Introduction to neural networks (2-a and 2-b)

Section 1 – Theoretical foundation

1. What are the train, val and test sets used for ?

The **training data** is used to adjust the parameters of our models in order to minimize the **loss function**.
The **validation set** allows us to **evaluate** and **find the best hyperparameters** for our models, and to perform **early stopping** to avoid **overfitting**.
The **test set** is not supposed to be known during the training of the models. It will be used at the end to **evaluate** our model.

2. What is the influence of the number of examples N ?

The more examples there are, the **more robust** the model will be to new examples. If there are very few examples, the model will **not be able to generalize**.

3. Why is it important to add activation functions between linear transformations ?

A composition of linear functions is a linear function: the model is not enriched. It is then necessary to introduce activation functions between the linear transformations in order **to add non-linearity**.

4. What are the sizes nx, nh, ny in the figure 1? In practice, how are these sizes chosen ?

In the figure 1, n_x is equal to **2**, n_h is **4** and n_y is **2**. In practice, **nx** is the **dimension** of the examples, **nh** is a **hyperparameter** that we define depending on the complexity of the data, and **ny** is the **number of classes**.

5. What do the vectors y^ and y represent? What is the difference between these two quantities ?

The vector y represents the **real labels of the data** and the vector ŷ represents **the model prediction** (the output of the model). The objective of the model is to predict **a vector ŷ as close as possible to the vector y**.

6. Why use a SoftMax function as the output activation function ?

The SoftMax is used as an output activation function for **non-binary classification** : it associates the **probability of belonging** to each class.

7. Write the mathematical equations allowing to perform the forward pass of the neural network, i.e. allowing to successively produce h˜, h, y˜ and yˆ starting at x.

$$\widetilde{H} = W_h X + b_h$$
$$H = \tanh(\widetilde{H})$$
$$\widetilde{Y} = W_y H + b_y$$
$$Y = SoftMax(\widetilde{Y})$$

8. During training, we try to minimize the loss function. For cross entropy and squared error, how must the yˆi vary to decrease the global loss function L ?

For cross entropy and squared error, to decrease the global loss function L, **ŷ should be as close as possible to y**.

9. How are these functions better suited to classification or regression tasks ?

- **MSE :** Better suited to **regression**, it calculates the **distance between the predicted values and the truth values**.
- **Cross entropy :** Better suited to **classification**, it calculates a **probability that each sample belongs to one of the classes**, then it **uses cross-entropy** between these probabilities as its cost function.

10. What seem to be the advantages and disadvantages of the various variants of gradient descent between the classic, mini-batch stochastic and online stochastic versions? Which one seems the most reasonable to use in the general case ?

- **Classic : Fast convergence**, but requires more memory and time than the other two methods and can always fall on the same local optimum.
- **Online stochastic :** This works well when we have many local optimums, but convergence may take some time and there is a lot of noise.
- **Mini-batch stochastic :** It is a compromise that injects enough noise to each gradient update, while achieving a relative speedy convergence.

In general, **mini-batch** seems the most reasonable.

11. What is the influence of the learning rate η on learning ?

The **learning rate is the step size** at each iteration. If the learning rate is too small, convergence will be slow, i.e. the algorithm will need many iterations to converge. But on the other hand, if the learning rate is high, the algorithm may diverge.

12. Compare the complexity (depending on the number of layers in the network) of calculating the gradients of the loss with respect to the parameters, using the naive approach and the backprop algorithm.

- Naive approach : It recalculates for each layer, all the gradients from the end of the network. Thus, **the complexity should be O(n (n-1) (n-2) ...)**.
- Backpropagation : Each layer calculates its gradients based on its inputs and weights, then propagates the error to the previous layer. Thus, **the complexity is proportional to the number of layers O(n).**

13. What criteria must the network architecture meet to allow such an optimization procedure ?

The network architecture **must be differentiable** to perform backpropagation.

