Assignment 3: Neural networks

Rémi Lespinet

remi.lespinet@ens-paris-saclay.fr

I Training a fully connected neural network

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, B_i 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.

First computing the derivative of the logistic loss function $s(Y, \bar{Y})$ with respect to $\bar{Y}$, which is a scalar function of one variable, we obtain

$$\frac{\partial s}{\partial \bar{Y}}(Y, \bar{Y}) = \left(\frac{1}{1 + \exp\left(-Y\bar{Y}\right)}\right)\left(-Y \exp\left(-Y\bar{Y}\right)\right)$$

Which simplifies to

$$\frac{\partial s}{\partial \bar{Y}}(Y, \bar{Y}) = -\frac{Y}{1 + \exp\left(Y\bar{Y}\right)}$$

Since

$$\bar{Y}(X) = W_o H + B_o = \sum_{i=1}^{h} W_o^{(i)} H_i + B_o \qquad (\star)$$

The derivative of $\bar{Y}$ with respect to W_o^i is

$$\frac{\partial \bar{Y}}{\partial W_o^i} = h_i$$

Hence writing as a vector form,

$$\frac{\partial \bar{Y}}{\partial W_o} = h$$

The derivative of $\bar{Y}$ with respect to B_o is

$$\boxed{\frac{\partial \bar{Y}}{\partial B_o} = 1}$$

Now,

$$\bar{Y}(X) = W_o \, \mathrm{ReLU}\,(W_i X + B_i) + B_o$$

$$\frac{\partial \, \mathrm{ReLU}}{\partial x}(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x > 0 \end{cases}$$

Let $l_j(W_i)$ be the j^{th} line of W_i

$$W_i = \left[\begin{array}{c} l_1(W_i) \\ \hline l_2(W_i) \\ \hline \vdots \\ \hline l_{N-1}(W_i) \\ \hline l_N(W_i) \end{array}\right]$$

We have

$$H_j(X) = \mathrm{ReLU}\,(l_j(W_i)^T X + B_i^j)$$

Since

$$\frac{\partial (l_j(W_i)^T X + B_i^j)}{\partial W_i^{(k,l)}} = \begin{cases} X_l & \text{if } k = j \\ 0 & \text{otherwise} \end{cases}$$

and

$$\frac{\partial (l_j(W_i)^T X + B_i^j)}{\partial B_i^k} = \begin{cases} 1 & \text{if } k = j \\ 0 & \text{otherwise} \end{cases}$$

Hence

$$\frac{\partial H_j}{\partial W_i^{l,k}} = \begin{cases} X_l & \text{if } k = j \quad \text{and} \quad l_j(W_i)^T X + B_i^j > 0 \\ 0 & \text{if } k \neq j \quad \text{or} \quad l_j(W_i)^T X + B_i^j < 0 \end{cases}$$

and

$$\frac{\partial H_j}{\partial B_i^k} = \begin{cases} 1 & \text{if } k = j \quad \text{and} \quad l_j(W_i)^T X + B_i^j > 0 \\ 0 & \text{if } k \neq j \quad \text{or} \quad l_j(W_i)^T X + B_i^j < 0 \end{cases}$$

By $\star$,

$$\boxed{\frac{\partial \bar{Y}}{\partial W_i^{l,k}} = \begin{cases} W_0^k X_l & \text{if } l_k(W_i)^T X + B_i^k > 0 \\ 0 & \text{if } l_k(W_i)^T X + B_i^k < 0 \end{cases}}$$

and

$$\frac{\partial \bar{Y}}{\partial B_i^k} = \begin{cases} W_0^k & \text{if } l_k(W_i)^T X + B_i^k > 0 \\ 0 & \text{if } l_k(W_i)^T X + B_i^k < 0 \end{cases}$$

In matlab, we can compute gradient of the logistic loss function with respect to each variable very simply using component wise multiplication :

```
function [grad_s_Wi, grad_s_Wo, grad_s_bi, grad_s_bo] = ...
                                gradient_nn(X,Y,Wi,bi,Wo,bo)
    H = relu(Wi * X + bi);
    Y_bar = Wo * H + bo;
    D_loss = - Y / (1 + exp(Y * Y_bar));

    T = (Wi * X + bi) > 0;

    grad_s_Wi = D_loss * transpose(X * Wo) .* T;
    grad_s_bi = D_loss * transpose(Wo) .* T;
    grad_s_Wo = D_loss * transpose(H);
    grad_s_bo = D_loss * 1;

end
```

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

From the Taylor expansion at order 1, we have

$$s(\theta + \Delta\theta) = s(\theta) + \nabla s(\theta)^T \Delta\theta + o(\Delta\theta)$$

In particular, if we consider the canonical basis $(e_1, \dots e_K)$ such that $e_i^j = \delta_{(i,j)}$.

$$\forall i \in \{1, \dots, K\},\ s(\theta + \xi e_i) = s(\theta) + \xi \frac{\partial s}{\partial \theta_i}(\theta) + o(\xi)$$

Hence,

$$\frac{\partial s}{\partial \theta_i}(\theta) = \frac{s(\theta + \xi e_i) - s(\theta)}{\xi} + o(1)$$

We can compute the gradient by applying this equation for each of the variables $W_i^{(j,k)}$, $W_o^{(k)}$, $b_o^{(k)}$ and b_i.

