Deep Learning in Practice: TP1

Hugo Laurençon (hugo.laurencon@gmail.com)
Charbel-Raphaël Ségerie (charbel-raphael.segerie@hotmail.fr)

Exercise 1

	opt	lr	bs	ep	hidlay	neur	act	loss	par	acc
model0	SGD	0.01	10	10	1	10	Relu	MSE	41	56%
model1	SGD	0.01	10	30	1	32	Relu	MSE	128	58%
model2	SGD	0.01	10	60	1	512	Relu	MSE	2049	65%
model3	SGD	0.01	10	150	1	2048	Relu	MSE	8193	72%
model4	SGD	0.01	10	200	2	16	Relu	MSE	337	70%
model5	SGD	0.01	10	200	2	32	Relu	MSE	1185	85%
model6	SGD	0.01	10	200	2	64	Relu	MSE	4417	99.5%
model7	SGD	0.01	10	200	2	16	Sigmoid	MSE	337	60%
model8	SGD	0.01	10	600	2	16	Tanh	MSE	337	95%
model9	SGD	0.01	10	200	3	4	Relu	MSE	57	70%
model10	SGD	0.01	10	200	3	8	Relu	MSE	177	95%
model11	SGD	0.01	10	200	3	16	Relu	MSE	609	99.5%
model4	SGD	0.01	10	200	2	16	Relu	MSE	337	70%
model12	SGD	0.01	32	1000	2	16	Relu	MSE	337	80%
model13	SGD	0.01	128	4000	2	16	Relu	MSE	337	94%
model14	SGD	0.01	400	15000	2	16	Relu	MSE	337	99.9%
model15	SGD	0.001	10	300	2	16	Relu	MSE	337	60%
model16	SGD	0.1	10	100	2	16	Relu	MSE	337	99.9%
model17	SGD	1	10	100	2	16	Relu	MSE	337	60%
model18	SGD	10	10	15	2	16	Relu	MSE	337	55%
model19	Adam	0.01	10	100	2	16	Relu	MSE	337	99.5%
model20	RMSprop	0.01	10	100	2	16	Relu	MSE	337	99.5%
model9	SGD	0.01	10	200	3	4	Relu	MSE	57	70%
model21	SGD	0.01	10	200	3	4	Relu	BCE	57	72%
model22	Adam	0.01	128	300	3	8	Relu	BCE	177	99.9%

Table 1: opt: optimizer. lr: learning rate. bs: batch size. ep: number of epochs. hidlay: number of hidden layers. neur: number of neurons in each hidden layer. act: activation function of each hidden layer. loss: loss function. par: number of parameters of the model. acc: accuracy **on the validation set**.

Impact of the architecture of the model (first part of Table 1)

For a certain number of hidden layers, in general the higher the number of neurons the better.

The network needs to be able to understand the non-linearity of the data, and hence 1 or 2 hidden layer is not optimal for this task. As a result, model9 with 57 parameters and 3 hidden layers performs better than model2 with 1 hidden layer and 2049 parameters. Deeper networks might represent more complex structures.

For the baseline model4, putting Sigmoid activations instead of Relu leads to a drop of accuracy of 10%, while replacing them by Tanh activations increases the performance of 25% with more epochs.

Impact of the optimizer (second part of Table 1)

We have to train wider and deeper networks longer (i.e increase the number of epochs) to finally end up in a local minimum.

If we increase the batch size, we typically have more chance of taking a good direction in the gradient descent, and hence end up in a better local minimum. Empirically, we moved the 70% accuracy baseline of model4 with a batch size of 10 to 99.9% just by changing the batch size to 400.

One epoch is performed when the network has seen every data point of the training set once. If we increase the batch size, we decrease the number of gradient descent steps and hence we need to perform more epochs to obtain the same accuracy.

If we decrease the learning rate, we need to perform more epochs to converge to a local minimum. If we decrease it too much (model15), the network cannot learn after a certain point. If we increase it too much, the weights oscillate and never converge. A bigger learning rate of 0.1 was suitable for this task since we reach 99.9% accuracy just by changing the learning rate comparing to the baseline at 70% accuracy. We found (not written in the table) that some models work best with a learning rate of 0.01, others with 0.1.

It is in practice much better to use optimizers like Adam or RMSprop since they use momentum to converge faster and better. We obtained in practice a gain of 30% accuracy just by changing the optimizer on this task.

