Assignment 3: Neural networks

Victor Busa

victor.busa@ens-paris-saclay.fr

1 Training a neural network by back-propagation

1.1 Computing gradients of the loss with respect to network parameters

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.

let's note:

$$\begin{aligned} H &= ReLU(W_iX + B_i) \\ Y_0(X) &= W_0H + B_0 \\ s(Y_0(X), Y) &= log[1 + exp(-YY_0(X))] \end{aligned} \tag{3}$$

- (i) We have:

$$\begin{aligned} \frac{\partial s}{\partial Y_0(X)} &= \frac{d}{dY_0(X)}[log(1 + exp(-YY_0(X)))] = \frac{-Y exp(-YY_0(X))}{1 + exp(-YY_0(X))} \\ &= -Y\frac{1}{1 + exp(YY_0(X))} = -Y\sigma(-YY_0(X)) \end{aligned} \tag{E}$$

  Where the first equality comes from the fact that $Y_0(X)$ is a scalar and where we introduce the sigmoid function in the last equality.

- (ii) The calculation of $\frac{\partial Y_0(X)}{\partial W_0}$ gives us:

$$\frac{\partial Y_0(X)}{\partial W_0} = \begin{pmatrix} \frac{dY_0(X)}{dW_{0_1}} \\ \vdots \\ \frac{dY_0(X)}{dW_{0_n}} \end{pmatrix}$$

  and:

$$\frac{dY_0(X)}{dW_{0_i}} = \frac{d}{dW_{0_i}}\left(\sum_{k=1}^{n} W_{0_k}H_k\right) = H_i$$

So finally:

$$\frac{\partial Y_0(X)}{\partial W_0} = H^\intercal \tag{1}$$

the gradient of $Y_0(X)$ w.r.t to B_0 is straightforward as $B_0 \in \mathbb{R}$:

$$\frac{dY_0(X)}{dB_0} = \frac{dB_0(X)}{dB_0} = 1 \tag{2}$$

Using the chain rule with identity (E), (1) and (2) we have:

$$\boxed{\begin{aligned} \frac{\partial s(Y_0(X),Y)}{\partial W_0} &= \frac{\partial s(Y_0(X),Y)}{\partial Y_0}\frac{\partial Y_0(X)}{\partial W_0} = -Y\sigma(-YY_0(X))H^\intercal \\ \frac{\partial s(Y_0(X),Y)}{\partial B_0} &= \frac{\partial s(Y_0(X),Y)}{\partial Y_0}\frac{\partial Y_0(X)}{\partial B_0} = -Y\sigma(-YY_0(X)) \end{aligned}}$$

- (iii) Noting $U \triangleq W_iX + B_i$, let's compute $\frac{\partial Y_0(X)}{\partial W_i}$ using the chain rule:

$$\begin{aligned} \frac{\partial Y_0(X)}{\partial U_l} &= \sum_k \frac{\partial Y_0(X)}{\partial H_k}\frac{\partial H_k}{\partial U_l} \\ &= \sum_k W_{o_k}\frac{\partial}{\partial U_l}max(0,U_k) \\ &= \sum_k W_{o_k}\mathbb{1}_{(U_k>0)}\mathbb{1}_{(k=l)} \\ &= W_{o_l}\mathbb{1}_{(U_l>0)} \end{aligned}$$

As U is a vector, it naturally follows that:

$$\begin{aligned} \frac{\partial Y_0(X)}{\partial U} &= \begin{pmatrix} \frac{\partial Y_0(X)}{\partial U_1} & \frac{\partial Y_0(X)}{\partial U_2} & \dots & \frac{\partial Y_0(X)}{\partial U_K} \end{pmatrix}^\intercal \\ &= W_o^\intercal \circ \mathbb{1}_{(U>0)} \\ &= W_o^\intercal \circ \mathbb{1}_{(W_iX+B_i>0)} \end{aligned} \tag{1'}$$

Moreover, by chain rule we also have:

$$\begin{aligned} \frac{\partial Y_0(X)}{\partial W_i} &= \frac{\partial Y_0(X)}{\partial U}\frac{\partial U}{\partial W_i} \\ &= W_o^\intercal \circ \mathbb{1}_{(W_iX+B_i>0)}\frac{\partial}{\partial W_i}(W_iX+B_i) \\ &= W_o^\intercal \circ \mathbb{1}_{(W_iX+B_i>0)}X^\intercal \end{aligned} \tag{2'}$$

Using the previous calculation and the chain rule, we can compute $\frac{\partial Y_0(X)}{\partial B_i}$ directly:

$$\begin{aligned}\frac{\partial Y_0(X)}{\partial W_i} &= \frac{\partial Y_0(X)}{\partial U}\frac{\partial U}{\partial B_i} \\ &= W_0^\intercal \circ \mathbb{1}_{(U>0)} I = W_0^\intercal \circ \mathbb{1}_{(W_iX+B_i>0)} \qquad (3')\end{aligned}$$

Finally, using the chain rule and the identities (1′), (2′), (3′), we have:

$$\boxed{\begin{aligned}\frac{\partial s(Y_0(X),Y)}{\partial W_i} &= \frac{\partial s(Y_0(X),Y)}{\partial Y_0(X)}\frac{\partial Y_0(X)}{\partial W_i} \\ &= -Y\sigma(-YY_0(X))W_0^\intercal \circ \mathbb{1}_{(W_iX+B_i>0)}X^\intercal \\ \frac{\partial s(Y_0(X),Y)}{\partial B_i} &= \frac{\partial s(Y_0(X),Y)}{\partial Y_0(X)}\frac{\partial Y_0(X)}{\partial B_i} \\ &= -Y\sigma(-YY_0(X))W_0^\intercal \circ \mathbb{1}_{(W_iX+B_i>0)}\end{aligned}}$$

1.2 Numerically verify the gradients

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

Using Taylor expansion we have:

$$\begin{aligned}f(x+h) &= f(x) + hf^{(1)}(x) + \frac{h^2}{2!}f^{(2)}(x) + \mathcal{O}(h^3) \\ f(x-h) &= f(x) - hf^{(1)}(x) + \frac{h^2}{2!}f^{(2)}(x) + \mathcal{O}(h^3)\end{aligned}$$

