# Deep Learning in Practice
# Practical Session 1: Hyper-parameters and training basics with PyTorch

DARGIER Antoine
CentraleSupélec, MDS - SDI
antoine.dargier@student-cs.fr

DOUILLY Thomas
CentraleSupélec, MDS - SDI
thomas.douilly@student-cs.fr

February 6, 2023

## Problem 1: Architecture, Training process and Performances

In problem 1, after testing all the parameters of the model, the one having the best performance reached had the following architecture:

- A 2D-Convolutionnal Layer with a kernel size set to 3
- A ReLU Activation Layer
- A Flattening Layer
- A Linear Transformation Layer with input size set to 196 and output size set to 10 (number of classes).
- A final Sigmoid activation Layer

Concerning the training process of this model, it has been made on the training set (6,000 images), and the validation part was done by calculating the accuracy on the validation set (1,291 images).

We used the following parameters for updating its parameters : The learning rate has been set to 2, the number of epochs to 50 (it could have been set higher but it really had little impact on the training loss), the size of the batches was set to 10 and the loss function used here was the Cross Entropy Loss combined with the Stochastic Gradient Descent optimizer.

Thanks to this architecture, we finally reached an accuracy of 91.43 % on the testing set, which is significant for this kind of algorithm.

## Problem 2: Architecture, Training process and Performances

In the second problem concerning the prediction of the price of the house, we first normalize all the data. Then the best performances reached were obtained with the following network:

- A Linear Layer with input size = 4, output size = 9
- A TanH Activation Layer
- A Flattening Layer
- A Linear Layer with input size = 9, output size = 1

For the training process, we used 1000 rows for the training, 200 for the validation set, and 260 for the test set. The learning rate has been set to 0.01 because the performances were similar while it was below 1. We keep the size of batches to 10, larger values slowing down the convergence of the loss function. We used the RMSprop as the optimizer and the Gaussian likelihood loss function, defined by the following equation :

$$L = \sum_{i=1}^{N} \frac{1}{2} \log(2\pi\sigma_i^2) + \frac{1}{2\sigma_i^2}(y_i - \mu_i)^2$$

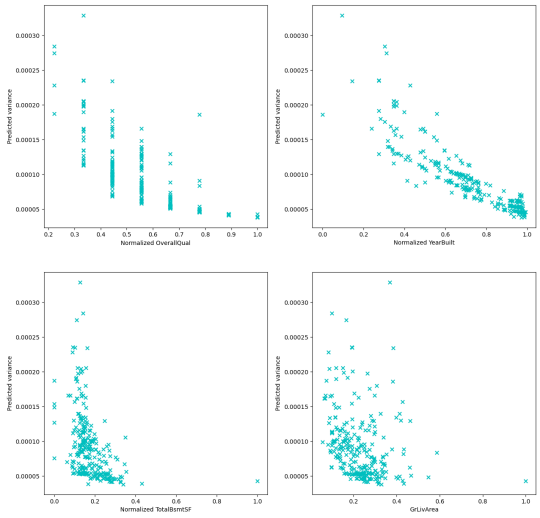


Figure 1: Evolution of the variance on our model according to input features

In Figure 1, we can see that two parameters, the year of construction and the global quality of the flat seems to make the variance of the predictions decrease.

# Discussions

In this part, we are going to explore quickly the facts concerning the choice of parameters for a deep learning architecture we met during our study.

First of all, we see that an increase in the kernel size used in the 2D-Convolutional Layer of the first notebook lead at a certain point to an increase in the training loss. Indeed, in this first problem, we saw that the loss value is smaller with sizes going from 1 to 3, but then, the greater the size, the worst the loss became. So there is a trade-off to find, and it's not necessary to increase this parameter to a very high level with respect to the size of the input pictures.

Concerning the activation functions, we don't see a big difference between ReLu, TanH and Sigmoid. When we upgrade our model to 3 layers (or more), the ReLU activation becomes however less efficient compared to the other ones.

Another important parameter is the batch size. From both problems, we saw that our training will give better results when it's low. The loss decreases lower when the size increases. However, a small batch size has a negative impact on the training time. It is once again a trade-off between accuracy and training time.

Then, changing the learning rate of the optimiser had an unexpected impact for us. For some little values, the loss function does not decrease very well, whereas for values between 0.1 and 2, the results were much better and the loss function decreased faster. It is in fact logical: in general, a high learning rate will result in quicker convergence to a minimum but may result in overshooting or oscillating around the minimum. A low learning rate will converge more slowly but is more likely to find the true minimum of the loss function. The optimal learning rate is thus a trade-off between precision and convergence speed and needs to be fine-tuned as an important hyperparameter.

Finally, as we could have expected, some hyperparameters are really important because they have a common impact on both the accuracy of the model and the training time. In the majority of cases, trying to increase accuracy has a negative impact on the training time and vice versa. In particular, we denoted that the choice of batch size and learning rate have such effects. We have finally also seen that the fine-tuning of all hyperparameters is very time-consuming, taking sometimes dozens of minutes for one single value so the trade-off between accuracy and training time should sometimes be re-evaluated considering this.

# Appendix

## Test achieved in Notebook 1

Here are the main tests that we made in the first notebook of this Practical Session to determine the best parameters for our model.

