Assignment 3: Neural networks

Adil Rhoulam

arhoulam@ens-paris-saclay.fr

Question 1.1: In your report, derive using the chain rule the form of the gradient of the logistic loss (3) with respect to the parameters of the network W_i, W_o, Bi and B_o. (i) First, write down the derivative of the loss (3) with respect to the output of the network $\bar{Y}$(X). (ii) Then write down the derivatives of the output $\bar{Y}$ with respect to the parameters W_o , B_o of the output layer, and (iii) then with respect to B_i and W_i. Note: present the complete derivations, not only the final results.

(i) We have s(Y, $\bar{Y}$(X))=log(1+exp(-Y. $\bar{Y}$(X))), then $\frac{\partial s(Y,\bar{Y}(X))}{\partial \bar{Y}} = \frac{-Y.exp(-Y.\bar{Y}(X))}{1+exp(-Y.\bar{Y}(X)}$

(ii) We know that $\bar{Y}$=W_oH+B_o then $\frac{\partial \bar{Y}}{\partial W_o} = {}^tH$ and $\frac{\partial \bar{Y}}{\partial B_o}$ = I where I is the unit vector with the same size of the size of B_o.

(iii) We have H=ReLU(W_iX+B_i) then:
$\frac{\partial \bar{Y}}{\partial W_i} = \frac{\partial (W_o ReLU(W_i X+B_i)+B_o)}{\partial W_i} = {}^tW_o.*\frac{\partial (ReLU(W_i X+B_i))}{\partial W_i} = {}^tW_o.$*ReLU'($W_i$X+$B_i$). ${}^tX$
and
$\frac{\partial \bar{Y}}{\partial B_i} = \frac{\partial (W_o ReLU(W_i X+B_i)+B_o)}{\partial B_i} = {}^tW_o.*\frac{\partial (ReLU(W_i X+B_i))}{\partial B_i} = {}^tW_o.$*ReLU'($W_i$X+$B_i$)
Where ReLU' = $\mathbf{1}_{R++}$ and .* is the pointwise product.
Finally we can use the chain rule to determinate the derivative of the logistic loss function w.r.t the network parameters.

Question 1.2.A: In your report, write down the general formula for numerically computing the approximate derivative of the loss s(θ), with respect to the parameter θ_i using finite differencing. Hint: use the first order Taylor expansion of loss s(θ+$\Delta\theta$) around point θ.

(i) The general formula for numerically computing the approximate derivative of the loss s(θ), where θ represent all the parameters of s (Wi, Wo ... in our case). Using finite differencing w.r.t a parameter θ_i in R we should fix all the other parameters that are not θ_i and then we get the formula:
s(θi+$\Delta\theta i$)=s(θi)+$\Delta\theta i.\frac{\partial s(\theta i)}{\partial \theta i}$+o($\Delta\theta i$)

Grad of s	True value	Approx value	Absolute error	Relative error
w.r.t W_i	[[-0.0045 , -0.0326] [-0.0075 , -0.0538] [-0.0013 , -0.0095]]	[[-0.0045 , -0.0326] [-0.0075 , -0.0538] [-0.0013 , -0.0095]]	1.0e-04 * [[0.0013 , 0.0670] [0.0035 , 0.1822] [0.0001 , 0.0057]]	1.0e-03 * [[0.0285 , 0.2053] [0.0471 , 0.3385] [0.0083 , 0.0600]]
w.r.t W_o^T	[-0.0543 -0.1368 -0.1380]	[-0.0543 -0.1367 -0.1379]	1.0e-03 *[0.0185 0.1176 0.1197]	1.0e-03*[0.3415 0.8599 0.8675]
w.r.t b_i	[−0.0357 -0.0589 -0.0104]	[-0.0357 -0.0589 -0.0104]	1.0e-04*[0.0803 0.2184 0.0069]	1.0e-03 *[0.2248 0.3706 0.0657]
w.r.t b_o	-0.0736	-0.0736	3.4098e-05	4.6305e-04

Table 1: table showing the values of gradient, their approximate derivative and the absolute, relative errors

Question 1.2.B: In your report, choose a training example X, Y and report the values of the analytically computed gradients : grad_s_W_i, grad_s_Wo, grad_s_bi, grad_s_bo as well as their numerically computed counterparts: grad_s_Wi_approx , grad_s_Wo_approx, grad_s_bi_approx, grad_s_bo_approx. Are their values similar? Report and discuss the absolute and relative errors.

