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

Siwar Mhadhbi - Eya Ghamgui - Saifeddine Barkia

January 26, 2022

Multi-classification Problem

Problem 1 is a multi-classification on handwritten digits with 10 classes from USPS Dataset. We gradually performed hyperparameters tuning to conceive a good model to solve this task. Since convolutional layers have proven very effective in image classification, we decided to build different architectures with 1, 2 and 3 convolutional layers. After training, we selected the model with the highest accuracy on validation set, which corresponds to the third model. Then, we trained the selected model with different numbers of neurons: 32, 64 and 128 and different activation functions: sigmoid, tanh and ReLU and we obtained better validation accuracy with 64 neurons and with ReLU activation function. The selected architecture consists of 3 convolutional layers each with 64 neurons, a kernel size of 3×3, and followed by a batch normalization layer. The third convolutional layer is followed by a max-pooling and a dropout layer with a probability equal to 0.25. The last layer is a fully connected layer followed by a softmax activation function. During the training process, we found that the best performance is achieved by training the model for 80 epochs with a batch size of 10, SGD as optimizer with a learning rate of 0.1, and MSE as loss function. It finally achieved 98.68% accuracy on validation set and 95.32% accuracy on test set.
We then created a model that can solve this problem with high accuracy and reasonable computational resources as the training takes around 13.5 seconds per epoch, hence around 18 minutes for 80 epochs on Google Colab GPU.

Regression Problem

Problem 2 is a regression problem to predict house selling prices given four features of interest: Overallqual, YearBuilt, TotalBsmtSF, and GrLivArea. We began by normalizing the data so that the feature values become in the same range. As in the first problem, we performed hyperparameters tuning to select the best model. In this problem, all the layers are fully connected layers. We tested different architectures with $1, 2, 3, 4$ and 5 layers and with different activation functions: tanh, ReLU, and sigmoid. We obtained the best performance on the training and validation datasets for the model with 5 layers of which the 3 inner layers have 25 neurons each along with the ReLU as an activation function. This model has 1120171648.0 as the best loss error on the validation set. During the training process, the SGD optimizer, despite trying various learning rate values, always gives nan values. In fact, this may be related to the problem of exploding gradients. Thus, we decided to use in this problem only the RMSprop and Adam optimizers. Finally, we obtained the best performance with a model trained for 2000 epochs using the Adam optimizer with a learning rate of 0.01 and a batch size of 100. In a second step, we implemented the Gaussian likelihood loss and changed the model to let it predict the mean (μ) and $\log(\sigma^2)$. We thus obtained a loss of 2159551.0 on the validation set and 2342854.25 on the test set.
This model seems to solve this problem with reasonable loss value and with very light computational resources as the training takes around only 1 minute for 2000 epochs on Google Colab GPU.

Trade-offs: Performance, Training time, Tuning time

Tuning time

The performance of models depends on the setting of their hyperparameters. Tuning these parameters is an important step but it is very time-consuming as we need to train the model using all combinations of parameters. Since we tune 8 parameters with 3 or more possible values, we have more than 3^8 combinations to train the model, which requires more than 82 days to finish for the first problem, which is impossible. Therefore, we proceeded to fine tune the hyperparameters gradually which lasted around 16 hours in total. During this practice, most of the time is spent on parameter tuning while choosing a moderate number of epochs for all the training in order to speed up this step and obtain reliable results.

Impact of the architecture

· **Number of layers**

In the regression problem, we found that adding more fully connected layers improved the performance of the model during training as shown in Figure 2. And in the multi-classification problem, adding more convolutional layers improved the model performance on validation set. However, the training performance of the model with 2 convolutional layers is better than that of the model with 3 convolutional layers, but the latter can generalize better. In fact, the advantage of multiple layers is that they can learn features at different levels of abstraction.

