Deep Learning in Data Science, DD2424

Short report on assignment 1 - Bonus

Image classification with a one-layer network

Côme Lassarat

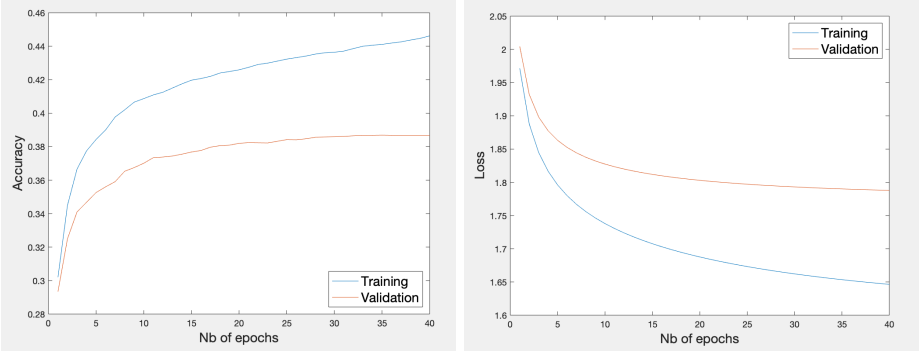
April 4, 2022

1 Introduction

This work aims to implement a one-layer network to classify images from the CIFAR-10 dataset. The network designed will be trained using mini-batch gradient descent and tested in different scenarios by varying the learning rate and integrating L2-regularization. A first report has already been submitted where the gradients analytical expression of the relevant quantities was checked and where the performance of the one-layer network was reported (depending on the values of its hyperparameters). This document intents to improve the performance of the network thanks to different techniques: use almost all the data for training, flip images to augment the training data or implement step decay. Then, the performance of the network will be analyzed when using a new loss function: multiple binary cross-entropy loss.

2 Improve the performance of the network

The goal is to improve the performance of the network. In the best scenario previously tested, the results were the following (Table 1 and Figure 1):



(a) Plot of the accuracy over the epochs (b) Plot of the loss over the epochs

Figur 1

Training accuracy	Validation accuracy	Test accuracy
44.62%	38.67%	39.15%

Tabell 1: Final accuracies

The same initial matrices W and b are used for the following experiments.

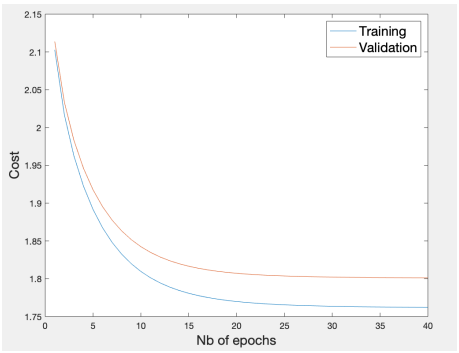
2.0.1 Use all the available training data for training

First, a first technique that can be tried is to use almost all the data available for training and decrease the size of the validation set to approximately 1000 samples. The experiment has been done for the following configuration (the one that worked best in the previous work): $\lambda = 0.1$, $n_{epochs} = 40$, $n_{batch} = 100$, $\eta = 0.001$.

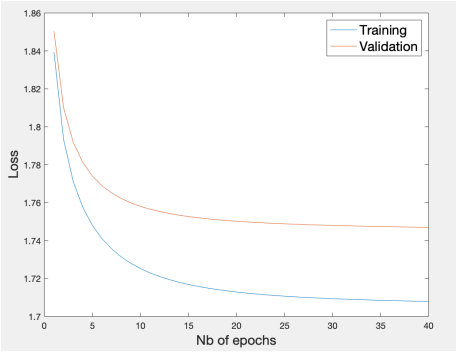
Training accuracy	Validation accuracy	Test accuracy
42.15%	41.56%	40.92%

Tabell 2: Final accuracies

It seems that this strategy is efficient to improve the classification performance of the network (validation accuracy increased from 38.67% to 41.56% and test accuracy increases from 39.15% to 40.92%) The downside of this strategy is the risk to have a validation batch not large enough to have a true estimation of the performance of the network.