(i) For the values: X=$[0.1270, 0.9134]^T$, Y=1, W_i=[[0.6324 , 0.5469] ; [0.0975 , 0.9575] ; [0.2785 , 0.9649]], b_i=$[0.1576, 0.9706, 0.9572]^T$, W_o=[0.4854 , 0.8003 , 0.1419] and b_o=0.4218 . We get the results reported in table 1

(ii) Using a delta=0.0001, we can see that the finite difference approximate well the true value of the grading, the absolute and the relative error is around a factor of 10^{-4} - 10^{-3} (10^{-5} for some parameters), then the precision of the approximation is not bad, if we want more precision we could choose a delta which is smaller.

Question 1.3.A: Include the decision hyper-plane visualization and the training and validation error plots in your report. What are the final training and validation errors? After how many iterations did the network converge?

(i) The visualization of the decision hyper-plane is shown in Figure1 and Figure2 is showing the training and validation error plots.

(ii) The final training and validation errors are equal to 0.
For the training error, there a range of iterations where it's equal to 0 but it increases after like from the iteration 27557 to 27578, but we can find the iteration where the convergence to 0 began by searching for the last element of the list find(tr_error =0), we find that the last iteration where the training error is not 0 is 68985 the the convergence to 0 begin after the iteration 68986 so the network converges after that iteration.

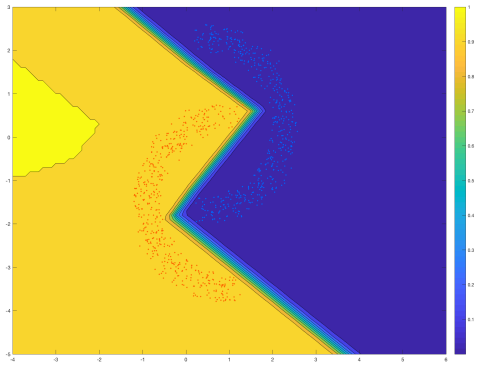
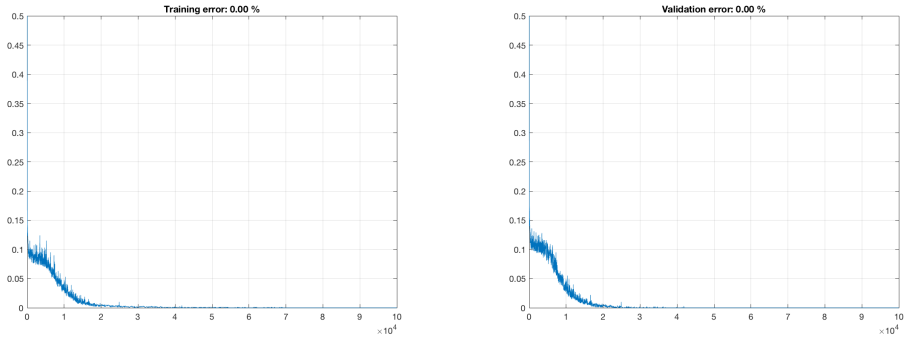


Figure 1: Decision hyper-plane for h=7 and learning rate= 0.02



(a) training error (b) Validation error

Figure 2: Evolution training and validation error as functions of iteration.

Question 1.3.B: Random initializations. Repeat this procedure 5 times from 5 different random initializations. Record for each run the final training and validation errors. Did the network always converge to zero training error? Note: to speed up the training you can set the variable visualization_step = 10000. This will plot the visualization figures less often and hence will speed up the training.

(i) By changing the visualization_step to 10000, for 5 random initialization we have recorded the final training and validation errors in table 2.

(ii) We could see that the values of error don't converge always to 0, it depends on the initialization and get trapped on different local minimas.

