MATHÉMATIQUES
VISION
APPRENTISSAGE

Deep Learning in Practice
Lab 1

Valentin Taillandier Quentin Garrido Clément Bonnet

January 2021

1 Binary Classification Problem

The problem we are tackling here is binary classification, where we have two spirals and want to classify points depending on which spiral they are from. We consider 4000 points in $\mathbb{R}^2$ for this task.

1.1 Optimizer importance

One of the main takeaways from this problem is how the choice of the optimizer is fundamental to the success or failure of our training. Here, even when changing the architecture (adding more layers, features etc) the performance was quite poor and required a lot of tweaking of the learning rate when using SGD (we reached at best 85% accuracy on the validation set).
However, when using Adam, we reached 100% accuracy on the validation set with the default parameters, which was both much better, and easier to work with.

1.2 Final solution

Our best results were obtained using two fully connected hidden layers with 20 features in each, with ReLU activation function. We used the Adam optimizer with parameters $lr = 0.001$, $\beta_1 = 0.9$ and $\beta_2 = 0.999$. We trained for 100 epochs with a batch size of 64.

Using BCE or MSE did not affect our performance. However, we preferred to use BCE as it is designed with binary classification problems in mind, and while a bit more complex to compute than MSE, it still is really inexpensive to compute.

This allowed us to reach 100% accuracy on the validation set pretty consistently, which then translated into 100% accuracy on the test set.
The data here is pretty simple and 4000 training samples allow us to get a good representation of the data, which is why the model generalized so well, however, this is not a behavior that we should expect on more complex tasks.

2 Multi-class Classification Problem

We are given handwritten digits from the USPS dataset, which consists of 7291 images of size 16×16.
Our goal is to classify digits between 0 and 10, giving us 10 classes overall.

2.1 Architecture choice

Here we are tackling image processing tasks where the data lies in low dimensional space, making fully connected layers usable here. However convolutional layers will still be more economical and can achieve similar levels of performance.
Similarly, as in the previous problem, MSE does not make the most sense here, so we will instead use cross-entropy as our loss function.
With all that in mind, after the first experiments with linear layers and MSE, we focused most of our efforts on finding the right parameters for CNNs with cross-entropy.

2.2 Final results

Again here the Adam optimizer instantly improved our performance and tweaking its parameters did not improve our results. We used it with $lr = 0.001$, $\beta_1 = 0.9$ and $\beta_2 = 0.999$.
We trained our network for 100 epochs with a batch size of 64, which is the combination that gave us the best results.
We also used random rotations in the range $(-20°, 20°)$ as data augmentation to improve the generalization abilities of our model.
Our final architecture is described in table 1

With this architecture, we were able to reach 98.56% accuracy on the validation set, which translated to 96.46% on the test set. Overall these results are very good, and also illustrate the importance of the test set as it gives us a better idea of how our model will generalize, since we optimized our hyperparameters on the validation set.

Layer	Kernel	Strides	Features	Activation	Output shape
Input					(16, 16, 1)
Convolution	(5, 5)	(1, 1)	16	LeakyReLU($\alpha = 0.3$)	(16, 16, 16)
Max Pooling	(2, 2)	(2, 2)			(8, 8, 16)
Convolution	(3, 3)	(1, 1)	16	LeakyReLU($\alpha = 0.3$)	(8, 8, 16)
Max Pooling	(2, 2)	(2, 2)			(4, 4, 16)
Dense			1	Sigmoid	(1)

Table 1: Architecture for problem two

3 Multi-task Problem with Colored USPS

We are given the USPS dataset of handwritten digits that have been colorized with 5 random colors. From the image of a digit, the goal here is to solve two tasks simultaneously:

- Classify the digit value between 0 and 9 (10 classes)
- Classify the digit color between red, green, blue, yellow, and magenta (5 classes)

The size of each image is 3 RGB channels of 16x16 pixels. There are 6,000 data examples in the training set, 1,291 examples in the validation set, and 2,007 in the test set. Some examples of digits from the dataset are displayed in figure 1.