14. The function SoftMax and the loss of cross-entropy are often used together and their gradient is very simple. Show that the loss can be simplified by : …

$$\begin{aligned} l(y, SoftMax(\widetilde{y})) &= \sum y_i log(SoftMax(\widetilde{y}))) \\ &= -\sum y_i \left(\frac{log(e^{\widetilde{y}_i})}{\sum_j e^{\widetilde{y}_j}} \right) \\ &= -\sum_i y_i [log(e^{\widetilde{y}_i}) - log(\sum_j e^{\widetilde{y}_j})] \end{aligned}$$

However, y is a one hot vector, so we can write:

$$\begin{aligned} l &= -\sum_i y_i \widetilde{y}_i + log(\sum_j e^{\widetilde{y}_j}) \\ &= -\sum_i y_i \widetilde{y}_i + log(\sum_i e^{\widetilde{y}_i}) \end{aligned}$$

15. Write the gradient of the loss (cross-entropy) relative to the intermediate output y …

$$\frac{\partial l}{\partial \widetilde{y}_i} = -y_i + \frac{e^{\widetilde{y}_i}}{\sum e^{\widetilde{y}_i}} \Longrightarrow \nabla_{\widetilde{y}} l = \begin{bmatrix} \frac{\partial l}{\partial \widetilde{y}_1} \\ \frac{\partial l}{\partial \widetilde{y}_2} \\ \vdots \\ \frac{\partial l}{\partial \widetilde{y}_n} \end{bmatrix} = \begin{bmatrix} \hat{y}_1 - y_1 \\ \hat{y}_2 - y_2 \\ \vdots \\ \hat{y}_n - y_n \end{bmatrix}$$

16. Using the backpropagation, write the gradient of the loss with respect to the weights of the output layer ∇Wy `. Note that writing this gradient uses ∇y˜`. Do the same for ∇by `.

$$\frac{\partial l}{\partial W_{y,ij}} = \sum_k \frac{\partial l}{\partial \widetilde{y}_k} \frac{\partial \widetilde{y}_k}{\partial W_{y,ij}}$$

1) We know that we have : $\frac{\partial l}{\partial \widetilde{y}_k} = \hat{y}_k - y_k$

2) Now let's calculate $\frac{\partial \widetilde{y}_k}{\partial W_{y,ij}}$

- We have $\widetilde{y}_k = (hW_k^T + b_y)_k = \sum_j h_j W_{y,kj} + b_y$
- Then,

$$\frac{\partial \widetilde{y}_k}{\partial W_{y,ij}} = \begin{cases} h_j & \text{if k=j} \\ 0 & \text{else} \end{cases}$$

So,

$$\frac{\partial l}{\partial W_{y,ij}} = \sum_k \frac{\partial l}{\partial \tilde{y}_k} \frac{\partial \tilde{y}_k}{\partial W_{y,ij}} = (\hat{y}_i - y_i) h_j$$

$$\implies \nabla_{W_y} l = \begin{bmatrix} \frac{\partial l}{\partial W_{y,11}} \cdots \frac{\partial l}{\partial W_{y,1n_h}} \\ \vdots \\ \frac{\partial l}{\partial W_{y,n_y 1}} \cdots \frac{\partial l}{\partial W_{y,n_y n_h}} \end{bmatrix} = \begin{bmatrix} (\hat{y}_1 - y_1)h_1 \ldots (\hat{y}_{n_y} - y_{n_y})h_{n_h} \\ \vdots \\ (\hat{y}_1 - y_1)h_{n_h} \ldots (\hat{y}_{n_y} - y_{n_y})h_{n_h} \end{bmatrix} = \nabla_{\tilde{y}} l . h$$

$$\frac{\partial l}{\partial b_{y,i}} = \textstyle\sum_k \frac{\partial l}{\partial \tilde{y}_k} \frac{\partial \tilde{y}_k}{\partial b_{y,i}} = \sum_k \frac{\partial l}{\partial \tilde{y}_k} * 1 = \nabla_{\tilde{y}} l$$