From what it naturally follows:

$$\frac{f(x+h)-f(x)}{h} = f^{(1)}(x) + \mathcal{O}(h) \qquad (E_1)$$

Yet, we can notice that:

$$\begin{aligned}\frac{f(x+h)-f(x-h)}{2h} &= \frac{1}{2h}(2hf^{(1)}(x) + \mathcal{O}(h^3)) \\ &= f^{(1)}(x) + \mathcal{O}(h^2) \qquad (E_2)\end{aligned}$$

(E_2) is better than (E_1) as the error approximation is only on order of $\mathcal{O}(h^2)$), so finally we will use the approximation:

$$s^{(1)}(\Theta) \approx \frac{s(\Theta+\Delta\Theta) - s(\Theta-\Delta\Theta)}{2\Delta\Theta}$$

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.

using $\Delta\Theta$ = 1e-6, we have:

- For `grad_s_Wo` and `grad_s_Wo_approx` we have:

$$grad\_s\_Wo = \begin{pmatrix}0.047586172091192 & 0.035408612226365 & 0\end{pmatrix}$$
$$grad\_s\_Wo\_approx = \begin{pmatrix}0.047586172065295 & 0.035408612226312 & 0\end{pmatrix}$$

- For `grad_s_Wi` and `grad_s_Wi_approx` we have:

$$grad\_s\_Wi = \begin{pmatrix}-0.018388512294571 & 0.002515533629043\\ -0.013677128631161 & 0.001871020149389\\ 0 & 0\end{pmatrix}$$
$$grad\_s\_Wi\_approx = \begin{pmatrix}-0.018388512314400 & 0.002515533670255\\ -0.013677128602417 & 0.001871020127893\\ 0 & 0\end{pmatrix}$$

- For `grad_s_bo` and `grad_s_bo_approx` we have:

$$grad\_s\_bo = 0.007884628172997$$
$$grad\_s\_bo\_approx = 0.007884628180389$$

- For `grad_s_bi` and `grad_s_bi_approx` we have:

$$grad\_s\_bi = \begin{pmatrix}-0.008378649232738\\ -0.006231926839746\\ 0\end{pmatrix} \quad grad\_s\_bi\_approx = \begin{pmatrix}-0.008378649180713\\ -0.006231926874718\\ 0\end{pmatrix}$$

	Wo	**Wi**	**bi**	**bo**
Absolute Error	2.595021e-11	6.270850e-11	8.699778e-11	7.391931e-12
Relative Error	3.126727e-10	1.955629e-09	5.954439e-09	9.375117e-10

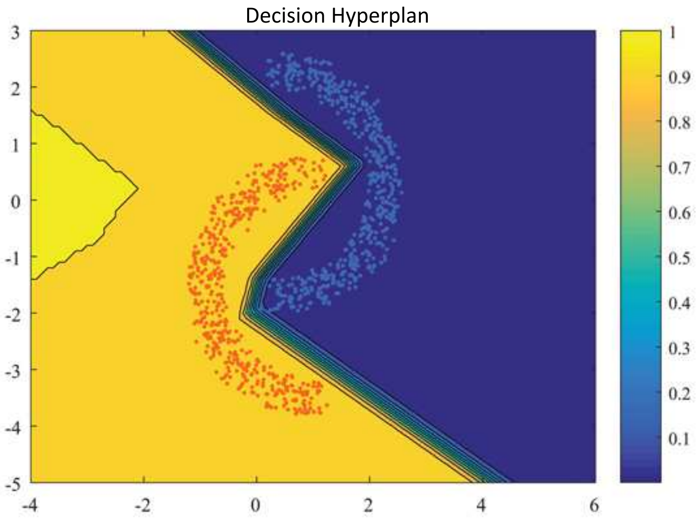
Table 1: Absolute and Relative error for Wi, Wo, bi, bo after 40000 iterations for $h = 3$, $\alpha = 0.02$

The absolute error shouldn't be taken into account. Let's consider that we computed an analytic gradient and a numerical gradient. If the 2 gradients are about 1.0 and the absolute error is 1e-4 than we consider the 2 gradients to match. However, if both gradients were on order of 1e-5 than there difference: 1e-4 is a very big difference and the gradients don't match. Hence, we need to divide by $max(grad\_s\_x\_appox, grad\_s\_x)$ to account for the order of each gradient.

1.3 Training the network using backpropagation and experimenting with different parameters.

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 network converges after roughly 30000 iterations. The final training and validation errors are $0,00\%$ as shown in Figure 1



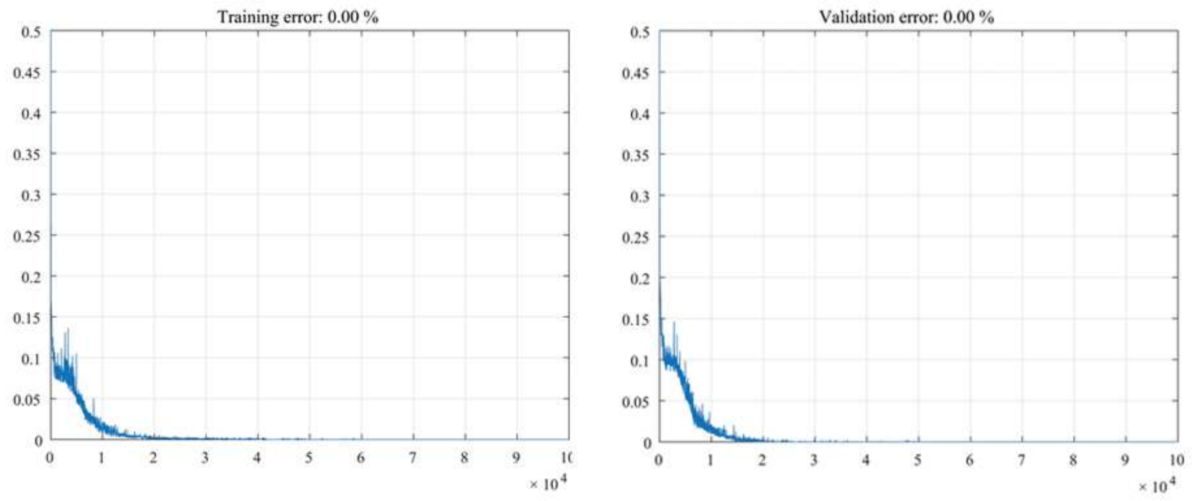
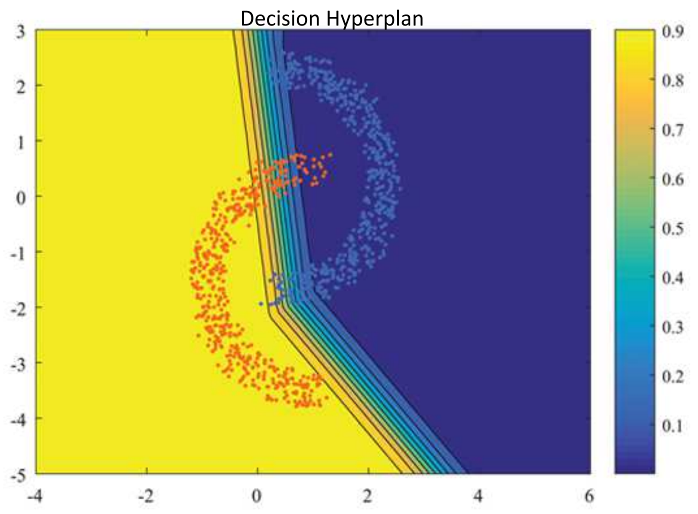
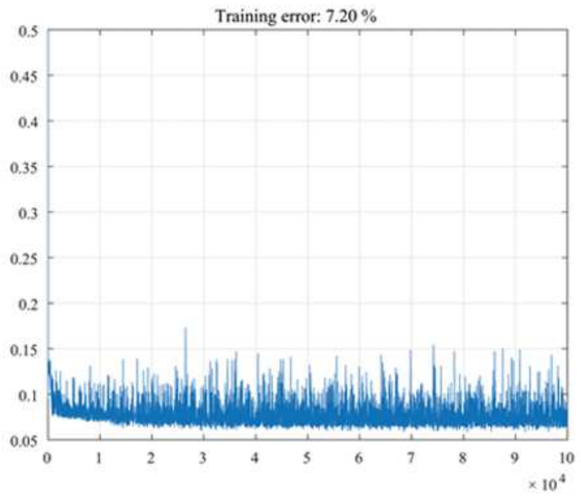