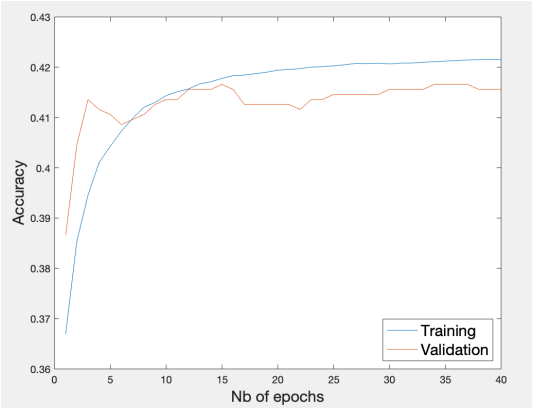


(a) Plot of the cost function according to the number of epochs



(b) Plot of the loss function according to the number of epochs

Figur 2



Figur 3: Evolution of the accuracy over the epochs

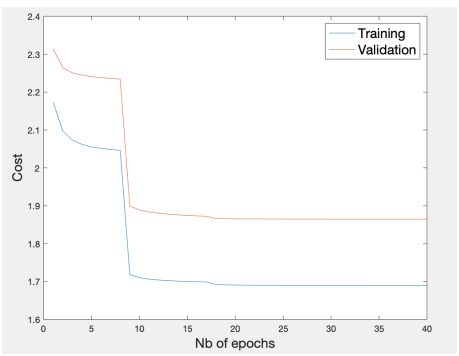
2.0.2 Step decay

Another strategy that can be implemented is step decay: every 9th epochs, the learning rate η is divided by 10. After few tries, a working configuration is the following: $\lambda = 0.1$, $n_{epochs} = 40$, $n_{batch} = 100$, $\eta_{start} = 0.028$. The results are exposed in Figure 4, 5 and Table 3:

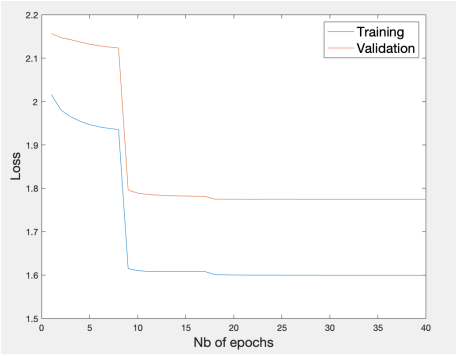
Training accuracy	Validation accuracy	Test accuracy
46.70%	39.23%	39.91%

Tabell 3: Final accuracies

This strategy is also efficient: it improves accuracy and allows a faster convergence.

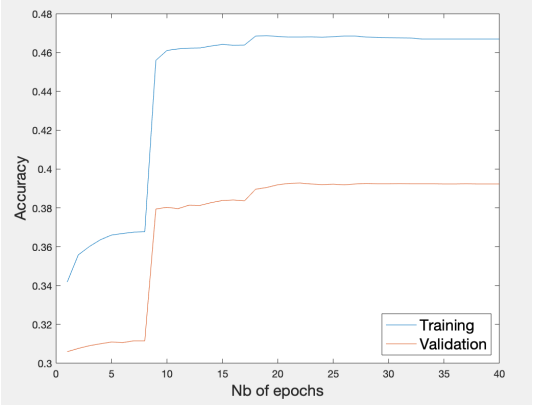


(a) Plot of the cost function according to the number of epochs



(b) Plot of the loss function according to the number of epochs

Figur 4



Figur 5: Evolution of the accuracy over the epochs

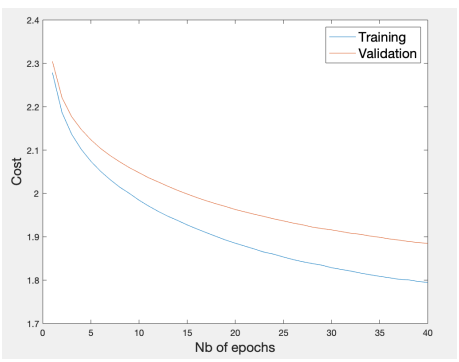
2.0.3 Flipping images horizontally in the training dataset

A final trick is to flip randomly horizontally around 50% of the images of the training dataset at each epoch. The configuration used is $\lambda = 0.1$, $n_{epochs} = 40$, $n_{batch} = 100$, $\eta = 0.001$. The results of such a trick are visible on Figure 6, 7 and Table 4:

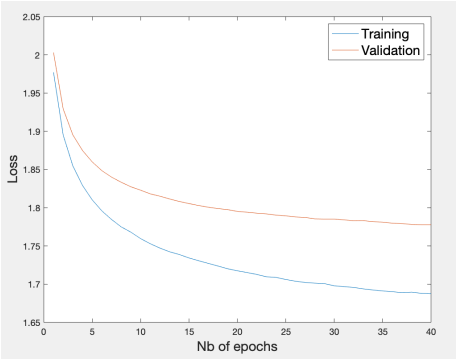
Training accuracy	Validation accuracy	Test accuracy
42.92%	38.93%	39.73%

Tabell 4: Final accuracies

An accuracy improvement is again noticeable.