Question 1.2.B: In your report, choose a training example $\{X, Y\}$ and report the values of the analytically computed gradients : `grad_s_Wi`, `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.

The figure 1 shows the euclidian distance between the analytic and numeric gradient for the training point 1 (absolute error). We can see pics between 0 and 10000 iterations. This happens when $l_j(W_i)^T X_1 + B_i^j$ is very close to 0, in which case it may be that $l_j(W_i)^T X_1 + B_i^j$ is negative for the numeric gradient and positive for the analytic one (or the opposite) which results in thresholding in one case and not the other and causes a much larger gradient difference than usual. (Note that it may happen later in the iterations, but since the gradient is decreasing, we don't see it in the curve).

Otherwise, we see that the numeric version of the gradient is very close to the analytic one (the euclidian distance between the 2 is less that 10^{-4} in our case for $\xi = 10^{-4}$). The relative error plot is represented in figure 1. We observe the same pics as in the absolute case. And we see that the relative numeric gradient error is almost constant with a value on the order of 10^{-4} after convergence. This illustrates the values of the numeric and analytic gradient are very similar.

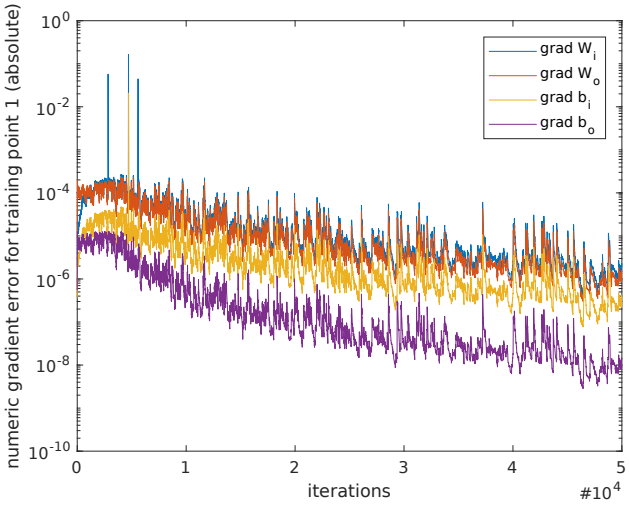


Figure 1: Representation of the absolute numeric gradients error of the training point 1 with respect to the parameters at each iteration, for $\xi = 10^{-4}$

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?

The decision hyper-plane is represented in figure 3. The training and validation error plots are shown in figure 4.

Table 1 presents the minimum number of iteration needed in order for all future iterations (in range 1-50000) to be above a given accuracy (on the validation data). Note that it is only valid in the range 1-50000 iterations. In particular for 100% accuracy, it may well be possible that if we would have prolonged the experience, there would have been a value of the accuracy below 100%. Here we only tell that from iteration 39594 to iteration 50000, the value of accuracy on the validation data is always 100% (or equivalently there is no misclassified point in the validation dataset).

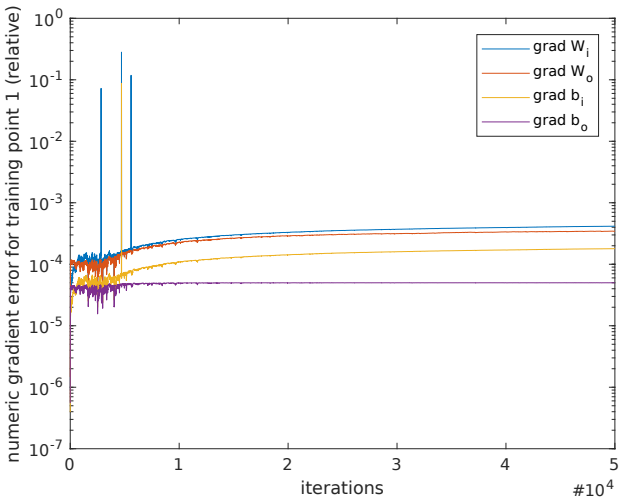


Figure 2: Representation of the relative numeric gradients error of the training point 1 with respect to the parameters at each iteration, for $\xi = 10^{-4}$