<u>17. Compute other gradients : ∇h˜`, ∇Wh `, ∇bh `.</u>

$$\nabla_{\tilde{h}} l = W_y^T \bigodot (1 - h^2)$$

$$\nabla_{W_h} l = \nabla_{\tilde{h}} x^T$$

$$\frac{\partial l}{\partial b_{h,i}} = \textstyle\sum_k \frac{\partial l}{\partial \tilde{h}_k} \frac{\partial \tilde{y}_k}{\partial b_{h,i}} = \sum_k \frac{\partial l}{\partial \tilde{y}_k} * 1 = \nabla_{b_h} l$$

Convolutional Neural Networks (2-c and 2-d)

Section 1 – Introduction to convolutional networks

1. Considering a single convolution filter of padding *p*, stride *s* and kernel size *k*, for an input of size *x × y × z* what will be the output size ?

The output size will be :

- **Height :** $(\frac{x+2p-k}{s} + 1)$
- **Width :** $(\frac{y+2p-k}{s} + 1)$
- **Channel :** kc_size*z (for each kernel channel, all input channels are added together)

How much weight is there to learn ?

The number of weights to be learned is : **(k * k * z * output_z) + output_z**
So : **(kernel_size * kernel_size * input_channels * output_channels) + output_channels**

How much weight would it have taken to learn if a fully-connected layer were to produce an output of the same size ?

The number of weights that should have been learned is **the same as the output size** of the fully-connected layer.

2. What are the advantages of convolution over fully-connected layers ? What is its main limit ?

The advantages of convolution over fully-connected layers are :

- **Less weights to learn**
- **Local operations** : preserves the locality of the features

Its main limitation is **the lack of global understanding** of the images because they consider only local information.

3. Why do we use spatial pooling ?

Spatial pooling is used to **get a small amount of translational invariance** at each level and to **reduce the number of inputs to the next layer (dimension reduction)**.

4. Suppose we try to compute the output of a classical convolutional network (for example the one in Figure 2) for an input image larger than the initially planned size (224×224 in the example). Can we (without modifying the image) use all or part of the layers of the network on this image ?

Unlike fully-connected layers, convolution layers can be applied **whatever the size of the image**. Fully connected layers must have the **correct input size** to be used, so we could not use them if the input is larger.

5. Show that we can analyze fully-connected layers as particular convolutions.

Each fully-connected layer can be seen as **particular convolutions of size 1x1**.

6. Suppose that we therefore replace fully-connected by their equivalent in convolutions, answer again the question 4. If we can calculate the output, what is its shape and interest ?

If we replace fully-connected by their equivalent in convolutions, we should **be able to use all parts of the network layers** on a larger image. However, the **output shape varies** depending on the size of the input image.

8. For convolutions, we want to keep the same spatial dimensions at the output as at the input. What padding and stride values are needed ?

To keep the same spatial dimensions at the output as at the input, we can **set the stride to 1** and **the padding to : $\frac{k-1}{2}$.**

The kernel size should be an **odd number** to obtain an integer padding.

9. For max poolings, we want to reduce the spatial dimensions by a factor of 2. What padding and stride values are needed ?

To reduce the spatial dimensions by a factor of 2, we can take **a kernel size of 2**, **set the stride to 2** and **the padding to 0**.

10. For each layer, indicate the output size and the number of weights to learn. Comment on this repartition.

Layer	Ouput size	Number of weights
Input	32 x 32 x 3	0
conv1	32 x 32 x 32	5 x 5 x 3 x 32 + 32 = **2 432**
pool1	16 x 16 x 32	0
conv2	16 x 16 x 64	5 x 5 x 32 x 64 + 64 = **51 264**
pool2	8 x 8 x 64	0
conv3	8 x 8 x 64	5 x 5 x 64 x 64 + 64 = **102 464**
pool3	4 x 4 x 64	0
fc4	1000	4 x 4 x 64 x 1000 + 1000 = **1 025 000**
fc5	10	1000 x 10 + 10 = **10 010**

11. What is the total number of weights to learn ? Compare that to the number of examples.

The total number of weights is :

2 432 + 51 264 + 102 464 + 1 025 000 + 10 010 = 1 191 170

The CIFAR-10 database contains 50,000 images in train, which is **very small compared to the number of weights,** so there is **a risk of over-fitting**.