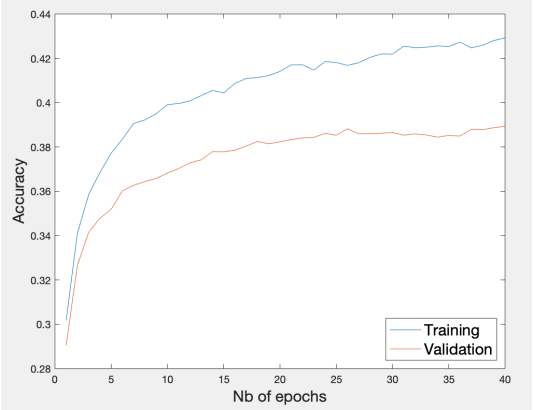


(a) Plot of the cost function according to the number of epochs



(b) Plot of the loss function according to the number of epochs

Figur 6



Figur 7: Evolution of the accuracy over the epochs

3 Train network - multiple binary cross-entropy losses

3.1 Expressions of the new training functions

In the section, the softmax operation when computing the probability vector is replaced by the sigmoid operation:

$$\sigma(s) = \frac{\exp(s)}{\exp(s) + 1}$$

The loss function is now the multiple binary cross-entropy loss:

$$l_{multiplebce} = -\frac{1}{K}\sum_{k=1}^{K}[(1 - y_k)\log(1 - p_k) + y_k \log(p_k)]$$

Let's $\sigma(\mathbf{s}) = \mathbf{p}$ with $\mathbf{s} = W\mathbf{x} + \mathbf{b}$. We know that

$$\frac{\partial \log(\sigma(s))}{\partial s} = 1 - \sigma(s) \quad \text{and} \quad \frac{\partial \log(1 - \sigma(s))}{\partial s} = -\sigma(s)$$

According to the chain rule,

$$\frac{\partial l_{multiplebce}}{\partial s_i} = \frac{\partial l_{multiplebce}}{\partial p_i}\frac{\partial p_i}{\partial s_i} = \frac{\partial l_{multiplebce}}{\partial p_i}\sigma'(s_i)$$

with

$$\frac{\partial l_{multiplebce}}{\partial p_i} = -\frac{1}{K}[-(1 - y_i)\frac{1}{1 - p_i} + y_i\frac{1}{p_i}]$$

Finally,

$$\frac{\partial l_{multiplebce}}{\partial s_i} = \frac{p_i - y_i}{K}$$

which gives

$$\boxed{\frac{\partial l_{multiplebce}}{\partial \mathbf{s}} = \frac{\mathbf{p} - \mathbf{y}}{K}} \tag{1}$$

Thanks to the chain rule, (1) allows to calculate $\frac{\partial l_{multiplebce}}{\partial \mathbf{W}}$ and $\frac{\partial l_{multiplebce}}{\partial \mathbf{b}}$, given $\mathbf{x}$ a sample vector:

$$\boxed{\frac{\partial l_{multiplebce}}{\partial \mathbf{W}} = \frac{\mathbf{p} - \mathbf{y}}{K}\mathbf{x}^T} \tag{2}$$

$$\boxed{\frac{\partial l_{multiplebce}}{\partial \mathbf{W}} = \frac{\mathbf{p} - \mathbf{y}}{K}} \tag{3}$$

These new gradients only differ from the previous ones with a factor $\frac{1}{K}$. The one-layer network is then trained with these new functions (cost and loss functions, gradients, classifier) using gradient decent.