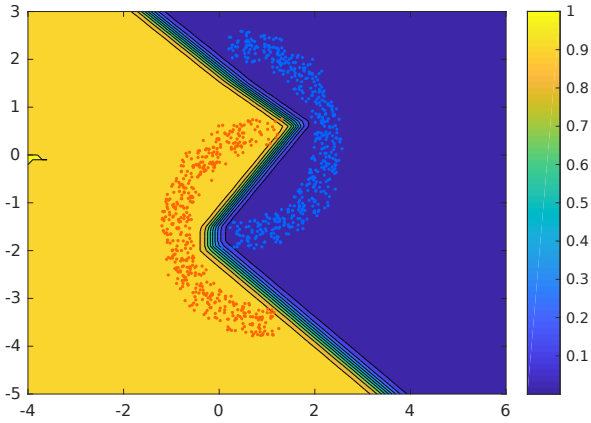
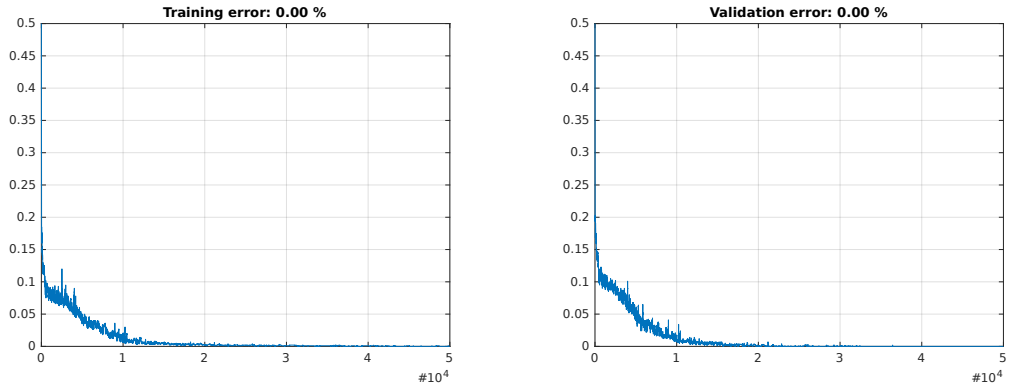


Figure 3: Representation of the training data point and the decision boundary after 50000 iterations (50 times the number of training points)

Percentage	90.00	91.00	92.00	93.00	94.00	95.00	96.00	97.00	98.00	99.00	100.00
Iteration	6070	6385	7301	8251	8554	8816	9500	9978	12194	14939	39594

Table 1: Number of iterations required to reach a given accuracy on validation data

The final training and validation error are respectively 10^{-3} and 0. Graphically, looking at the validation error and training error, we could say that the network converges after $15000 - 20000$ iterations.



(a) Training error (b) Validation error

Figure 4: Representation of the training and validation error as the number of iteration increase

The figure5 represents the sum of the gradient of the training points with respect to each parameter ($W_i^{(j,k)}$, $W_o^{(k)}$, $b_o^{(k)}$ and b_i). We see that the total gradient is still decreasing after 20 iteration, which means that it has not fully converged, despite that the training and validation errors are both ≈ 0

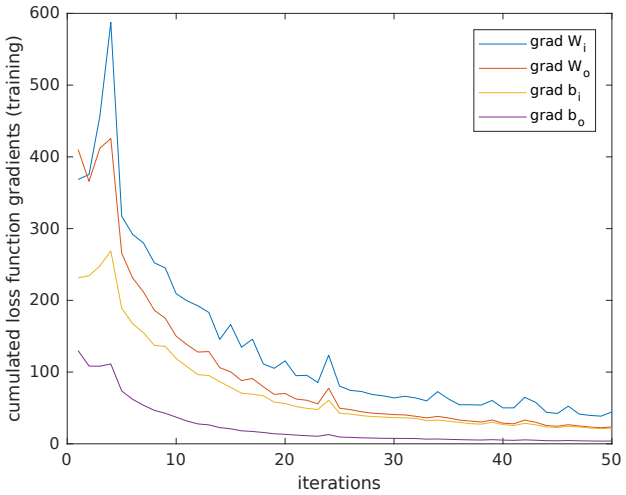


Figure 5: Representation of the sum of the gradient for each of the training points with the number of iterations

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.

The procedure is repeated 5 times with 5 different random initializations. The table 2 presents the results obtained for training and validation errors.

Training error	7.90	0.00	6.70	0.00	0.00
Validation error	7.90	0.00	8.20	0.00	0.00

Table 2: Training and validation error obtained after 50000 iterations for 5 run of the neural network training with random initializations

We see that the network does not always converge. figure 6 shows the boundary with a different random initialization. We see that the network does not converge. figure 7 shows the training and validation curves for this case.