Figure 1: Examples of some digits from the dataset.

3.1 Hyperparameter optimization

While building a neural network solution to this problem, we study below the impact of architectural choices such as the chosen optimizer, the different layers, and the number of neurons, as well as the loss function. If not mentioned otherwise, all experiments are stopped after 10 epochs on the train set. We use the accuracy on the validation set as comparison metrics.

To study the impact of the architecture, we first consider a shallow linear model composed of linear units and a softmax layer. This model achieves 100% accuracy on the color task but only 52% on the digit classification. From this experiment, one can understand the disparity in task difficulty.

We display some of the experiments in figure 2.

3.2 Chosen architecture

From the experiments we could conclude that having a shared layer between tasks was optimal. We found out that the Adam optimizer and the cross-entropy loss function achieved the best convergence. The final chosen architecture is therefore the following:

- (Shared) Convolution (kernel = $(5, 5)$, stride = $(1, 1)$, features=150)
- (Shared) ReLU activation and Max Pooling (kernel = $(2, 2)$, stride = $(2, 2)$)
-
  - For the color task
    - Dense Layer
    - Softmax Layer
  - For the digit task
    - Convolution (kernel = $(3, 3)$, stride = $(1, 1)$, features = 6×150)
    - ReLU activation and Max Pooling (kernel = $(2, 2)$, stride = $(2, 2)$)
    - Dense Layer
    - Softmax Layer

We trained this network using the cross-entropy loss function and the Adam optimizer with $lr = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$. We used a batch size of 100 and we trained during 100 epochs. To find the right number of epochs, we used early stopping by monitoring the validation loss during training.

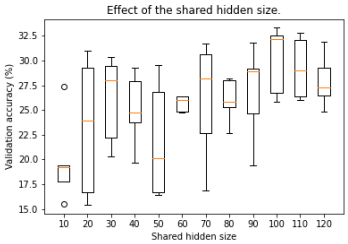
With this architecture, we achieved 98.53% accuracy on the validation set and 95.27% on the test set. Although these results are truly satisfying, the small gap between validation and test accuracy can be noticed since we optimized the parameters using the validation set, leading to some overfitting towards it.

4 Regression Problem on House Prices Prediction

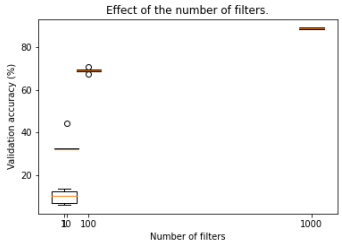
We are given a dataset where entries are houses to be sell. The features are the overall quality of the house, the year of construction, the surface area, the above grade living area (height from the ground), and finally, the price. The dataset contains 1460 entries and we are asked to predict prices from the other features. To do so we will split the dataset as follows: 1000 entries for the training dataset, 200 for the validation dataset, and the remaining for the test set.

4.1 Preprocessing

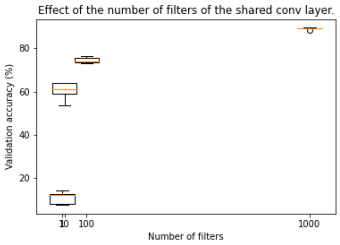
We find out that the neural networks are very difficult to train when the data are not normalized. We observed exploding gradients that result in outputs full



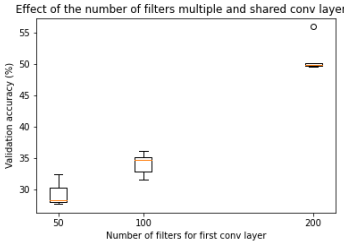
(a) Dense neural network with one shared hidden layer.



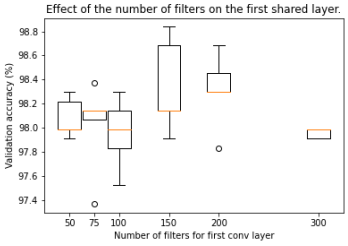
(b) Convolutional neural network with one convolution layer for each task.