3.1.1 Training and results

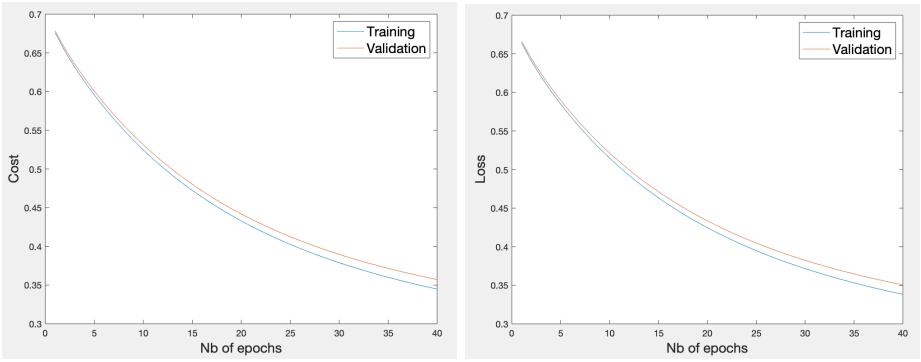
After few tries, the following configuration seems to perform well: $\lambda = 0.004$, $n_{epochs} = 40$, $n_{batch} = 100$, $\eta = 0.013$. The results are exposed on Figure 8, 9).

The final accuracies are the following (Table 5):

When comparing the different accuracies between the new training procedure (using sigmoid) and the previous one (using softmax), it is noticeable that the softmax training accuracy is lower but that its validation and test accuracies are higher: the new training procedure appears to be more likely to overfitt.

X	Training accuracy	Validation accuracy	Test accuracy
Softmax	44.62%	38.67%	39.15%
Sigmoid	46.81%	38.40%	38.72%

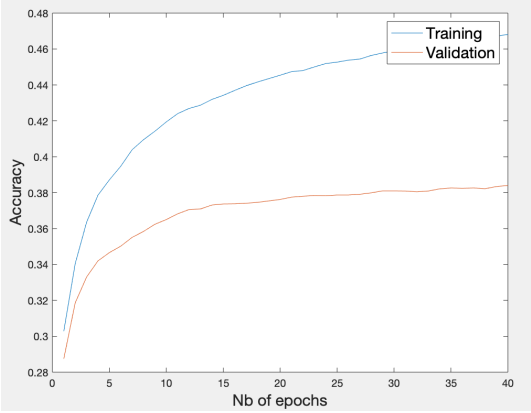
Tabell 5: Final accuracies



(a) Sigmoid training procedure - Plot of the cost function according to the number of epochs

(b) Sigmoid training procedure - Plot of the loss function according to the number of epochs

Figur 8

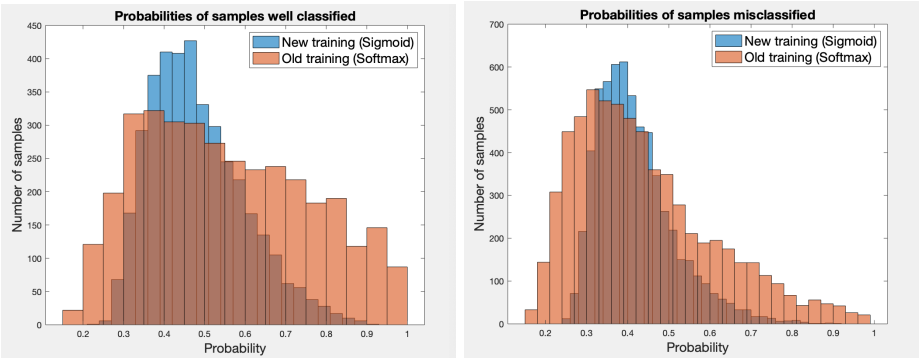


Figur 9: Sigmoid training procedure - Evolution of the accuracy over the epochs

Finally, the histograms of the probabilities for well classified and misclassified samples for both ways of training have been plot (Figure 10):

One can notice that the distributions between the two training procedures are different:

- Well classified samples categorized by Softmax training procedure have more distributed probabilities, unlike the the new training procedure



(a) Histogram of probabilities for wellclassified samples (for Softmax training and Sigmoid training procedures)

(b) Histogram of probabilities for missclassified samples (for Softmax training and Sigmoid training procedures)

Figur 10

- The same observation can be made on missclassified samples, even if the distributions are more similar

4 Conclusion

Given the experiments previously made, the best strategy to improve the performance of the classification process (ie. the test accuracy) is to use step decay, even if the strategy of using almost all the data available for training (the validation set might be too small to have a true estimation of the performance of the network). Of course, combining all the techniques seen is the best idea.