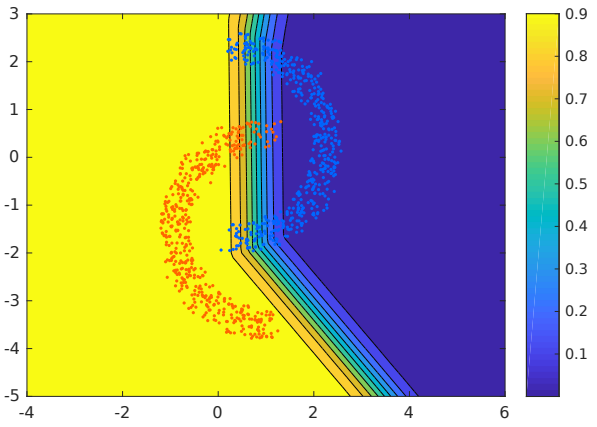
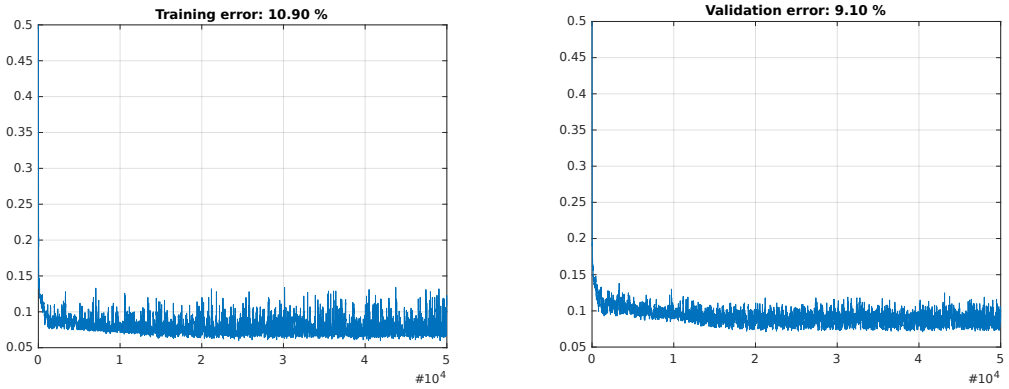


Figure 6: Representation of the training data point and the decision boundary after 50000 iterations in a case where the network does not converge (only initialization has changed)



(a) Training error

(b) Validation error

Figure 7: Representation of the training and validation error as the number of iteration increase

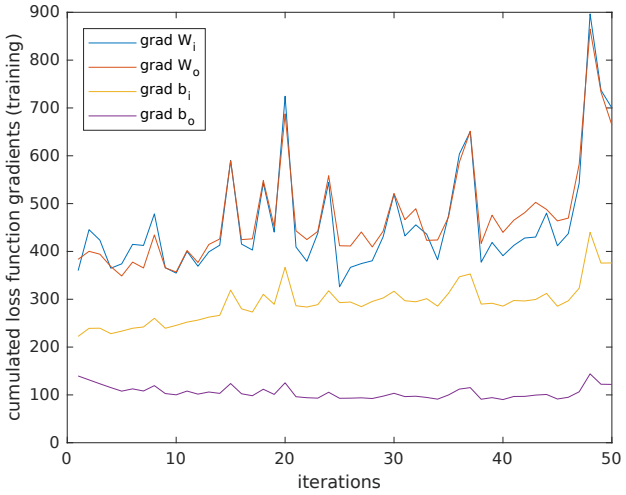


Figure 8: Representation of the sum of the gradient for each of the training points with the number of iterations when the network does not converge

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.

For this question, we consider that the network has converged if there exist an iteration k such that for all iterations after k, the validation error is below 0.5%. For each run, the final training and validation errors as well as a such k (if it exist) are reported in the table 3.

We see that taking a gradient step with a learning rate of 2 often leads to an increase of the loss function, which explains why it doesn't converge, hence this value of the learning rate is too large. At the opposite, taking a learning rate of 0.002 for this particular problem causes very slow convergence. Taking a learning rate of 0.2 seems to give good results for each case, the training and validation errors are low and the convergence is fast when compared to other training rates.

Learning rate	$l = 0.002$			$l = 0.02$		
Train. error	00.30%	00.30%	00.30%	0.00%	0.00%	06.70%
Valid. error	00.10%	00.10%	00.10%	0.00%	0.00%	08.50
Convergence	117 484	111 348	123 603	19 837	14 685	-

Learning rate	$l = 0.2$			$l = 2$		
Train. error	0.00%	0.00%	0.00%	35.70%	28.90%	34.30%
Valid. error	0.00%	0.00%	0.00%	38.00%	30.32%	36.30%
Convergence	5 947	5 295	7 277	-	-	-

Table 3: Representation of the training error, validation error and iteration such that next iterations have a validation error below 0.5%. (- means that there is no such iteration) (This is obtained after 150000 iterations with 7 hidden neurons)

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've run the training procedure for different number of hidden units ($h = \{1, 2, 5, 7, 10, 100\}$) (3 times for each possible value). The results are reported in the table 4 (with the same definition of convergence as before). We see that the more hidden unit we have, the more likely it is for the neural network to converge. In total we have $hd + h + h + 1$ parameters, when h is small, this is not enough to describe the complex structure of the data. We observe that the decision boundary is the intersection of semi-plane. And the number of semi-plane increases with h, such that, for 100 hidden units, the neural network fit the shape of the double moon very well, and the decision boundary seems to be a smooth curve (see 9).