Figure 1: Hyperplan, Training and Validation Error

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.

After 5 random initialization starts, one of them didn't converges to 0.00% training and validation errors. The final training and validation errors for this bad random initialization of the parameters are respectively $7,20\%$ and $9,90\%$ as shown in Figure 2.





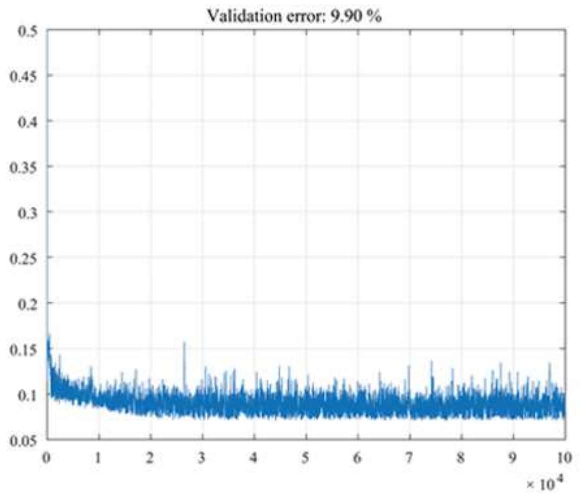


Figure 2: Hyperplan, Training and Validation Error for bad random initialization of the parameters

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). Visualize the decision hyperplane and the evolution of the training error for each value of the learning (here only one of the three runs is sufficient). Briefly discuss the different behaviour of the training for different learning rates.

	learning rate	2	**0.2**	0.02	0.002
	Training error	22.70%	0.00 %	0.00 %	0.50 %
run 1	Validation error	23.10 %	0.00 %	0.00 %	0.60 %
	Convergence (nb of iterations)	No	6k	20k	No
	Training error	31.70 %	0.00 %	0.00 %	0.50 %
run 2	Validation error	33.30 %	0.00 %	0.00 %	0.70 %
	Convergence (nb of iterations)	No	6.5k	28k	No
	Training error	25.90 %	0.00 %	0.00 %	1.70 %
run 3	Validation error	26.40 %	0.00 %	0.00 %	1.60 %
	Convergence (nb of iterations)	No	12k	28k	No

Table 2: Performance of the neural network for 4 different values of the learning rate

On one hand, if we choose a learning rate too small, the neural network will converge but it will take too many iterations. On the other hand, if we choose a learning rate too high, the neural network won't converge. Hence we have to perfectly tune the learning rate hyperparameter in order to achieve the best **Validation error**. One good idea would be to use a **decay learning rate**, i.e. a high learning rate that decays with the number of iterations.

