuracy is higher for the unbalanced data. This fact is completely reasonable, since when both training and validation sets are unbalanced the network, that learned to predict the dominant classes, will have high accuracy on the unbalanced validation set. On the other hand, when we train on a balanced set and evaluate on unbalanced set we result in lower accuracy and higher loss. Table 1, below, shows the confusion matrices for the cases of training with unbalanced (left) and balanced (right) data respectively. As we can observe in the case of the unbalanced data there are some classes that are always predicted, such as class 5, 11, 15 and these are the dominant classes, whereas in the case of the balanced data, the predictions are more spread out among the classes. In addition, the number of correctly classified examples for each class is higher in the case of the balanced data.

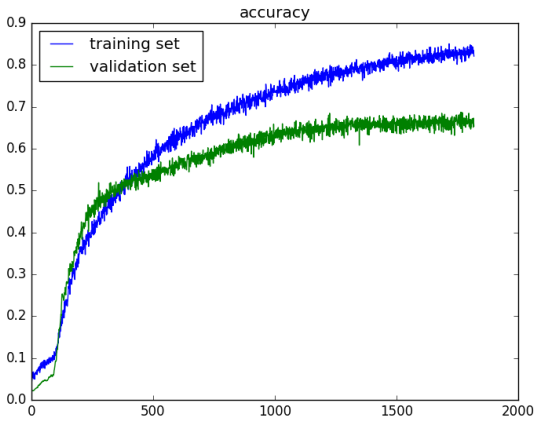
classes	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1	171				6						8				9			
2	10	11			10						1							
3	2				16						1				23			
4	4				20					1					12			
5	5				269		2			3	2				75			
6					20					1	2				8			
7	3				51		2			1					26			
8	2				2					5	3				5			
9					17										12			
10	6				5					30	5				30			
11	18	3			3					2	63				22			
12	2	1			4										1			
13				1						1				14				
14	1				2					1	1							
15	7	2			26					2	7				875			
16					8										1			
17	6				5					4	4				15			
18					5													

classes	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1	120	9	3			1	3	5	13	6	9	4	1	9	1	1	2	7
2		19							1			5						7
3			8	1	1		8	1		1	4		10	1	2	3	2	
4			4	10	2	2	11	2	1			1		1		2	1	
5	13	2	14	16	83	41	49	4	29	6	1	5	1	8	13	57	12	2
6	1		1		4	13	4	1	3			1				1	2	
7	3		4	11	8	2	40	2	3	1	1	1		2	1	3	0	1
8	1			1		1		13		1								
9	1	1			3	2			17				4			1		1
10	1		1			6		3		45	1			7		1	11	
11	14	6	2			1		1		3	76	1	1	2			4	
12							1					6						1
13			3	1	1								10				1	
14						1				2						1	1	
15	18	6	33	21	29	21	26	6	46	11	24	1	33	2	605	25	10	2
16					5											3	1	
17	1				2	1		3		3	2			9	2		11	
18																		5

Table 1: Confusion matrices of training the ConvNet for 25000 steps without compensating (left) and compensating (right) for the unbalanced data.

Continuing we tried to train our model with suitable parameters, so that we could gain the highest accuracy possible. For this task we only considered the case of the balanced data. In order to find the most appropriate number of filters per layer and the size of each filter, we implemented the function search_size(). Using this function we tried to train our model (500 steps) and observe the accuracy, for a wide number of filters sizes and numbers per layer. We achieved the best accuracy using n1=n2=20, k1=8 and k2=5. For these chosen parameters we then performed a new search (search_param()), in order to find the best values for the η and ρ parameters. Again after searching uniformly in a wide range of values we concluded to the following: $\eta = 0.003$ and $\rho = 0.9$. The validation accuracy achieved by this ConvNet is presented in Figure 3 and the accuracy for each class in Table 2.

Figure 3: Validation accuracy of the best model. Trained for 20000 update steps with $\eta = 0.003$, $batch = 100$ accuracy at the validation set 68.7%



Classes	Accuracy
1	96.9%
2	68.7%
3	38%
4	54%
5	36.7%
6	34.3%
7	54.5%
8	70.5%
9	58.6%
10	57.8%
11	78.3%
12	50%
13	43.7%
14	0%
15	75.1%
16	12.5%
17	38.2%
18	80%

Table 2: Accuracy of the best model for every class

In this assignment we implemented both of the efficiency gain techniques proposed in the description. We pre-calculated the Mx matrices for the first layer and we also implemented the generalized form of the function that calculates the Mx matrix. We performed a training of batch size 100 for 100 update steps, that took around 33,77 seconds before the speed up techniques and 15.47 seconds after incorporating them into the code.

The last task that we had to perform was to extract the probability vector of our best model, when it was applied to five surnames. For this task the surnames used were: *'Ichtiaroglou', 'Sager','Kassiou','Giramondi', 'Coppens'*. Ichtiaroglou and Kassiou are greek, Sager is german, Giramondi is italian and Coppens is french. The propability vector for these names is shown in Table 3 below. Ichtiaroglou is predicted as a polish surname, Sager as czech and Kassiou Giramondi and Coppens as russian.

Names	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
Ichtiaroglou	2.55e-6	5.71e-6	3.02e-2	1.4e-5	7.46e-4	6.56e-8	1.01e-6	1.72e-2	4.99e-5	5.52e-6	1.91e-1	1.64e-11	7.38e-1	9.11e-11	2.11e-2	4.82e-5	3.29e-9	4.99e-13
Sager	2.42e-7	1.66e-4	7.8e-1	5.86e-5	5.58e-2	1.48e-3	4.19e-2	3.33e-4	1.51e-3	4.62e-4	3.28e-5	3.33e-11	2.99e-3	1.44e-7	1.06e-1	1.34e-3	6.02e-3	4.85e-7
Kassiou	3.88e-5	4.12e-8	6.72e-2	1.09e-2	4.90e-2	6.13e-4	5.84e-3	2.52e-2	2.90e-1	3.51e-5	1.57e-3	8.63e-11	3.64e-2	7.66e-12	5.11e-1	8.32e-5	9.93e-8	3.52e-9
Giramondi	7.53e-4	4.54e-5	1.48e-1	8.69e-3	1.1e-1	1.92e-4	2.34e-2	2.38e-4	9.14e-3	1.87e-6	7.77e-4	3.88e-6	3.29e-1	2.62e-9	3.67e-1	3.89e-5	5.63e-8	7.12e-8
Coppens	3.62e-5	9e-7	5.34e-2	2.06e-2	1.99e-1	2.11e-3	8.55e-3	1.47e-5	1.53e-2	2.42e-4	7.05e-7	1.45e-11	2.78e-5	4.69e-7	6.92e-1	7.73e-3	9.48e-7	2.37e-7

Table 3: Probability vectors of the best model, when applied to the names *'Ichtiaroglou', 'Sager','Kassiou','Giramondi', 'Coppens'*.