Learning rate	$h = 1$			$h = 2$			$h = 5$		
Train. error	8.50%	8.70%	8.20%	8.80%	8.90%	9.20%	10.90%	7.50%	0.00%
Valid. error	10.10%	9.90%	10.30%	11.20%	11.30%	11.50%	8.90%	09.60%	00.00%
Convergence	-	-	-	-	-	-	-	-	19077

Learning rate	$h = 7$			$h = 10$			$h = 100$		
Train. error	0.00%	6.50%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
Valid. error	0.00%	7.60%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
Convergence	22866	-	16738	12918	16123	17614	10724	18905	13467

Table 4: Representation of the training error, validation error and convergence iteration (0.5% accuracy on validation data) with different number of hidden neurons. (This is obtained after 50000 iterations with a learning rate of 0.02)

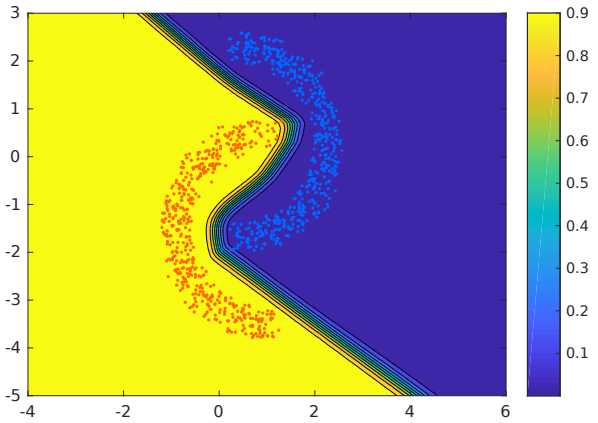


Figure 9: Representation of the decision boundary after 50000 iterations for $h = 100$ hidden units

II CNN building blocks

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) This is a discrete laplacian filter, by Taylor expansion we have

$$f(x+\xi,y)+f(x-\xi,y)\approx 2f(x,y)+\frac{\partial^2 f}{\partial x^2}(x,y)\xi$$

and

$$f(x,y+\xi)+f(x,y-\xi)\approx 2f(x,y)+\frac{\partial^2 f}{\partial y^2}(x,y)\xi$$

And summing gives

$$f(x+\xi,y)+f(x-\xi,y)+f(x,y+\xi)+f(x,y-\xi)-4f(x,y)\approx 2\xi\left(\frac{\partial^2 f}{\partial x^2}(x,y)+\frac{\partial^2 f}{\partial y^2}(x,y)\right)$$

(ii) In this case since the filters are the same along the depth axis, the image RGB channels are convolved independantly by the 3×3 filter

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

and then added together.

(iii) This filter detects edges.

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

If every operation was linear, there would be no point in having multiple layers, since multiple layers could be merged together. In the formula,

$$f(x)=f_L(\dots f_2(f_1(x;w_1);w_2)\dots),w_L).$$

Considering the image as a $W\times H$ vector, we could write $f_i(x)=w_i x$ for a certain matrix w_i, hence we would have

$$f(x)=w_L\dots w_1 x=wx$$

for a certain matrix w, and all the operations would be equivalent to applying a single linear function on the image.

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

The max pooling effect is represented in figure 10. The first thing to notice is that on locally constant zones, the max pooling function has no effect. Reasoning in grey scale images, when there is a change between 2 uniform zones, there will be one that is lighter than the other, and this lighter zone will be expanded, because every point in the dark zone that account for a point in the light zone in its max pooling calculation will become white. Since the points q that account for a point p in their max pooling are the points in a window of size 15×15 around p, the result would be as if we put a 15×15 window around each lighter point and set it to this lighter color.

(a) Original image (b) Result of a max pooling function of size 15

Figure 10: Effect of the max pooling function on an image

This also explains why we can see square shapes around specularity points. These specularity points are almost white hence maximum for each channel, which means that every point accounting for a specularity point in its max pooling calculation will select it as the maximum for each channel and hence be white after the max pooling. The figure 11 represents the difference between the original image and the result after max pooling. We can see the specularity points (they are black point in the middle of a colored square) and that the shape of the veggies are expanded as explained.

Note that here the max pooling function is applied on each channel, so it can create colors that are not in the original image (because it can take different pixels when taking the maximum values of each RGB component on a window around each pixel).

Figure 11: Difference between the result of the max pooling and the original image

III Back-propagation and derivatives

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

Because the loss function f is a scalar, the derivative of f with respect to a set of parameters w_l is defined by the derivative of f for each of the parameter. Hence the size of the derivative is the same as the size of the parameters.

IV Learning a character CNN

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?

This convolutional neural network has 8 layers. The figure 12 resumes the operation done, the size of the filters and the feature map dimensions at each step. (I've generated this graph by slightly adapting a small python script found here `https://github.com/gwding/draw_convnet`). The last convolution filter operates on the whole data and can be viewed as a fully connected layer.