12. Compare the number of parameters to learn with that of the BoW and SVM approach.

The convolutional network has much more weight than a BoW and SVM approach.

14. In the provided code, what is the major difference between the way to calculate loss and accuracy in train and in test (other than the difference in data) ?

- **Train :** The loss is calculated **for each batch and the average is applied at the end**.
- **Test :** The loss is calculated **at the end of each epoch**.

16. What are the effects of the learning rate and of the batch-size ?

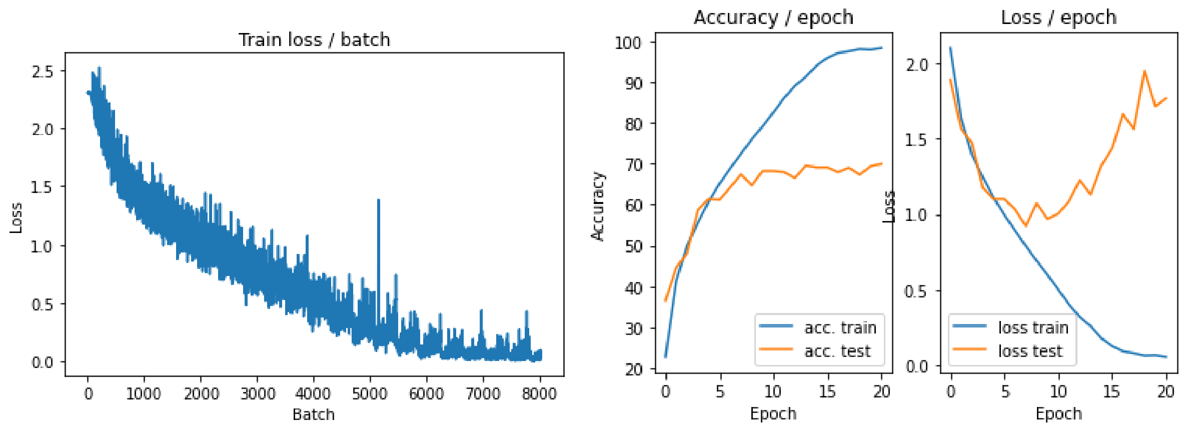
- **Learning rate :** If the learning rate is too small, **convergence will be slow**. On the other hand, if the learning rate is high, **the algorithm may diverge**. In practice, **the learning rate can be decreased with iterations**.

- **Batch-size :** If the batch size is too small, **convergence will be slow due to noise**, but it **costs less memory** and can help to **avoid some local optimums**. If the size is high, **convergence can be fast**, but it **costs a lot of memory** and **can sometimes give the same local optimum**.

17. What is the error at the start of the first epoch, in train and test ? How can you interpret this ?

At the start of the first epoch, the error in *train* (precision : ~0.1 ($\frac{1}{nb\ classes}$)) corresponds to **the error of a random classifier** because **the weights are randomly initialized**. **The error in the train is higher than in the test** because **the test error is calculated at the end of an epoch** and does not take into account the errors at each batch.