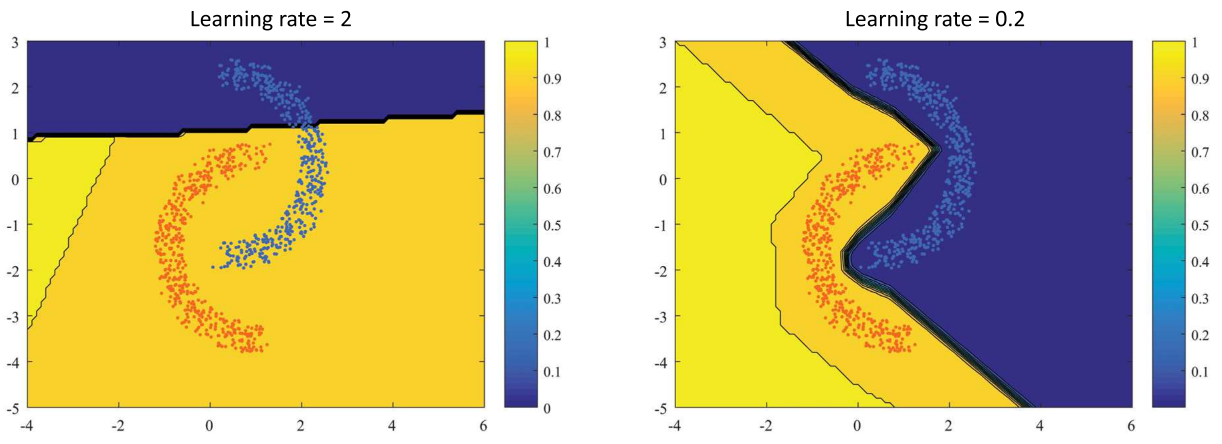


Figure 3: Hyperplans for a learning rate of 2 and 0.2

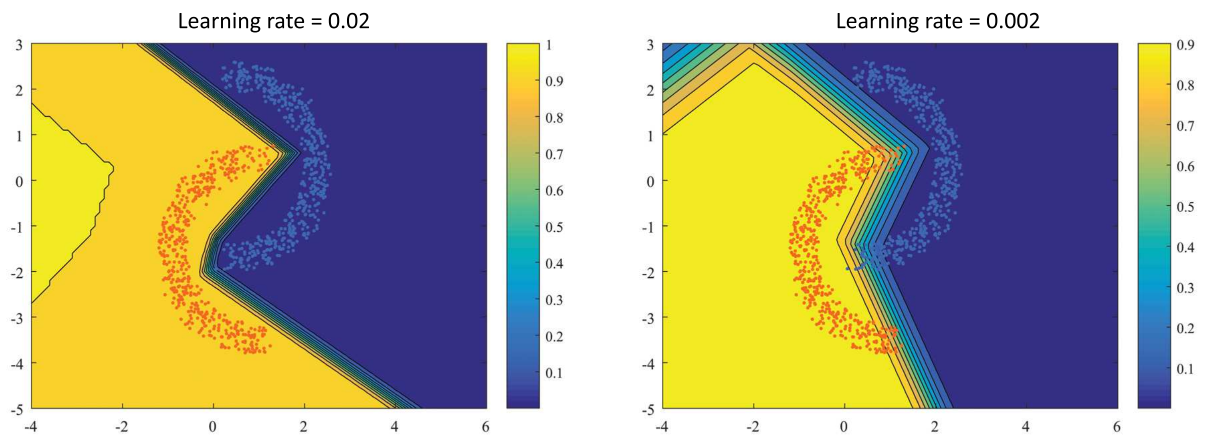


Figure 4: Hyperplans for a learning rate of 0.02 and 0.002

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 behavior. 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). Show the plot of the final decision hyperplane for each value of h (only one of the three plots for each h is sufficient). Discuss the different behaviors for the different numbers of hidden units.

	Number of hidden units	1	2	5	7	10	100
run 1	Training error	8.80 %	8.10 %	7.10 %	0.00 %	0.00 %	0.00 %
	Validation error	11.40 %	8.70 %	8.10 %	0.00 %	0.00 %	0.00 %
	Convergence (nb of iterations)	No	No	No	25k	20k	18k
run 2	Training error	9.10 %	9.10 %	6.80 %	0.00 %	7.20 %	0.00 %
	Validation error	9.90 %	11.10 %	8.90 %	0.00 %	7.90 %	0.00 %
	Convergence (nb of iterations)	No	No	No	20k	No	16k
run 3	Training error	8.70 %	8.50 %	7.30 %	0.00 %	0.00 %	0.00 %
	Validation error	11.30 %	10.30 %	9.40 %	0.00 %	0.00 %	0.00 %
	Convergence (nb of iterations)	No	No	No	25k	25k	19k

Table 3: Performance of the neural network for 7 different values the number of hidden units

We can see that the higher the number of hidden units the more likely the neural network will converge. However adding more hidden units increase the time of training because the gradient will have to flow through each hidden units during the forward and the backward passes. Hence, there is a trade-off between the number of hidden units to consider and the time needed to train the neural network. Finally, depending on the problem, adding more hidden units will actually increase the probability of overfitting, especially if there is not enough data to feed to the neural network.