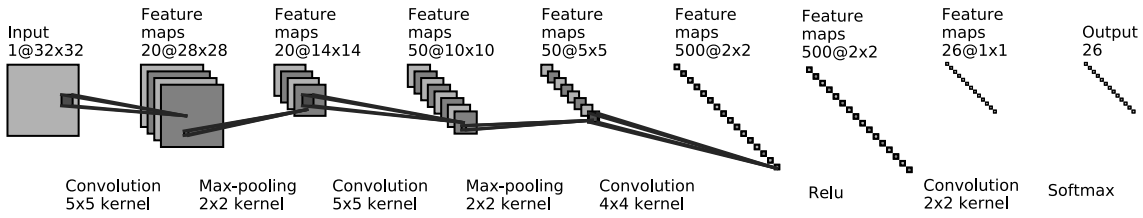


Figure 12: Graphic Representation of the layers of the character convolutional neural network

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 minimising the log-loss. Which neuron's output should become larger?

(i) The softmax operator is effectively turning a vector x in $\mathbb{R}^{26}$ in a vector y in $]0;1[^{26}$ such that

$$\sum_{k=1}^{26} y_k = \sum_{k=1}^{26} \frac{\exp x_k}{\sum_{k=1}^{26} \exp x_k} = 1$$

It can then be interpreted as a probability :

$$p(\text{Input image represents character k}) = y_k$$

(ii) In our case, if the image is of ground truth class k, which means it represents the k^{th} character in the alphabet, The log-loss function will try to minimize $y \to -\log(y_k)$. this means making y_k closer to 1 and also, given the formulas above, making y_i for $i \neq k$ closer to 0. With the previous interpretation of the probability, if we would predict the character of this training image, we would have

$$p(\text{Training image represents character i}) \approx \begin{cases} 1 & \text{if } i = k \\ 0 & \text{if } i \neq k \end{cases}$$

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?

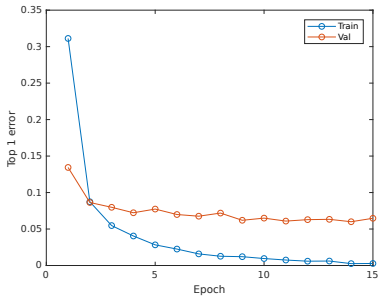
(i) Energy represents the sum of the loss function (here what we see on the objective curve is probably the mean). The prediction errors corresponds to the number of case when the most probable class returned by the convolutional neural network does not match the ground-truth class for this image

(ii) In this dataset, there is 20098 images selected for training and 9100 images selected for evaluation. All the images have a ground-truth label. Since training the convolutional neural network means reducing the log loss function for these specific images, (and by the same process, we increase the component of the output vector matching the ground-truth label of the image as explained above, hence reducing the $top1err$) we expect the training images to have a lower error than the evaluation images (val). (Note that this applies if we measure the error on the train data and on the validation data at the end of each epoch, which is not what is reported in the curves (we measure the training error while training, and the validation after the training. This explains the validation curve appears to be below for the first epoch).

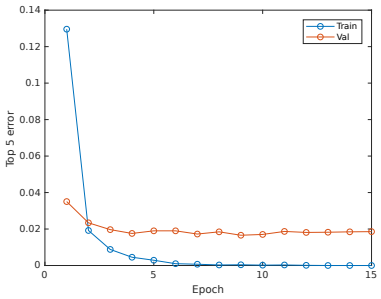
(iii) By looking at the code, we see that the $top1err$ is the number of time that the larger component of the output of the convolutional neural network does not match the ground-truth class of the image, and the $top5err$ is the number of time that the ground-truth class of the image is not one of the 5 larger component of the output vector.

This can be found in `error_multiclass` function, and more specifically, here's a simplified version of the code that computes these errors :

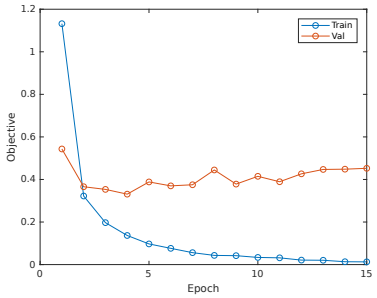
```
[~,predictions] = sort(output_vector, 'descend') ;

error = ~bsxfun(@eq, predictions, labels) ;
top1err = sum(error(1,:)) ;
top5err = sum(min(error(1:m,:),[],1));
```



(a) Top 1 error



(b) Top 5 error



(c) Objective

Figure 13: Representation of the objective, top 1 error and top 5 error with respect to the number of epoch (Done on 15 epoch with batches of 100 images in the stochastic gradient descent

Question 4.4: what can you say about the filters?