Trail	Final Record train error	Final Record validation error
Trail 1	0.00%	0.00%
Trail 2	0.00%	0.00%
Trail 3	6.70%	7.60%
Trail 4	7.10%	8.20%
Trail 5	6.80%	9.40%

Table 2: Final training and validation error percentage for 5 different random initialization.

lrate: trail	Final train error	Final valid error	Convergence iteration
2 Trail 1	50.00%	50.00%	104
2 Trail 2	50.00%	50.00%	99999 (didn't converge oscillate)
2 Trail 3	50.00%	50.00%	99999 (didn't converge oscillate)
0.2 Trail 1	0.00%	0.00%	13184
0.2 Trail 2	0.00%	0.00%	9827
0.2 Trail 3	0.00%	0.00%	25291
0.02 Trail 1	6.50%	8.60%	99999 (didn't converge oscillate)
0.02 Trail 2	7.60%	7.90%	99999 (didn't converge oscillate
0.02 Trail 3	0.00%	0.00%	60477
0.002 Trail 1	0.10%	0.00%	274234 (I added more iterations till 300K)
0.002 Trail 2	7.10%	7.90%	299991 (oscillate)
0.002 Trail 3	0.10%	0.00%	240953

Table 3: Final training and validation error percentage for 3 different lrate

Question 1.3.C: Learning rate. Keep h=7 and change the learning rate to values lrate=2,0.2,0.02,0.002. For each of these values run the training procedure 3 times and observe the training behaviour. For each run and the value of the learning rate report: the final (i) training and (ii) validation errors, and (iii) after how many iterations the network converged (if at all). Briefly discuss the different behaviour of the training for different learning rates.

(i) The table 3 shows the results demanded.

(ii) We can say that for a large values of lrate (2), the network didn't converge (walk over the minimas because the steps are large = 2) in trail 2 and 3) and when it converges, it's done rapidly but still with a large error. For values 0.2 and 0.02, when the network converges it takes much more iterations (13184 for the first trail with the value 0.2 and 60477 for the value 0.02), when the value is very small it's more likely that the network get trapped in the local minimas. For 0.002 of learning rate, we can see that the convergence takes many more steps to converge (274234 iterations for trail 1) and it's get often trapped in local minimas.

Num of hidden units: trail	Final train error	Final valid error	Convergence iteration
1 Trail 1	9.20%	11.60%	99999 (didn't converge oscillate)
1 Trail 2	8.10%	10.30%	99999 (didn't converge oscillate)
1 Trail 3	8.40%	10.30%	99999 (didn't converge oscillate)
2 Trail 1	8.50%	10.60%	99999 (didn't converge oscillate)
2 Trail 2	9.20%	8.40%	99999 (didn't converge oscillate)
2 Trail 3	9.10%	10.40%	99999 (didn't converge oscillate)
5 Trail 1	0.00%	0.00%	59285
5 Trail 2	7.50%	8.80%	99999 (didn't converge oscillate)
5 Trail 3	9.20%	9.20%	99999 (didn't converge oscillate)
7 Trail 1	0.00%	0.00%	80457
7 Trail 2	7.60%	7.90%	99999 (didn't converge oscillate
7 Trail 3	0.00%	0.00%	60477
10 Trail 1	0.00%	0.00%	57669
10 Trail 2	0.00%	0.00%	51053
10Trail 3	0.00%	0.00%	63781
100 Trail 1	0.00%	0.00%	47058
100 Trail 2	0.00%	0.00%	36221
100 Trail 3	0.00%	0.00%	35320

Table 4: Final training and validation error percentage for 3 different lrate

Question 1.3.D: The number of hidden units. Set the learning rate to 0.02 and change the number of hidden units h=1,2,5,7,10,100. For each of these values run the training procedure 3 times and observe the training behaviour. For each run and the value of the number of hidden units record and report: the value of the final (i) training and (ii) validation error, and (iii) after how many iterations the network converged (if at all). Discuss the different behaviours for the different numbers of hidden units.

(i) The table 3 shows the results demanded.

(ii) We could see that as we increase the number of hidden units, the network becomes well adjusted to the training data in few steps, but we have to learn more and more parameters, we notice also that when we increase the number of hidden units (10,100) the network doesn't get trapped in local minimas often unlike the case of 5, 7. For 1, 2 hidden units, the network is not flexible to the training data, so the bad training and validation errors. So in conclusion for a large number of hidden units the network becomes well flexible with the training data in few iterations but we have to learn more parameters, for small values of hidden units, the network doesn't adapt to the training data and get trapped in local minmas often. For medium values, we control a little bit the trade-off between flexibility and the number of parameters to learn.