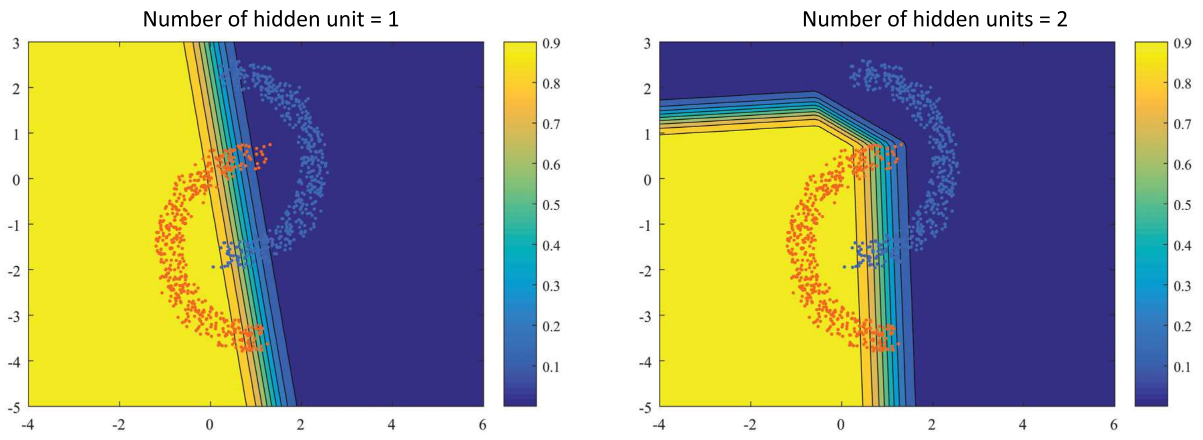


Figure 5: Hyperplans for 1 and 2 hidden units

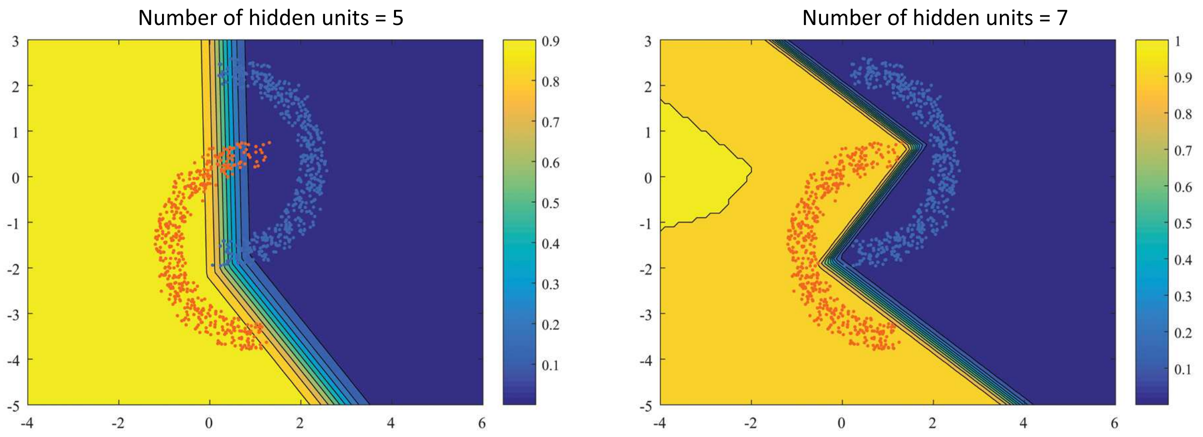


Figure 6: Hyperplans for 5 and 7 hidden units

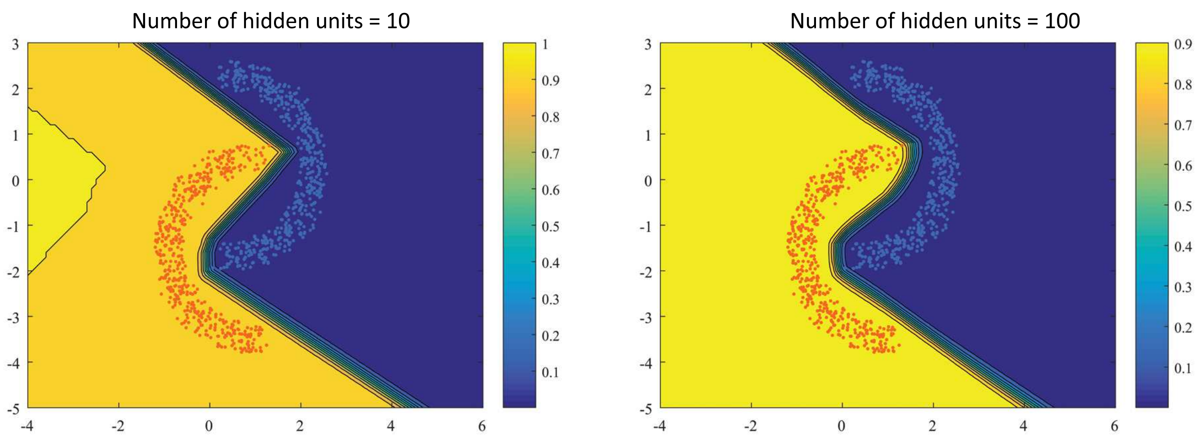


Figure 7: Hyperplans for 10 and 100 hidden units

2 CNN building blocks

2.1 convolution

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?**

A detailed explanation is provided along with a Figure. See Figure 8

- i. The filter implemented is a edge detector.
- ii. each RGB channel are filtered with the same filter
- iii. This filter detects variation of color intensity along cardinal directions. It is an edge detector

Note: A better filter to detect edges is:

$$\begin{pmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

As it also accounts for the difference of color along the diagonals.

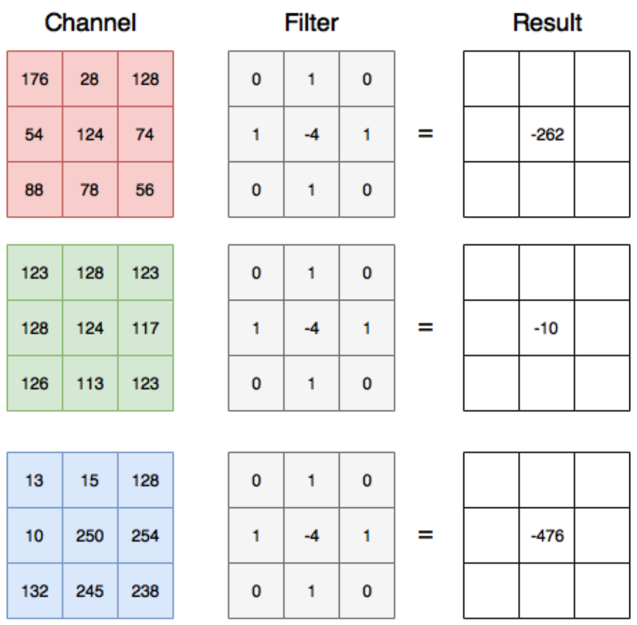


Figure 8: *Up*: Red channel with very different color along oriented axis. *Middle*: Green channel with uniform color in all directions, i.e no sharp edges. *Down*: Blue channel with sharp edges along the up and left direction. After convolution with the filter we see that when there is a very high difference of intensity of color the absolute value of the result is high (presence of edges) while when the color is roughly uniform along all cardinal directions the absolute value of the result is small

2.2 non-linear activation functions

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

The goal of a Neural network is to be able to separate very complex forms. If all the functions in a CNN (and more generally in a Neural Network) were linear then the output of the neural network would be linear and hence the neural network becomes just a linear classifier which we don't want. So neural networks should have non-linear functions to be able to separate very complex forms. In a CNN, ReLU and max-pooling allow to break the linearity of the model.

2.3 pooling

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

Max-pooling is an operation that keeps the maximum value of each RGB channel in a certain neighborhood. Generally in practice it has been shown that lower max-pooling filter is better. Hence, in practice we often use 2×2 max-pooling filters such that the max-pooling operation will keep the brightest pixel among 4 *neighbors* pixels. Hence, the resulting image will look like a smaller blurred version of the input image.

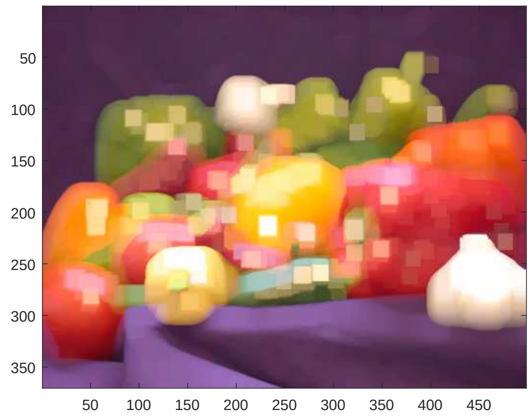


Figure 9: Result of max-pooling

3 back-propagation and derivatives

3.1 the theory of back-propagation

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

The derivative $\frac{\partial f}{\partial w_l}(x_0; w_1, \dots, w_L)$ as the same size as w_l because the output of the network is a **scalar** and we are taking the derivative w.r.t w_l.