18. Interpret the results. What's wrong ? What is this phenomenon ?

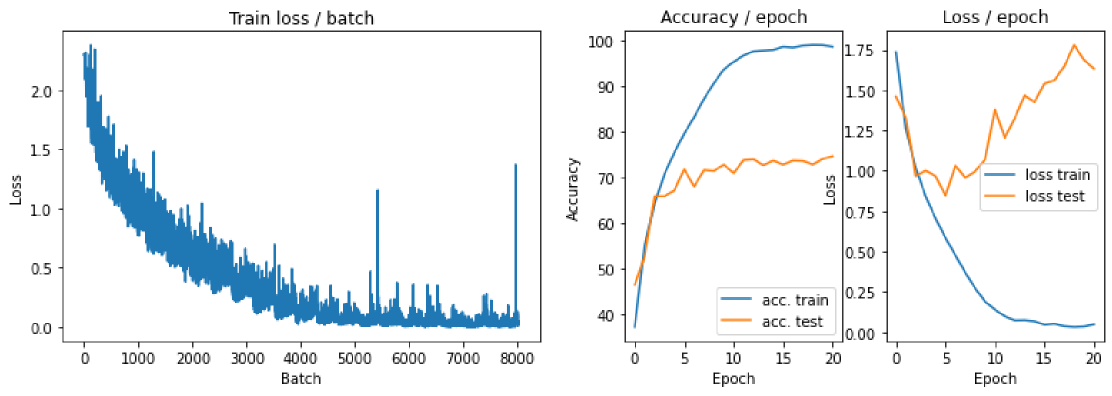


The train loss continuously decreases towards 0 and the accuracy increases towards 100%, so the model becomes more and more efficient on the training data.
The test loss decreases in the first 6 epochs and then starts to increase, while the accuracy in test increases and then stagnates at 69%. This phenomenon is called **over-fitting**, which is when the train loss decreases while the test loss increases.

Section 3 – Results improvements

Section 3.1 - Standardization of examples

19. Describe your experimental results.



We can again observe an **overfitting**, but with the standardization of the examples, **the convergence is faster (from 6 epochs to 5) and the accuracy in test is better than without (from 69% to 74%)**.

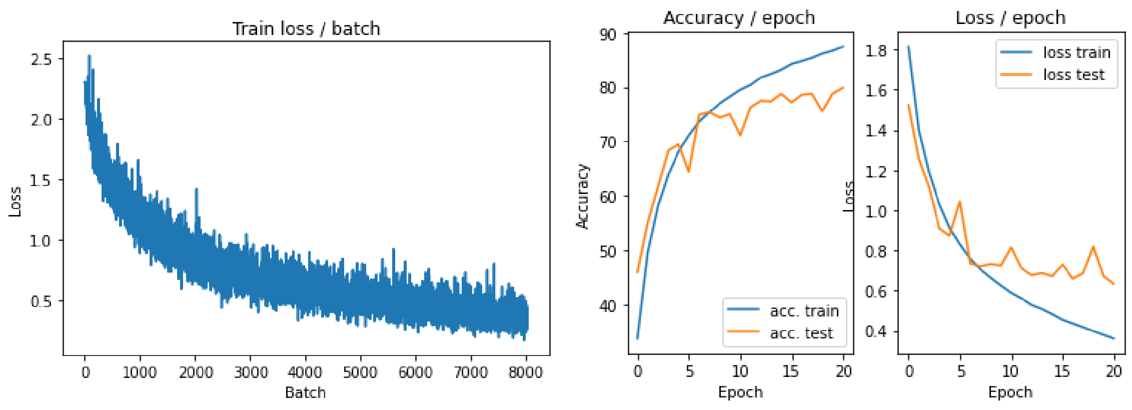
20. Why only calculate the average image on the training examples and normalize the validation examples with the same image ?

We calculate the average image only on the training examples and normalize the validation examples with the same image because, in practice, **we do not have access to the validation / test data** and **it adds bias if we do**.

The standardization is used to **put the inputs in the same scale** and there is an **invariance of the learning rate** from the beginning to the end.

Section 3.2 - Increase in the number of training examples by data increase - Data augmentation

22. Describe your experimental results and compare them to previous results



Data augmentation allows our model to **generalize** and thus **avoid or delay overfitting**. Indeed, unlike the previous results, **there is no overfitting** in the current results and the **accuracy in test is also better (from 74% to 79%)**.