Impact of the loss function (third part of Table 1)

In this binary classification case, we obtained similar results with MSE and BCE, but with a small advantage for the BCE.

Prediction on test set (fourth part of Table 1)

Combining the previous discoveries and tuning again the parameters, we found that model22 achieves 99.9% accuracy on both validation set and test set (that it has never seen before), for a small number of parameters (177).

Exercise 2

	opt	lr	bs	ep	convlay	ks	hidlay	neur	act	loss	par	acc
model0	SGD	0.01	10	10	0	0	0	10	-	MSE	2570	60%
model1	SGD	0.01	10	10	0	0	0	10	-	CE	2570	92%
model2	SGD	0.01	10	20	1	8	1	1152	Relu	CE	11738	97%
model3	SGD	0.01	10	20	2	8	1	512	Relu	CE	6946	97%
model4	SGD	0.01	10	40	5	3	1	288	Relu	CE	5306	97%
model5	SGD	0.01	10	40	5	3	1	288	Sigmoid	CE	5306	16%
model6	SGD	0.01	10	40	5	3	1	288	Tanh	CE	5306	97%
model7	Adam	0.01	10	40	5	3	1	288	Relu	CE	5306	96%
model8	RMSprop	0.01	10	40	5	3	1	288	Relu	CE	5306	16%
model9	SGD	0.01	100	100	5	3	1	288	Relu	CE	5306	97%
model10	SGD	0.001	10	100	5	3	1	288	Relu	CE	5306	97%
model12	SGD	1	10	40	5	3	1	288	Relu	CE	5306	15%

Table 2: opt: optimizer. lr: learning rate. bs: batch size. ep: number of epochs. convlay: number of convolutions layers. ks: kernel size of each convolutional layer. hidlay: number of hidden layers. neur: number of neurons in each hidden layer. act: activation function of each hidden layer. loss: loss function. par: number of parameters of the model. acc: accuracy **on the validation set**.

Impact of the loss function (first part of Table 2)

The first thing we did was changing the loss function because the underlying assumptions of the cross-entropy function make more sense for this setting of classification.

Impact of the architecture of the model (second part of Table 2)
We noticed that we can significantly reduce the number of parameters while keeping the same accuracy by using deeper convolutional layers.

Tanh activations perform the same as Relu, but it takes a bit longer to perform one epoch. Sigmoid activations do not allow the model to converge.

Impact of the optimizer (third part of Table 2)
Surprisingly, Adam optimizer performed less than SGD with a learning rate of 0.01, while RMSprop simply did not allow the model to converge.

We obtained the same accuracy with a batch size of 100 as with a batch size of 10, but it takes longer to train.

A learning rate of 0.001 made the training longer for the same accuracy. The accuracy started to drop with a learning rate of 1.

Prediction on test set
We used the model4 trained with more epochs since this model achieves a good accuracy with few parameters. On the test set, it performed 98% accuracy.

Exercise 3

We believe that the appropriate number of common layers the network should have between the part to classify the number and the part to classify the color is 0. In fact, this is really easy for a network to read directly the color from the image input. We just need to apply a convolutional layer of 5 filters with kernel size 16 for each of them plus a relu activation to allow the network to classify perfectly the color from the very first epoch and with few parameters. Waiting deeper in the network to classify the color will only make things more complicated since the output will be based on more complicated features that does not verify the nice property of channels rgb in the original input.

Because we decided to split directly the network into two subnetworks, the architectures we used to test the subnetwork to classify the numbers are the same as for Exercise 2 (the only difference was that the very first convolutional layer has now kernels of depth 3 and so there is more parameters in total), so we obtained almost the same results for everything and we will not rewrite everything here.

Exercise 4

In this exercise, in order to prevent the network from getting nan values and ease the training, we normalized each input of X by subtracting the mean and dividing by the standard deviation for each features. We also normalized the y's, by performing the mean and the standard deviation only on the training set, and use these statistics for the normalization of y_{train}, y_{val} and y_{test}. Note that we 'de'-normalized for the computation of the value of the loss function on the validation set, in order to obtain the real loss value (RMSE in table 3).

As in previous exercises, we tried different number of hidden layers, neurons, activation function, optimizer, batch size, learning rate and epochs. In this exercise, we found it was better to use a large batch size (of all the training set). Other settings are quite common. For the loss function MSE, the best model was model11 and it performed 29300 on the testing set.