4 learning a character CNN

4.1 prepare the data

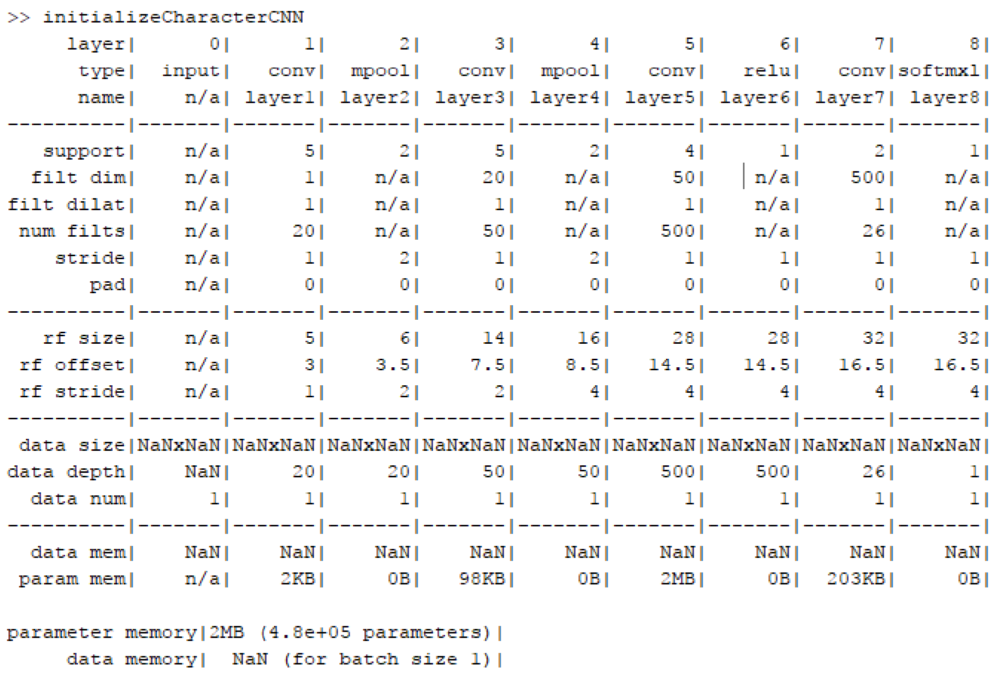
4.2 initialize a CNN architecture

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?

There are 8 layers, and we have 4 filters which are:

- 20 filters of size $5 \times 5 \times 1$
- 50 filters of size $5 \times 5 \times 20$
- 500 filters of size $4 \times 4 \times \times 50$
- 26 filters of size $2 \times 2 \times 500$

A most detailed structure of the CNN is provided in Figure 10



```
>> initializeCharacterCNN
     layer|      0|      1|      2|      3|      4|      5|      6|      7|      8|
      type|  input|   conv|  mpool|   conv|  mpool|   conv|   relu|   conv|softmxl|
      name|    n/a| layer1| layer2| layer3| layer4| layer5| layer6| layer7| layer8|
----------|-------|-------|-------|-------|-------|-------|-------|-------|-------|
   support|    n/a|      5|      2|      5|      2|      4|      1|      2|      1|
  filt dim|    n/a|      1|    n/a|     20|    n/a|     50|  | n/a|    500|    n/a|
filt dilat|    n/a|      1|    n/a|      1|    n/a|      1|    n/a|      1|    n/a|
 num filts|    n/a|     20|    n/a|     50|    n/a|    500|    n/a|     26|    n/a|
    stride|    n/a|      1|      2|      1|      2|      1|      1|      1|      1|
       pad|    n/a|      0|      0|      0|      0|      0|      0|      0|      0|
----------|-------|-------|-------|-------|-------|-------|-------|-------|-------|
   rf size|    n/a|      5|      6|     14|     16|     28|     28|     32|     32|
 rf offset|    n/a|      3|    3.5|    7.5|    8.5|   14.5|   14.5|   16.5|   16.5|
 rf stride|    n/a|      1|      2|      2|      4|      4|      4|      4|      4|
----------|-------|-------|-------|-------|-------|-------|-------|-------|-------|
 data size|NaNxNaN|NaNxNaN|NaNxNaN|NaNxNaN|NaNxNaN|NaNxNaN|NaNxNaN|NaNxNaN|NaNxNaN|
data depth|    NaN|     20|     20|     50|     50|    500|    500|     26|      1|
  data num|      1|      1|      1|      1|      1|      1|      1|      1|      1|
----------|-------|-------|-------|-------|-------|-------|-------|-------|-------|
  data mem|    NaN|    NaN|    NaN|    NaN|    NaN|    NaN|    NaN|    NaN|    NaN|
 param mem|    n/a|    2KB|     0B|   98KB|     0B|    2MB|     0B|  203KB|     0B|

parameter memory|2MB (4.8e+05 parameters)|
     data memory| NaN (for batch size 1)|
```

Figure 10: Detailed structure of the CNN

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 allows to create a probability distribution vector, i.e a vector whose sum of the rows are 1 and whose value in each row represents the probability of belonging to the class corresponding to the the index of the row.
- ii. Minimizing the log-loss would have for effect to choose the class that has the highest probability and hence it will favor the truth neuron's output to be larger and the others to be lower.

4.3 train and evaluate the CNN

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 labeled "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?**