23. Does this horizontal symmetry approach seems usable on all types of images ? In what cases can it be or not be ?

The horizontal symmetry approach is **not usable on all types of images**. For example, **we can use this approach when we want to classify animals**, but **it will not work with a textual data set (*b* can be changed to *d*)**.

24. What limits do you see in this type of data increase by transformation of the dataset ?

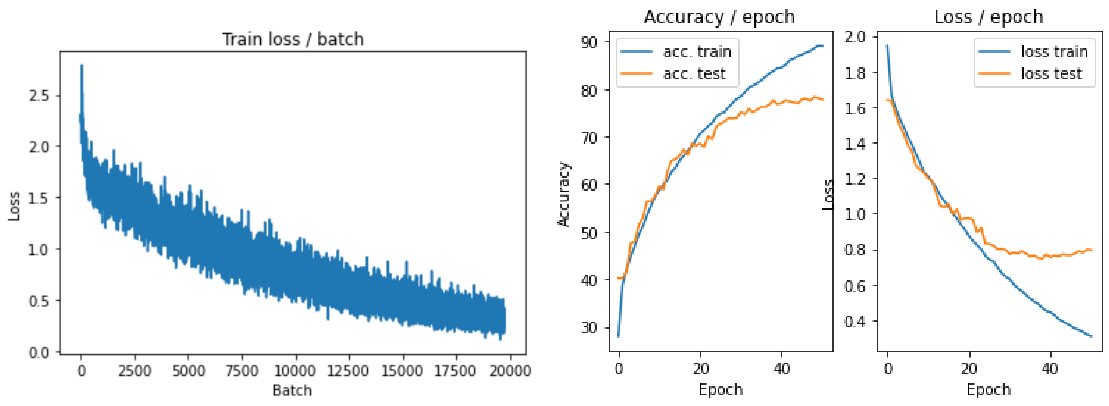
It may introduce **new patterns that are not consistent** and cannot be observed in reality, so we need representative examples in the original data set. Generating a large number of examples **can also take some time**.

25. Bonus : Other data augmentation methods are possible. Find out which ones and test some.

Mixup is another data augmentation method. For example, we can combine an image of a cat and an image of a dog with the **average of the pixels**. This method allows for **greater diversity** in the training data.

Section 3.3 - Variants on the optimization algorithm

26. Describe your experimental results and compare them to previous results, including learning stability.



It seems that the **test accuracy stagnates at 79% as before**, but there is **more stability** on the accuracy and loss in test, **the curves are smoother**. **The convergence is slower than before** (from 16 epochs to 40) but **the overfitting is delayed.**

27. Why do these two methods improve learning ?

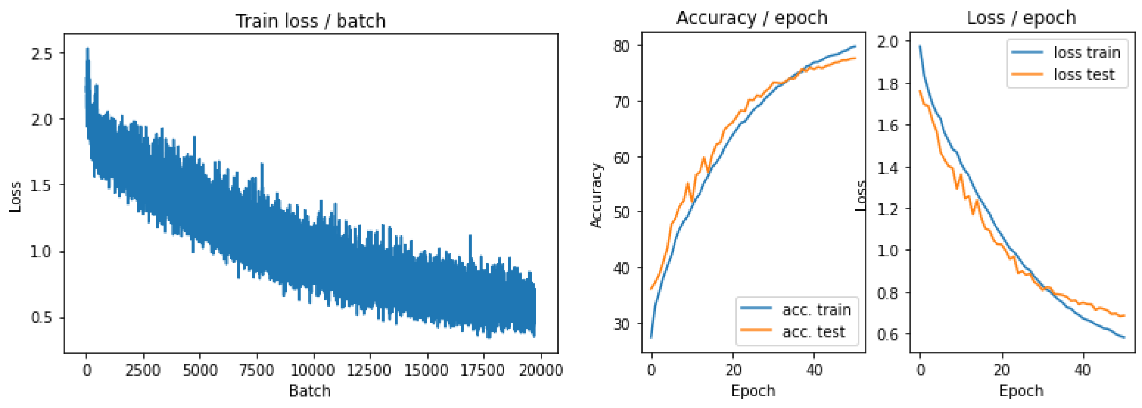
- **SGD with momentum** is a method that **takes into account previous gradients** and accelerates the gradient vectors in the right directions, resulting in **faster convergence and less oscillations**. The momentum will be **used to weight the previous gradient** and then to calculate the new gradient.
- **Learning rate scheduler** is used to **decays the learning rate every epoch**. Thus, **the convergence is faster at the beginning** to get closer to an optimum, and **slower at the end** to be more accurate.

