Philippe Formont, Rizwan Nato, Pauline Berberi

,Practical Session 1: Hyper-parameters and training basics with PyTorch

Second Problem Global results

I want to start with the second problem given (house prices predictions), as we provided more graphs since training the networks was globally quick. This will help understanding our choices in the first problem. All the prices were divided by 1e5 to have predicted values with an order of magnitude of 1.

First of all, we were asked to try to modify the activation layers and the architecture of our models. The first thing to notice is that *sigmoid* was the worse activation layer as it often took very long to converge compared to *relu* and *tanh*. *Relu* activation gives good results, the best model we had used *relu*, but *tanh* gives close results, and models using tanh consistently produces similar results, whereas results with *relu* are less consistent (FIG 2).

Then we were asked to tune the following hyperparameters: learning rate, batch size, optimizer. Changing the value of those parameters might lead an architecture to perform better compared to how it did in the previous paragraph, so we also tried those parameters for different architecture.

-Learning rate: If the learning rate exceeded 1, the model does not converge and predicts the mean of the values. We trained our models for 200 epochs, and concerning the activation functions, the same property discussed above holds. Learning rates between 0.1 and 0.01 provided the best results, although if the architecture was deeper than 3 layers, a learning rate 0.1 did not get results as good as with lower learning rates. Finally, we observed that, the higher the learning rate, the faster our models converged (FIG 1).

-Optimizer : Overall, the best optimizers were ADAM and RMSprop, RMSprop provided the best results, but ADAM's results often came close. However, SGD consistently gave models with higher validation losses (FIG 3).

-Batch size : The impact of the batch size was less clear at first, if we do not have enough epochs. However, with more than 100 epochs, lower batch sizes provided the best results (FIG 4). A deeper analysis will be provided for the second problem.

-Architecture : There were no clear, and generalizable results on the architecture. Deeper nets should be able to learn more complex functions, but take longer to converge, and they do not consistently improve the results (although our best architecture used 4 hidden layers, some models with 4 hidden layers performed worst than those with only one). On the other hand, it seems that wider architecture did improve the results, but we stopped at 10 neurons per layers, since we only have 3 features we thought it would be a reasonable boundary.

Our best architecture was in the end a neural net with 4 hidden layers with 10 neurons each, with a batch size of 10, a learning rate of 1e-3, and using RMSprop as an optimizer. The MSE is then 0.099 (on data divided by 1e5).

Then we tried a different loss function : the gaussian likelihood loss function. It actually provided better results even judging it on the MSE metric (the MSE on the validation set is 0.096) (FIG 5). The variance is an indicator of how confident the prediction is, we plotted its variance according to each feature where we can see how it evolves with each, and the predicted value.

The best architecture was in the end a neural net with 4 hidden layers with 10 neurons each, with a batch size of 10, a learning rate of 1e-3, and using RMSprop as an optimizer. The loss is then -0.93 (on data divided by 1e5). Its MSE on the test set is 0.23 (but there is an outlier with an MSE of about 49 with a predicted value of 8 with a ground truth of 2, forgetting this outlier, the performances must be better).

First Problem Global results

The purpose of this problem is to predict the digit written in the MNIST dataset. The performances we reached were the highest using a CNN architecture. We first checked the batch size parameter. We decided to set the length of a training with a time in seconds rather than a number of epochs since a training with a batch size of 1024 will reach 10 epochs much faster than a training with a batch size of 10.

For the learning rate, with a CNN, the optimal one we found was 0.01, with learning rates below this value, trainings were too slow, above, the final value was not as good. The same observation applies for the fully connected model, with an optimal learning rate of 0.1.

For the optimizers, SGD performed better than RMSprop on this task, although ADAM provided the best results.

The cross-entropy loss function did improve our performance (from 98,3% to 99% on the validation set) which makes sense since this function is used more often for classification tasks.

Fully connected architecture hardly achieved 98% of validation accuracy, when this threshold can easily be reached with a convolutional architecture, this is why we focused our analysis on these types of networks. For CNNs, deeper architecture did not improve by much the results, and after 128 channels at the end of the convolutional layers, results were not improved as well (but the improvement of going from few channels to 128 can go up to 2 points of percentage of accuracy) .

Our best architecture was then a CNN model with a batch size of 4, a learning rate of 0.01, and the ADAM optimizer and the cross-entropy loss. Our final loss on the validation set is 99.0%, on the test set it is 96.8%. (Architecture : convolutional layer with a kernel size of 3 followed by a maximum pooling layer with a kernel size of 2, with number of channels of 16,32,64,128 respectively, and one final dense layer with an output shape of 10 neurons)

Philippe Formont, Rizwan Nato, Pauline Berberi

ANNEXE :

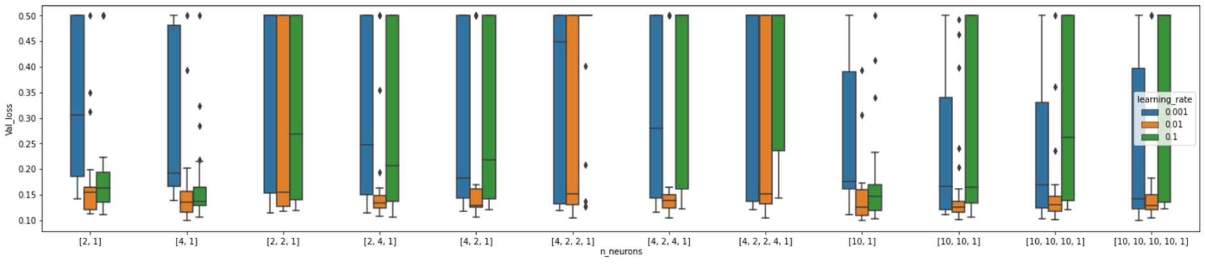


FIG 1 : Validation loss for each architecture with different learning rate (Prob 2)