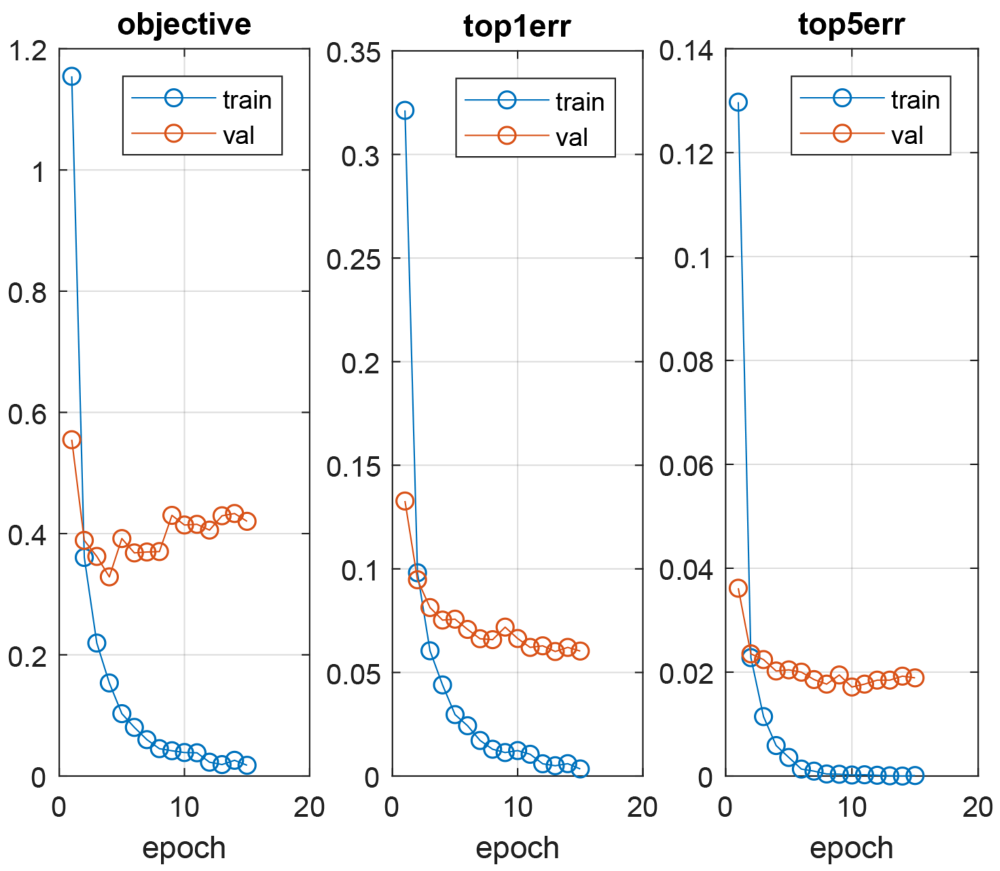


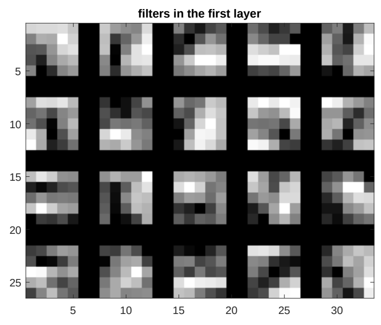
Figure 11: Result of training the CNN

- i. The energy curve represents simply the norm 2 difference between the true classification (here for example $(0, \dots, 1, \dots, 0)$ with a 1 in index i) and the classification outputted by the neural network (for example: $(p_1, \dots, p_i, \dots, p_{26})$ with $\sum_{i=1}^{26} p_i = 1$ and hopefully $p_i > p_j, \forall j \neq i$)
- ii. In practice the validation curve should be above the training curve. It reveals that the model suffers from overfitting **which is natural**. Yet, we want to reduce as much as possible the overfitting as we want the model to performs well on unseen data. Here the model overfits a lot as the validation error is around 0.06 % after 15 epochs and the training error is 0.00 %.
- iii. Top5 error is simply the fraction of test samples where the correct label (class) does not appear in the top 5 predicted results when the results are sorted in decreasing order of confidence. Hence, having a top5 error of 0.00 % means that the ground-truth label associated with the input image is always in the top 5 labels returned by the neural network. Top1 error then represents the error committed by the neural network (predicted label different from the ground-truth label).

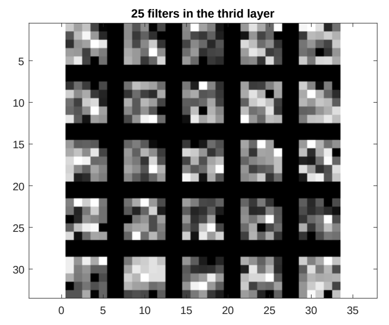
4.4 visualise the learned filters

Question 4.4: what can you say about the filters?

the filters of the first layer represent high level features of the input image such that the filter will be active when the specific feature that encode the filter is in the input image. Figure 12 and 13 display 25 filters for each layer

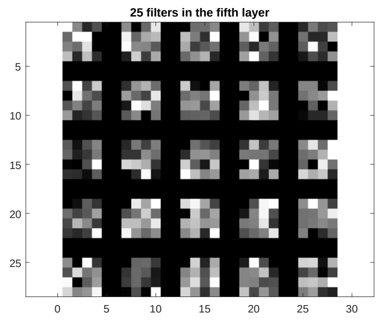


(a) 25 filters from the first layer

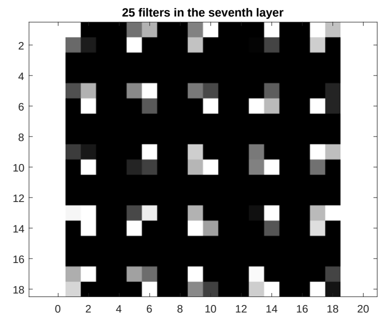


(b) 25 filters from the third layer

Figure 12: 25 filters for the 2 first convolution layers



(a) 25 filters from the fifth layer



(b) 25 filters from the seventh layer

Figure 13: 25 filters for the 2 latest convolution layers

4.5 apply the model

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)`

We can apply to it the CNN learned before because the width of the image is a multiple of 32. In our case the width is 576 = 32 * 18.

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)?

`decodeCharacters` return lot's of incorrect characters. It is due to the fact that the model overfits a lot as we've mentioned in 4.3.ii. As seen in Figure 14 the output of decodeCharacters is:
ntdmhmueesmmrnqamntdhqddmhmkeeouxcqdamntdhqddmhmkeemumdakboyyggjmtcd
fhmnqamssreammdddhlkudddhmkeemmdifhqamdmheesmntdmhmueesmmnmnmhmadmhtt