28. Bonus : Many other variants of SGD exist and many learning rate planning strategies exist. Which ones ? Test some of them.

Adam (Adaptive Moment Estimation) is another variant of SGG, it automatically determines a learning rate for each parameter and varies it over the different iterations.

Section 3.4 - Regularization of the network by dropout

29. Describe your experimental results and compare them to previous results.



Unlike precedently, there is **no overfitting in 51 epochs** and **the model can still be improved**. **The test loss is also better** than before (from 0.8 to 0.68).

30. What is regularization in general ?
Regularization is a set of techniques used to **prevent our model from overfitting by adding extra information** to it.

31. Research and "discuss" possible interpretations of the effect of dropout on the behavior of a network using it ?
Sometimes **some weights can be redundant**, so in training, **dropout will be used to randomly disable some weights**. It helps the model to be **more efficient**. For example, if you practice playing tennis with one hand, the day you play with both, **you will be better**.

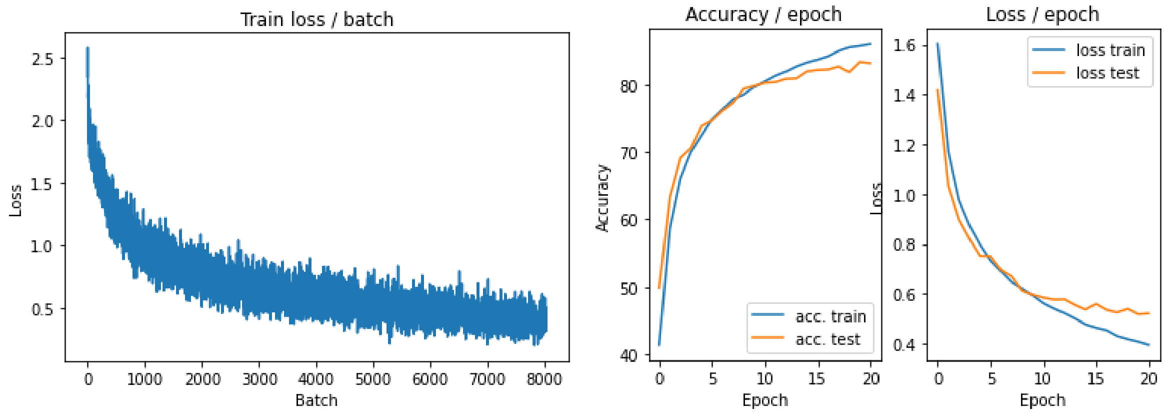
32. What is the influence of the hyperparameter of this layer ?
The dropout layer hyperparameter defines the **probability that a neuron is disabled**. A high value can lead to **slow convergence** and **underfitting** because many neurons are not optimized. Moreover, if the value is too small, **the convergence will be fast** but there is **a risk of overfitting** as without dropout layer.

33. What is the difference in behavior of the dropout layer between training and test ?

- **Train : Probability *p*** to disable a weight.
- **Test :** The weights are not disabled but the output/activation is multiplied by ***p***.

Section 3.5 - Use of batch normalization

34. Describe your experimental results and compare them to previous results.



Unlike precedently, **the accuracy has slightly improved** (from 79% to 83%). Moreover, **the batch normalization accelerates the convergence** (from more than 51 epochs to 21).