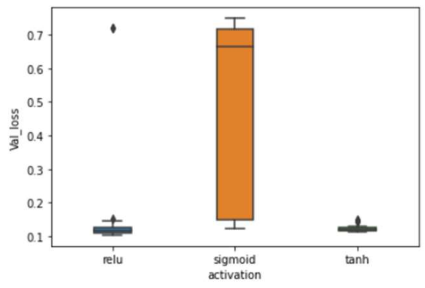


FIG2 : Validation loss according to the activation function

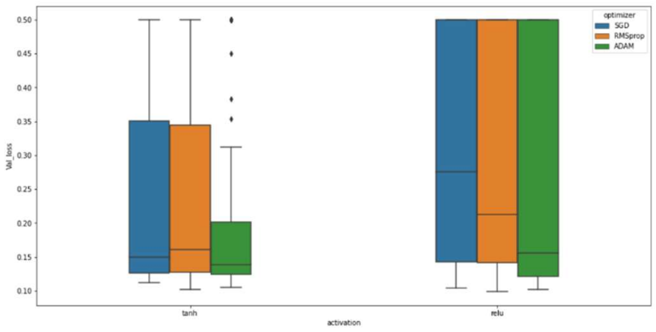


FIG 3 : Validation loss depending on the optimizer and activation loss

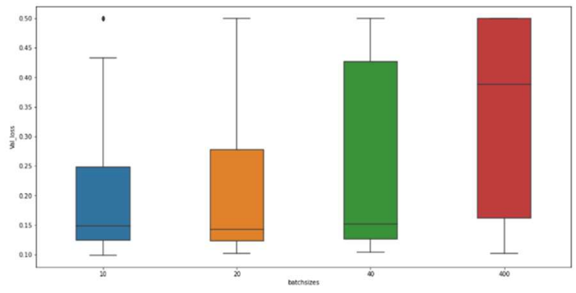
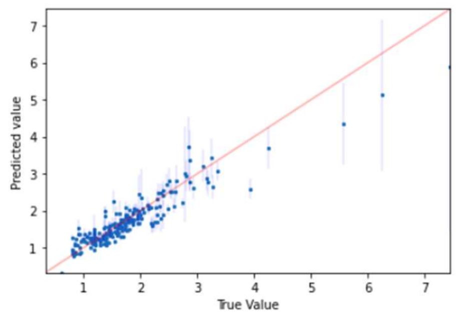


FIG 4 : Validation loss according to the batchsize



```
The MSE on validation set is: 0.09551030522433482
The average variance on the validation set is : 0.08476663380861282
```

FIG 5 : Predictions of our best model compared to the ground truth on the validation dataset.

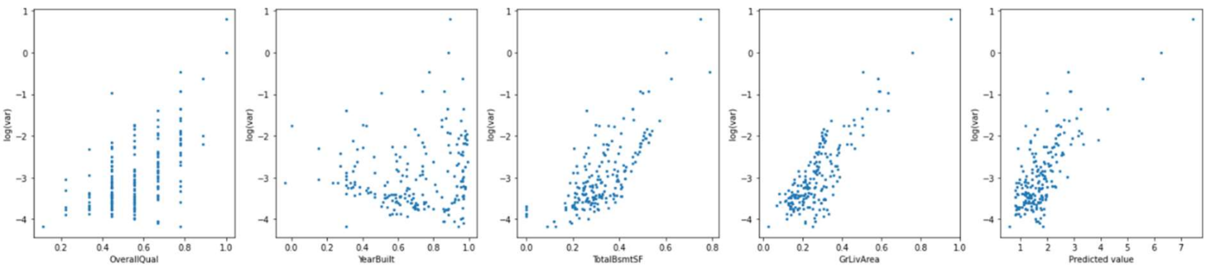


FIG 6 : Logarithm of the variance predicted according to each feature

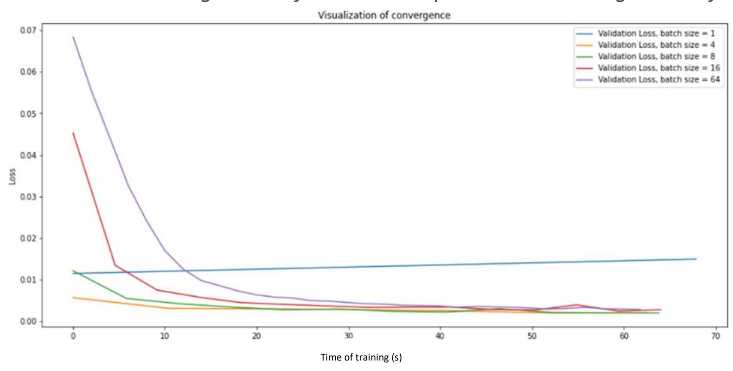


FIG 7 : Validation loss evolution along the training for different batch sizes