For the gaussian loss, we used the same architecture to compute μ, and the same architecture but with 20 neurons instead of 8 to compute $\log(\sigma^2)$. We tried different optimizer / batch size, etc, but we found that letting the parameters the same as with the MSE loss yields almost the best result. We had an RMSE (computed with μ) of 32100 on the validation set and of 37100 on the testing set. Figures asked are on the figure 1.

	opt	lr	bs	ep	hidlay	neur	act	loss	par	RMSE
model0	SGD	0.01	10	100	1	4	Relu	MSE	5	39800
model1	SGD	0.01	10	200	2	4	Relu	MSE	25	32500
model2	SGD	0.01	10	200	2	8	Relu	MSE	49	33100
model3	SGD	0.01	10	200	3	4	Relu	MSE	45	33200
model4	SGD	0.01	10	200	3	8	Relu	MSE	121	31700
model5	SGD	0.01	10	200	3	8	Sigmoid	MSE	121	35600
model6	SGD	0.01	10	200	3	8	Tanh	MSE	121	31200
model6	Adam	0.01	10	200	3	8	Relu	MSE	121	29100
model7	RMSProp	0.01	10	200	3	8	Relu	MSE	121	31000
model8	Adam	0.001	10	100	3	8	Relu	MSE	121	32200
model9	Adam	0.1	10	100	3	8	Relu	MSE	121	46600
model10	Adam	0.01	100	700	3	8	Relu	MSE	121	29300
model11	Adam	0.01	1000	800	3	8	Relu	MSE	121	28800

Table 3: opt: optimizer. lr: learning rate. bs: batch size. ep: number of epochs. hidlay: number of hidden layers. neur: number of neurons in each hidden layer. act: activation function of each hidden layer. loss: loss function. par: number of parameters of the model. RMSE: Root Mean Squared Error **on the validation set**.

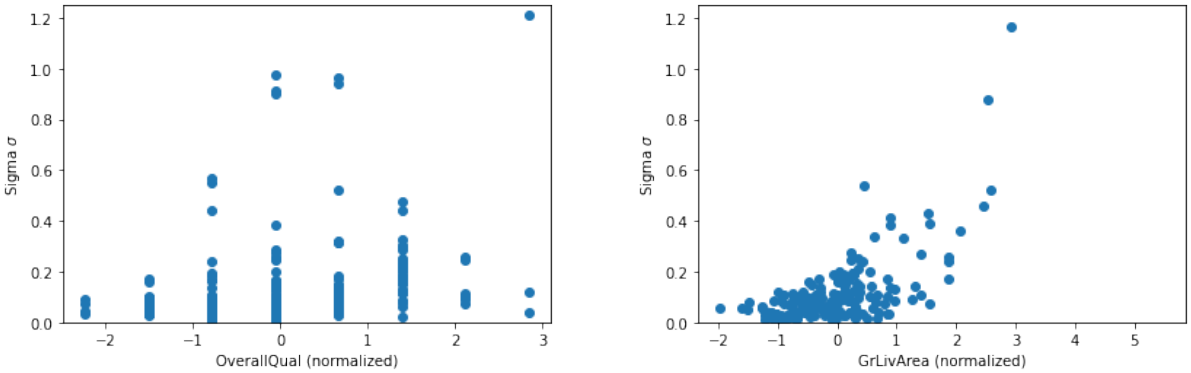

Figure 1: Plots of $\sigma(x)$ as a function of different features of X

Global discussion

This TP clearly showed us that this is really time-demanding to tune hyperparameters (way longer than just training once since we have to train multiple times to be able to tune), especially if models are getting bigger (not the case of this TP). This is nonetheless a crucial part since, as we have seen, with different combinations of learning rate / optimizer the model might simply not converge. It can also improve significantly the performance.

We have found the well-known properties that it is often better to go deeper than wider for the same number of parameters, that for a fixed number of layers, usually the wider the better, and that optimizers like Adam or RMSprop are better than SGD. We also noticed interesting results that might or might not be specific to the proposed tasks. For example, we never obtained good results with the sigmoid activations. More interestingly, for the same model, the Tanh activations worked often better than Relu activations with SGD, but worse with Adam.

Concerning the other hyperparameters and how they impact the training, we gave more detailed insights inside each paragraph of exercises, especially the first one where we gave more general results.