Question 2.1: i. What filter have we implemented? ii. How are the RGB colour channels processed by this filter? iii. What image structure are detected?

(i) We have implemented the Laplace 2D filter which calculate the sum of the second derivatives along the two axis x and y.
(ii) Each RGB colour channel is convolved with a LaPlacian filter separately and then combining (summing) the results.
(iii) The image structures detected are the edges.

Question 2.2: Some of the functions in a CNN must be non-linear. Why?

(i) Some of the functions in a CNN must be non-linear because if note the CNN can be reduced to a two layer neural network as in Part 1 (because in each layer of the CNN we would multiply the input by a matrix and translating it with a bias if there exists and then this operation repeated in each layer can be reduced to a one hidden layer which multiply directly the input by the product of the matrices of each of the CNN layers and the bias will be a combination of product and sums between the matrices and biases of each CNN layer.).

Question 2.3: Look at the resulting image. Can you interpret the result?

(i) We can see from the picture that the max-pooling operation is applied using a square filter (of size 15 in our case), this operation helps the image to be approximately invariant to small changes in the input image (small translations, rotations) this means that the pooling of the same image with a small rotation or translation will result the same pooled image (the maximum will stay in the same square but not in the same place and then the output of approximately all the pooled outputs doesn't change), the pooling operation is also useful because it reduces the dimension of the image while keeping the important local informations.

Question 3.1.A: The derivatives $\frac{\partial f}{\partial w_l}(x_0;w_1,....,w_L)$ (derivatives of the loss with respect to any parameters w_l) have the same size as the parameters w_l . Why?

(i) the loss function f has values in R, then the gradient of f w.r.t w_l is simply a vector of size the size of dim(R)*size(w_l)=1*size(w_l)=size(w_l) and its elements are the derivatives of f w.r.t to the components of w_l. So the derivatives w.r.t to any parameters w_l have the same size as the parameters w_l because f is a R valued function.

Question 4.2.A: By inspecting initializeCharacterCNN.m get a sense of the architecture that will be trained. How many layers are there? How big are the filters?

(i) The CNN initialized by the function initializeCharacterCNN contains 4 operations of convolutions (the first two convolutions are followed by two max-pooling operations, the third one followed by the operation of the non-linear function Relu and the last convolution followed by the application of the softmaxloss function.). There are then 8 layers + the input. In the first convolution, we create a bank of 20 filters of dimension 5x5x1 and null bias for each filter. In the second convolution, we create a bank of 50 filters of dimension 5x5x20 and null bias for each filter. In the third convolution, we create a bank of 500 filters of dimension 4x4x50 and null bias for each of the 500 filter

and in the last convolution, we create a bank of 26 filters of dimension 2x2x500 and null bias for each filter.

Question 4.2.B: i. Understand what the softmax operator does. Hint: to use the log-loss the data must be in the (0, 1] interval. ii. Understand what is the effect of minimizing the log-loss. Which neuron's output should become larger?

(i) The softmax operator takes a vector of real-valued elements and transform it to a vector of elements between [0,1] by taking for each elements it exponential and then normalizing it by the sum of exponentials of all the elements. It can then have a probabilistic interpretation also, for example the letter which has the highest softmax value should be the assigned letter. Then, it become possible to use a log-loss with the transformed data from real valued to bounded value between [0,1] and that sums to one.

(ii) The gradient of the log loss has the effect to exactly update the weights of the softmax classifier in the right direction using gradient descent, then the neuron output that has the smaller log loss function becomes the larger.

Question 4.3: Run the learning code and examine the plots that are produced. As training completes answer the following questions: i. There are two sets of curves: energy and prediction error. What do you think is the difference? What is the 'energy'? ii. Some curves are labelled 'train' and some other 'val'. Should they be equal? Which one should be lower than the other? iii. Both the top-1 and top-5 prediction errors are plotted. What do they mean? What is the difference?

(i) The difference is that to get a good performance and accuracy, we have to maximize the energy, from the other side we have to minimize the prediction error. The energy could be the log likelihood.