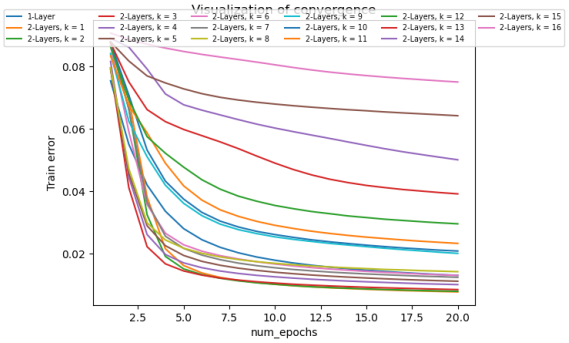


Figure 2: Evolution of training error wrt. the model used

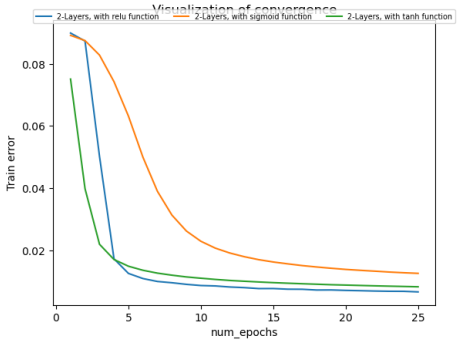


Figure 3: Evolution of training error wrt. the activation function used 1/2

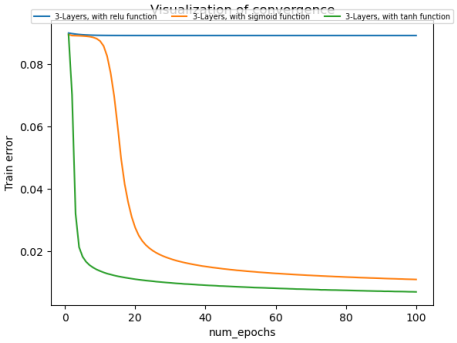


Figure 4: Evolution of training error wrt. the activation function used 2/2

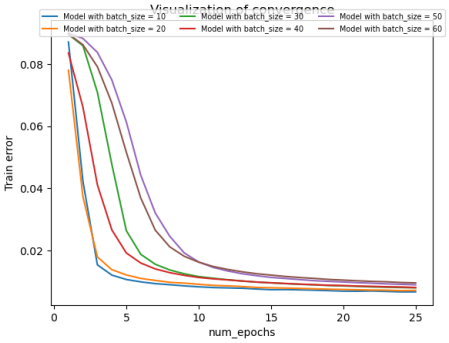


Figure 5: Evolution of training error wrt. the size of batch

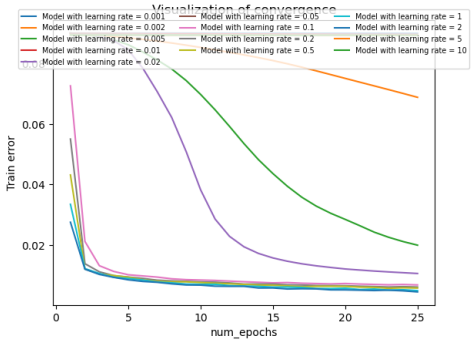


Figure 6: Evolution of training error wrt. the learning rate

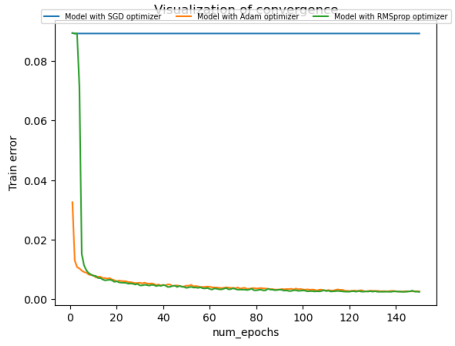


Figure 7: Evolution of training error wrt. the optimizer

## Test achieved in Notebook 2

Here are the main tests we made in the second notebook of this Practical Session to determine the best parameters for our model.

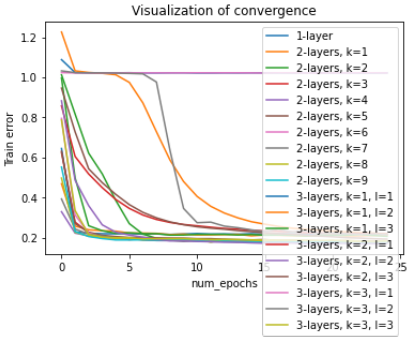


Figure 8: Evolution of training error wrt. the model used

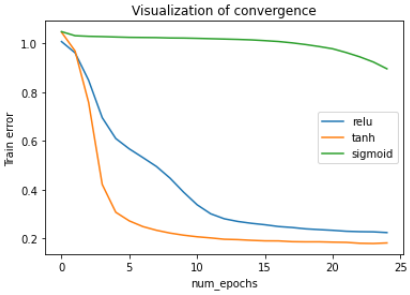


Figure 9: Evolution of training error wrt. the activation function used

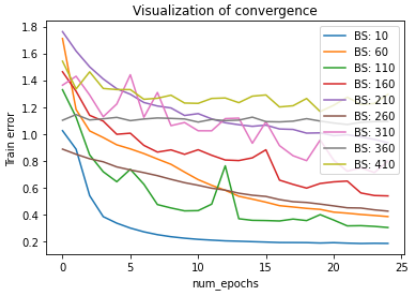


Figure 10: Evolution of training error wrt. the size of batch

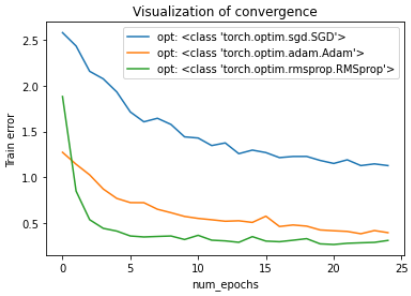


Figure 11: Evolution of training error wrt. the optimizer used