The figure 14 shows the 20 filters of the first layer after the training. We can see that they react to different 5×5 patches (this is a good thing, we want them to be well spread accross the space of 5×5 patches in some sense)

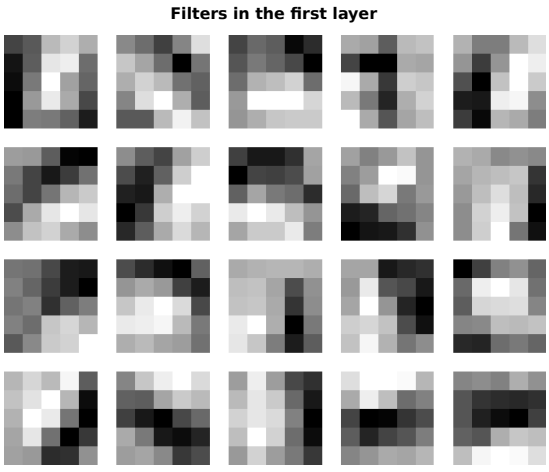


Figure 14: Representation of the 20 filters of the first convolution layer of the CNN

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

From the definition of the convolutional neural network, we are only doing convolutions, and max pooling, which are operations that can be done on tensors, and a relu which is a componentwise operation (the *softmaxloss* layer has been removed for this part). We can get a relation between the number of output coordinates N and the width W of the feature map at each step. These relations are given in the figure 15 (W is expressed in function of N instead of the reverse to avoid floor functions in the expressions, but they can be derived easily).

We can verify it in practice, here the size of the output is $1 \times 137 \times 26$, and the size of the image is 576 and we have $4 \cdot 137 + 28 = 576$.

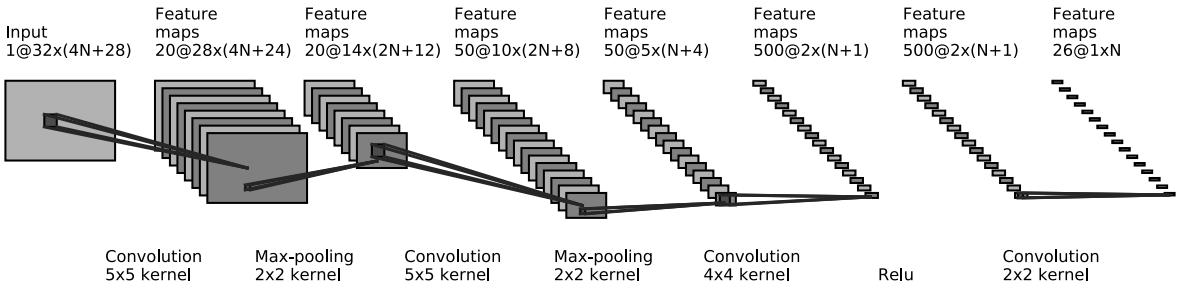


Figure 15: Graphic Representation of the layers of the character convolutional neural network on an image of size $32 \times (4N + 28)$

Taking an image of size $32 \times W$ in fact corresponds to taking regularly images from the input (the stride between each image is 4 and is due to the max pooling (2 max pooling of size 2×2)). If we have done only convolutions (which corresponds to one big convolution of size 32×32, we would have obtained an output with 544 components).

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)?

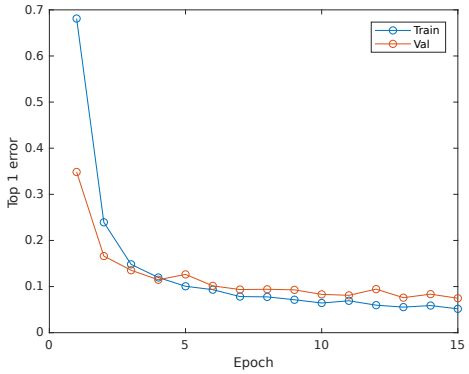
When testing against the 9100 images of the validation set, the average top 1 error was below 10%. On this sentence, the error is much higher, the CNN fails to recognize most of the letters. This is because the CNN outputs 137 vectors (of dimension 26), and one such vector corresponds to a 32×32 image. However, the character to recognize is not necessarily centered in this image, and there can also be multiple characters appearing at the border, and our network has not been trained to recognize characters with such perturbation.

Some of the characters are still good (they corresponds to cases where the character was centered in the image (see 16))

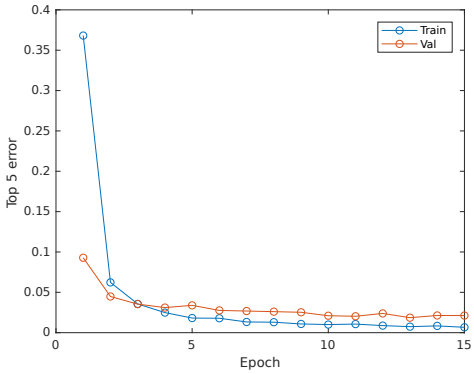
Figure 16: Input image, and characters (in blue) predicted by the CNN

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 recognise the characters in the sentence by repeating the previous part. Does it work better?

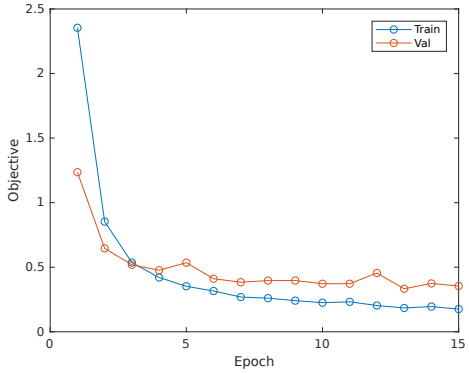
(i) The training and validation error is given in figure 17. The gap between the training error and the validation error for all of the 3 criterion (objective, top 1, and top 5) is much lower than before.