(ii) The curve labelled train measures the error on train data, and the one labelled val measure it on validation data. There is no reason why they should be equal, because we train our CNN to minimize the train error and then we test the learned parameters on validation data (cross-validation technique). Normally, the error should be lower for train data, because we train our CNN to become flexible to the train data and we test it on validation data, we repeat this process while the validation error decreases and then stop to have the best trade-off between under-fitting (model not flexible on train data) and overfitting (model not flexible on test data).

(iii) the top-1 prediction error compares only the top predicted label (the one having the highest probability) with the target label while the top-5 checks if the target label exist among the 5 top prediction (the 5 ones with the highest probabilities). Then it's logical that the train/val error of the top5err are lower than the ones of the top1err.

Question 4.4: what can you say about the filters?

(i) The filters are learned to minimize the error and then to extract the most relevant features in the input images, we could see that each filter has its own properties of filtering, each filter extract then a specific feature from the image.

Question 4.5.A: The image is much wider than 32 pixels. Why can you apply to it the CNN learned before for 32x32 patches? Hint: examine the size of the CNN output using size(res(end).x).

(i) The image size is 32x576 and then doesn't have the same dimension as the trained images for the CNN, When applying it to the CNN, the dimension of the output in each layer w.r.t to the second axis, which has 576 dimension, changes because will apply more convolutional operations along this axis (the dimension of the second axis can be computed by (576-F)/stride+1 for the first layer which is 572 by inspecting size(res(2).x)) and so on for the next layers, finally, we get an output of dimension 1x137x26 which has larger dimension w.r.t the second axis due to the large dimension of the second axis of the output 576, we can notice that for the first axis the dimension of the output is always 1 because we preserved the same dimension as training images 32.

Question 4.5.B: Comment on the quality of the recognition. Does this match your expectation given the recognition rate in your validation set (as reported by cnn_train during training)?

(i) The quality of the recognition is bad, the letter that are predicted aren't the correct ones, even if the validation error were around 7% which is not a huge error. So those results didn't match the expectations if we rely only on the validation error.

Question 4.6: i. Look at the training and validation errors. Is their gap as wide as it was before? ii. Use the new model to recognize the characters in the sentence by repeating the previous part. Does it work better?

(i) The gap between val error and train error has decreased comparing it to the previous part.
(ii) This model works better, it recognizes well the characters of the sentence.

Question 5.1: i. Show the classification result. ii. List the top-5 classes.

(i) The classification result is shown in Figure 3 with a score of 0.704 (highest probability) for the bell pepper class.
(ii) The top-5 classes are shown in Figure 4, in particular the top five classes are: ['bell pepper' , 'cucumber, cuke' , 'orange' , 'lemon' , 'corn'].

Question 5.2: Report your AP classification results for the three object classes. In the same table, compare your results using CNN features to your results obtained in Assignment 2 for corresponding object classes on the testing image set. What conclusions can you draw?

(i) Figure 5 shows the Precision-recall curves of test data using CNN features as input using as SVM parameter C=1.
(ii) Figure 6 shows the Precision-recall curves of test data of assignments 2 using as SVM parameter C=1.
(iii) Table 5 shows the comparison of AP classification results for the three object classes between the CNN features as input and the results using the results of Assignment 2 with an SVM parameter C=1.

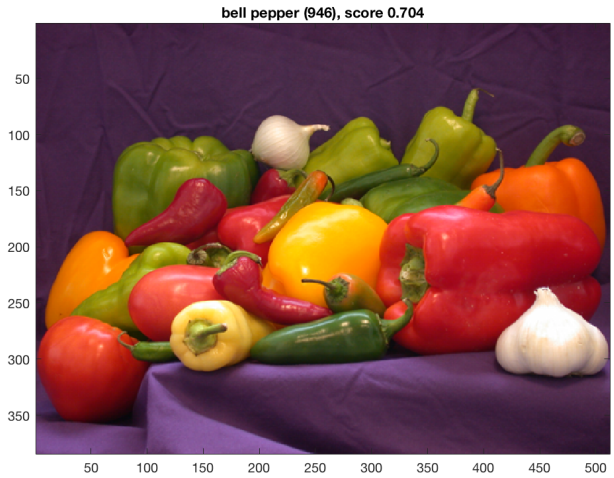


Figure 3: Best classification result

Class/Input	Assignement 2	CNN features
Airplane AP	0.5495	0.9370
Motorbike AP	0.4376	0.9288
Person AP	0.6587	0.9566

Table 5: Comparison between AP results using CNN features and using Assignment 2 results (C=1)

(iv) We can see that the AP results when using the CNN features is much better than the simple approach, we achieve an AP above the 90% using the CNN features.