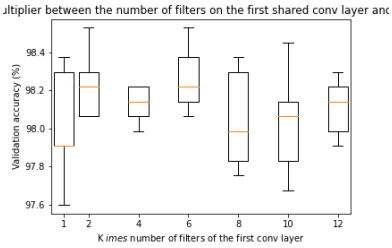
(c) Convolutional neural network with one shared convolution layer.



(d) Convolutional neural network with one shared convolution layer and one additional convolution layer for the digit task.



(e) CNN with one shared conv layer with varying size and one other conv layer for the digit task whose size is ten times higher.



(f) Conv net whose second conv layer varies as a function of the first conv layer (100 epochs).

Figure 2: Multi-task Problem: Experiments on some hyperparameters. Plots were averaged over 5 runs.

of "NaN". To solve the problem, each feature of the training set are normalized by subtracting the mean and dividing by the variance. The validation and test features are normalized with the means and variances from the training set. This allows us to avoid every data-leakage from those datasets.

4.2 Baseline

We start with a naive architecture which is a simple linear model (affine to be correct). This model is trained with batches of size 10, 10 epochs, the MSE loss, and finally using a stochastic gradient descent (SGD) optimizer. This gives us a Root Mean Square Error (RMSE) of 80 000 on the test dataset.

4.3 More model architectures

We have tried different neural network architectures, our parameters were the number of hidden layers (from 1 to 4), the number of neurons per layer (5, 10, 20, 20), and the activation function (ReLU, sigmoid, tanh, ReLu6). We have tried every possible configuration, varying one parameter while fixing the others, we plot several diagrams. The diagram in fig. 3 shows a clear advantage for ReLU or ReLU6. We then fix ReLU as our preferred activation function.

Figure 4 suggests to keep only one hidden layer and that a high number of neurons is good (despite it may require more training time). We keep the architecture with ReLU activation, 1 hidden layer, and 30 neurons in the following. It achieves a non-normalized RMSE of 32000 on the validation set.

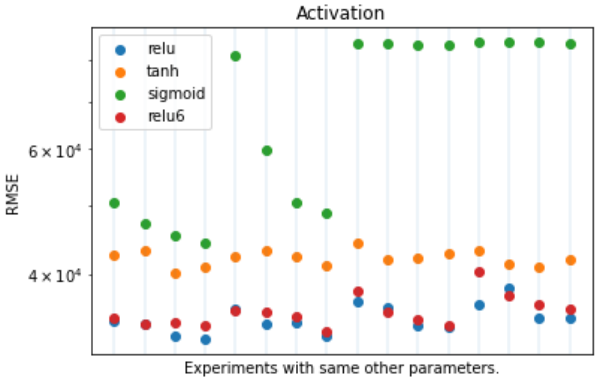


Figure 3: We have tried different activation. The vertically aligned points share the same configuration but the activation.

4.4 Training scheme

To modify the way we train the network we tried different optimizers such as Adam and RMSprop. We have also tried to modify the batch size, the learning rate, and the number of epochs. We have made the same kind of experiments as in the last section. We found that the Adam optimizer performs well overall while RMSprop gives the best results in some instances. Our best method is to use RMSprop with a batch size of 20, an initial learning rate of 0.01, and 100 epochs. Using PyTorch's ReduceLROnPlateau method (which dynamically reduces the learning rate when the loss is not decreasing anymore) and 300 epochs, we could manage to obtain a validation RMSE of 28000.

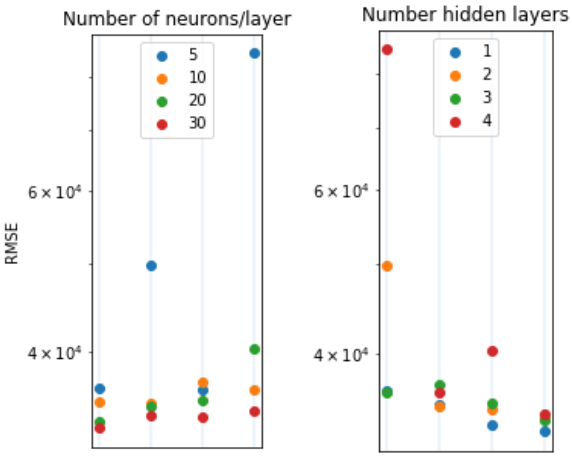


Figure 4: Experiments with ReLU activation fixed.