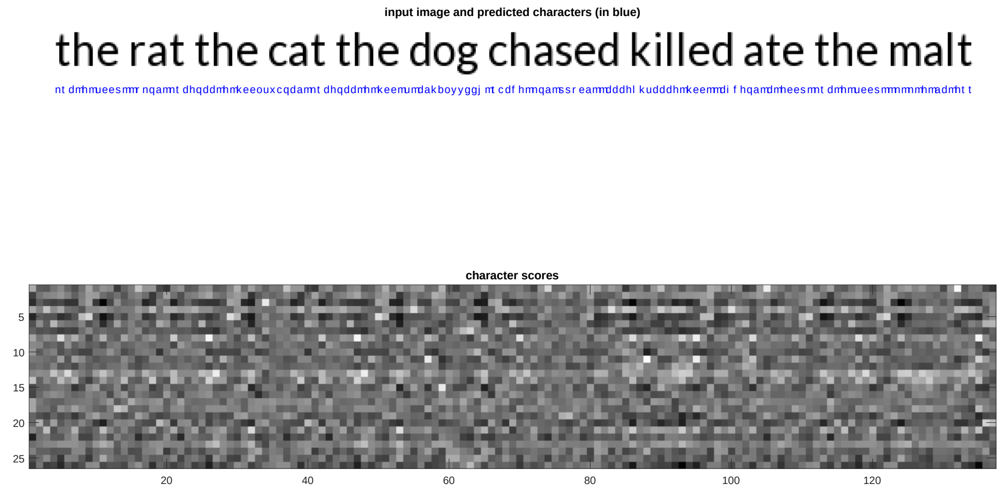


Figure 14: Result of `decodeCharacters` function

4.6 training with jitter

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?**

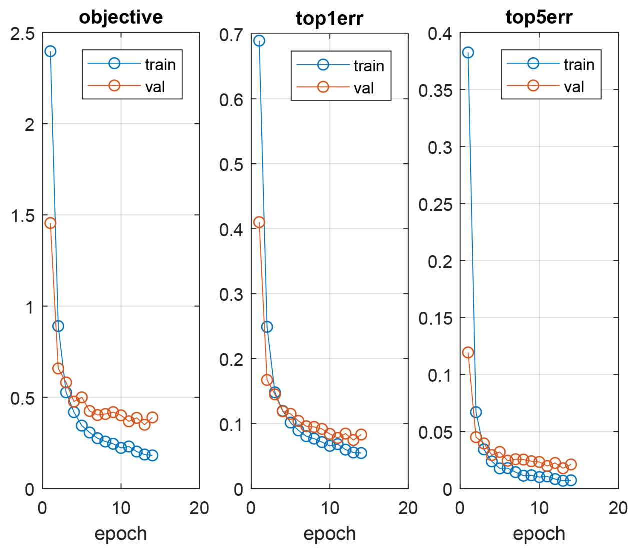


Figure 15: Result of training the CNN with jitter

- i. The gap between the validation and training error is not as big as previously. That indicates that the model doesn't overfit as much as previously

- ii. The new model performs better at recognizing the character of the input image. As seen in Figure 16 ,the output of decodeCharacters is now:
  ttthhheeeeirrraaatttttttthhheeeecccaaattttttthhheeeeedddeoooggggcccc
  hhhaaaassseeedddjkkkkddduheeeedddjeaaattteeeittthhheeeeinmmmmaaaahttt

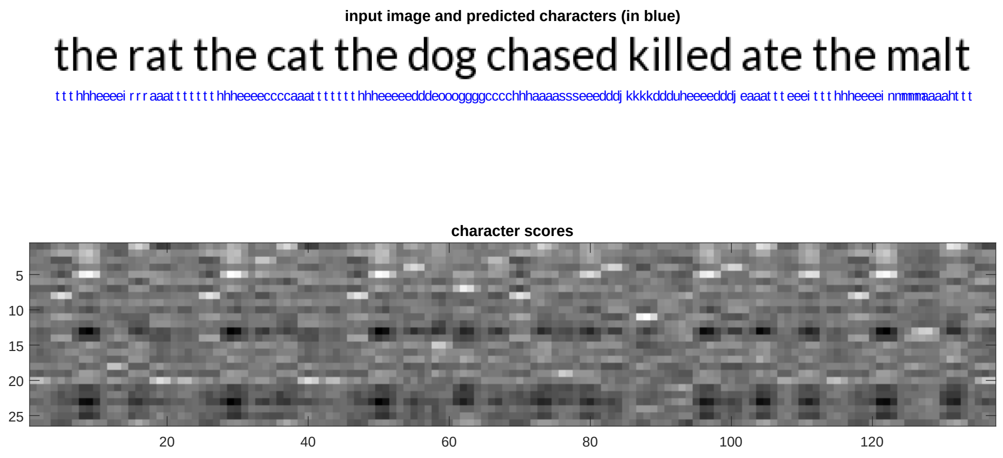


Figure 16: Result of `decodeCharacters` function

5 using pretrained models

5.1 Image classification with a pretrained CNN

Question 5.1:

- **i. Show the classification result.**
- **ii. List the top-5 classes.**

- i. The classification outputs that the image represents a **bell pepper** with probability 0.704. See Figure 17

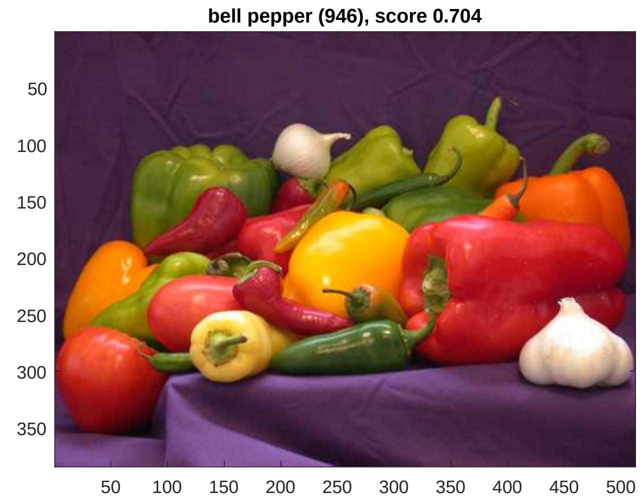


Figure 17: Classification output for pepper image

- ii. the top-5 classes are:

Top-5 classes	1	2	3	4	5
Class	bell pepper	cucumber	orange	lemon	corn
probability	0.704	0.165	0.025	0.017	0.008

Table 4: Top-5 classes outputed by the neural network

5.2 Image classification using CNN features and linear SVM

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?

Using classes images as positives images and background images as negative images we can train a Linear SVM. As I used a random permutation of labels the result of the Linear SVM might vary a bit between two simulations but the order remains the same. See Table 5 below for a comparison of Average Precision results using CNN features and histograms of assignment 2:

classes	Motorbike	Aeroplane	Person
CNN Features with L2 normalization	**92.78 %**	**93.47 %**	**95.60 %**
FV encoding with Hellinger Kernel	81.14 %	78.13 %	82.11 %

Table 5: Average Precision using CNN features and FV encoding

In assignment 2 we have seen that FV encoding using Hellinger kernel achieve the best performances for all three classes. Here we see that using features learned by a well-suited CNN achieve even better performances. This explains why CNN are the most used architecture for object recognition. We don't need to *handcraft* the features anymore and we are able to achieve even better performances.