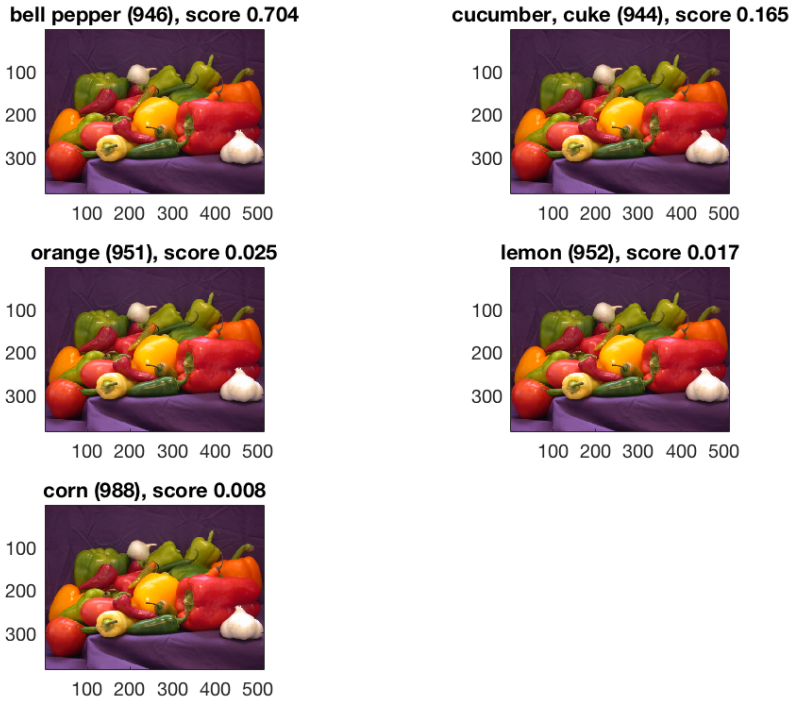
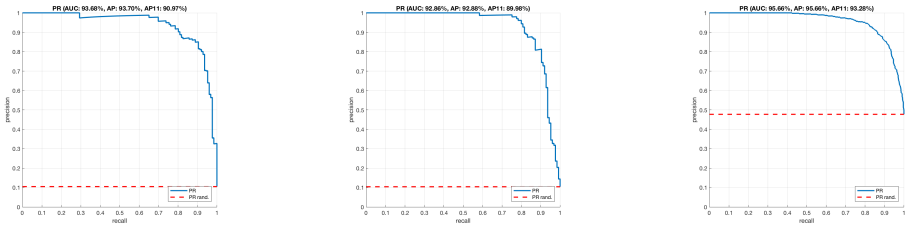
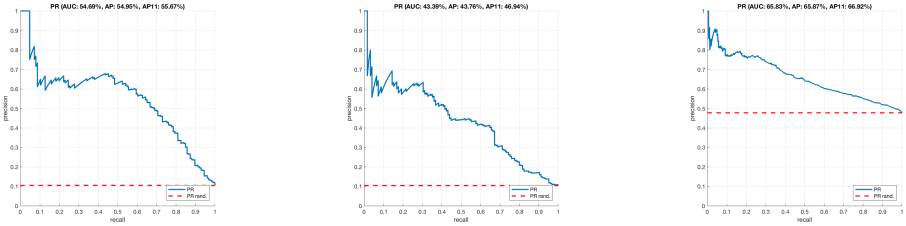


Figure 4: Top 5 classification result



(a) Airplane classe (b) Motorbike class (c) Person class

Figure 5: The Precision-recall curves for the test data of all the three classes using CNN features with C=1



(a) Airplane classe (b) Motorbike class (c) Person class

Figure 6: The Precision-recall curves for the test data of all the three classes of assignment 2 with C=1