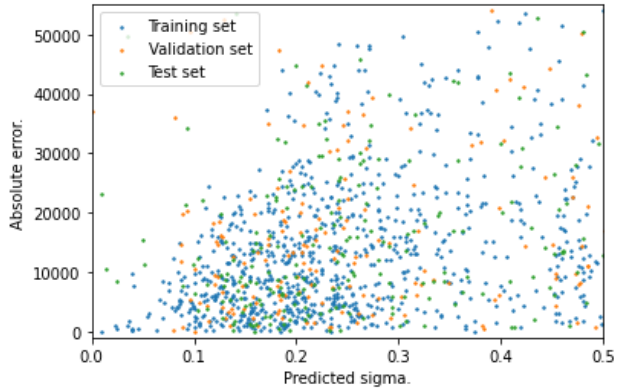


Figure 5: Absolute error as a function of the predicted std.

4.5 Modifying the loss

We have modified the network to give outputs of length 2 (predicted price and variance) and have implemented the Gaussian loss. It makes no observable difference on the final validation RMSE loss. In the next section, we give some results about the predicted variance.

4.6 Evaluation on the test set

On the test set, our best model (ReLU, RMSprop, 1 hidden layer, 30 neurons/layer) trained with or without the Gaussian loss achieves an RMSE of 30000 ± 2000. To understand better the effect of the Gaussian loss, we have plotted in figure 5 the absolute error between the ground truth and the expected price as a function of the predicted standard deviation sigma. The figure shows a lower triangle on the first diagonal. The lower the standard deviation, the lower the risk of error. However, a high standard deviation doesn't always lead to errors, which is normal. We found a correlation between input features and predicted sigma. For a given feature and its average (overall instances), the predicted sigma tends to increase when an input is far from the mean. The network seems to be confident on inputs that are not looking like the others.

5 Global Conclusion

We share below our analysis on the tuning of hyperparameters we went through during the lab.

5.1 Hyperparameters

We found out that the choice of loss and optimizer highly depends on the nature of the problem. For the classification problems, the cross-entropy loss was a good choice while the MSE loss is good for regression problems.

The next step is to find a good optimizer. Overall, Adam with its default parameters was the best optimizer regarding both convergence speed and parameter tuning. In Exercise 4, we found out that RMSprop was the best optimizer when it was used on our best architecture. On the contrary, SGD required too much tweaking. Therefore, SGD may not be recommended to be used as a baseline.

The learning rate is crucial to find a good optimum. We found out that when it is too high, the optimization may not converge, whereas a small learning rate may make the optimization get stuck in a suboptimal local minimum. A scheduler may be also used to reduce dynamically the learning rate when the validation loss is not decreasing anymore (simulated annealing). When reducing the learning rate, if the validation loss increases instead of decreases, it means we are overfitting. Therefore, we need to stop training.

A small batch-size may lead the network not to converge whereas a big batch size may lead the network to a suboptimal minimum. The best batch-size depends on the training data, the computing resources, and the network architecture.

Regarding the architecture, we observed that for simple problems like the ones in this lab, the simpler the architecture, the better. It is almost always better to use a suited architecture that depends on the problem geometry (e.g. convolution for images). We noticed that shallow networks tend to generalize less whereas too deep networks tend to be slower at training while bringing no real benefit. Therefore, one needs a small depth. One hidden layer may be sufficient, however, it is not always enough and depends on the task.

5.2 Other Remarks

In this lab, it is easy to achieve good performances. Yet, it plateaus rather quickly and takes a very long training time to get the best results.

Since it is impossible to run extensive grid searches on ResNet, having the intuition on the hyperparameters to tune and whether the model is large enough is crucial.