(a) Top 1 error



(b) Top 5 error



(c) Objective

Figure 17: Representation of the objective, top 1 error and top 5 error with respect to the number of epoch when jittering is applied (Done on 15 epoch with batches of 100 images in the stochastic gradient descent

(ii) If we apply this model to the same sentence, the performances are greatly improved, the CNN recognize almost all characters except the l and i which are harder to recognize because they have a small width, and hence are always mixed with other characters on a 32×32 window (see figure 18).

Figure 18: Input image, and characters (in blue) predicted by the CNN trained with jittering

V Using pretrained models

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

(i) The figure 19 represents the image being classified as well as the label corresponding to the highest score, in this case *Bell Pepper*

Figure 19: Image and result of the classification by the CNN

(ii) The top 5 classes as well as their score are represented in table 5. We can notice that the class *Bell Pepper* has a very high score even when compared to the second top class *Cucumber, Cuke*, which means that the model is confident that the image represents a *Bell Pepper*.

	Classe	Score
Top 1 class	Bell Pepper	0.7041
Top 2 class	Cucumber, Cuke	0.1646
Top 3 class	Orange	0.0253
Top 4 class	Lemon	0.0168
Top 5 class	Corn	0.0078

Table 5: Label and score of the top 5 classes for the image pepper

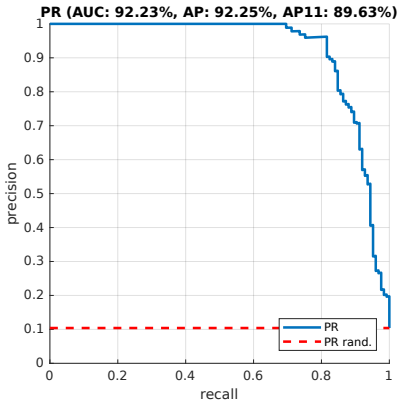
Question 5.2: Report your AP classification results for the three object classes. In the same table, compare your results using CNN features to your results obtained in Assignment 2 for corresponding object classes on the testing image set. What conclusions can you draw?

The table 6 presents the results obtained using the output of the 7^{th} layer of the pretrained CNN as input to a linear and a Hellinger kernel SVM classifier (with $C = 1$). We can see that the CNN based features descriptors outperforms all other descriptors, and gives excellent average precision (more than 90 for all the classes). We can also notice that the top 36 test images returned are all right. Figure 20 shows the average precision for all of the 3 classes.

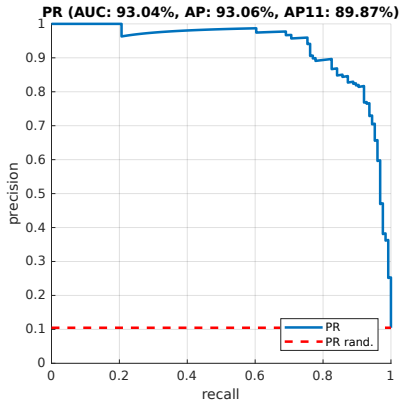
In conclusion, the CNN features descriptors provides a noticeable improvement to the best hand-crafted features descriptors that were previously designed, the main disadvantage being that it requires a lot of labeled images as well as a lot of time to train.

	Linear kernel				Hellinger kernel			
classes	BoVW	VLAD	FV	CNN	BoVW	VLAD	FV	CNN
motorbike	48.66%	68.82%	72.67%	**92.25%**	63.25%	75.42%	81.14%	**92.01%**
aeroplane	54.95%	74.62%	70.64%	**93.06%**	70.72%	75.56%	78.13%	**93.08%**
person	70.64%	75.54%	77.44%	**95.69%**	77.39%	78.86%	82.11%	**95.52%**

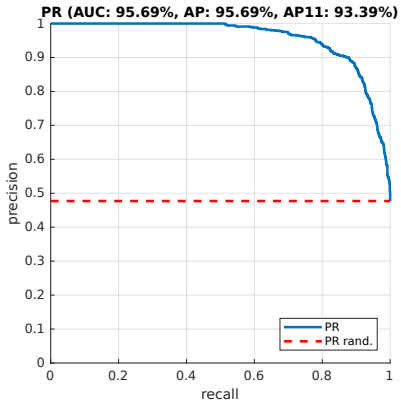
Table 6: Comparison of the features descriptors obtained via a pretrained CNN with features descriptors of assignment 2 for each of the 3 classes (using a SVM classifier for both linear and hellinger kernel)



(a) Motorbike class



(b) Airplane class



(c) Person class

Figure 20: Average precision for each of the 3 classes with CNN based features descriptors and a linear SVM classifier