· **Number of neurons**

According to both problems, adding more neurons gives better performance up to a certain value. For the first problem, we found that 64 is the best number of neurons and in the second problem 25 is the best number as illustrated in Figures 3 and 4. Exceeding these values decreases the performance of both models and slows down the learning. In other words, adding more neurons means that we add more non-linearity to the model, which can potentially overfit the data. This explains why we got better performance on training set for more neurons, but not the best performance on the validation set.

· **Activation function**

We found that the sigmoid function performed the worst and the ReLU function performed the best for both problems. Since only a certain number of neurons are activated with this latter function, it is more efficient compared to the sigmoid and tanh functions.

Impact of the optimizer

· **Batch size**

We found that, for the multi-classification problem, using a larger batch size leads to slower empirical convergence of the loss function during training as clearly shown in Figure 7. However, this depends on the problem to solve as for the regression problem practically higher batch sizes seem to converge fast, cf. Figure 8. Plus, we observed that too large batch size leaded to poor generalization as we have lower accuracy on validation set when the batch size increases. Thus, a very large batch size not only slows training time but also has a significant degradation in the quality of the model. Hence, we opted for a batch size equal to 10 in Problem 1 and 100 in Problem 2.

· **Learning rate**

Interestingly, we observed through the two problems how the learning rate controls how quickly the model is adapted to the problem. A value too small resulted in a long training process which can be due to the small changes made to the weights. Aside from training time, these small changes can lead the process to get stuck and fail to converge. However, we noticed that a value relatively large leads to faster convergence, but it may induce an unstable training process or even divergence of the model as shown in Figure 9. For our problems, values in between led to best performance of the chosen model.

· **Number of epochs**

We have observed that the larger the number of epochs, the higher the training accuracy and the lower the training loss. However, we need to focus on the validation loss with which we can control the model in order to avoid overfitting and generalize better. We should also mention that the more epochs we add, the longer the training process, but the better the performance of the model up to a certain number of epochs at which the validation loss stops decreasing.

· **Optimization algorithm**

In practice, we found that the optimization algorithm depends on the problem in question. For problem 1, SGD optimizer is very stable and leads to better performance as illustrated in Figure 13. Whereas in problem 2, despite trying different learning rates, it gives exploding values while we obtain good results and relatively stable training process with Adam and RMSprop optimizers.

Impact of the loss function

For multi-classification problems, we usually use cross-entropy loss. However, in Problem 1, we found that the MSE loss is better for classifying handwritten digits. In problem 2, we used Gaussian likelihood loss. The advantage of such a loss is that it predicts a probability distribution over the possible samples, which can help deal with ambiguous cases and express the uncertainty given by the variance. However, when compared to our best model trained with the MSE loss (with $\sigma = 1$), the error explodes to more than 10^{15} on the validation set. Furthermore, we studied the evolution of σ with respect to the Overallqual feature as shown in Figure 16. We found that for houses with Overallqual less than or equal to 9, our model is more or less confident compared to the prediction for houses with Overallqual equal to 10.

Appendix

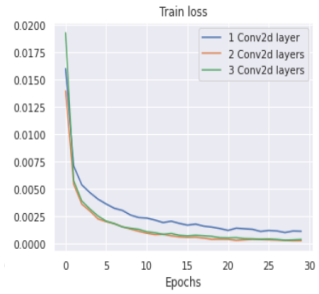

Figure 1: Nbr layers Pb1

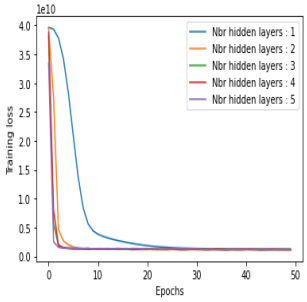

Figure 2: Nbr layers Pb2

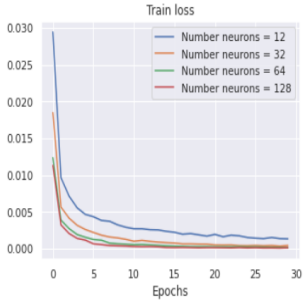

Figure 3: Nbr neurons Pb1

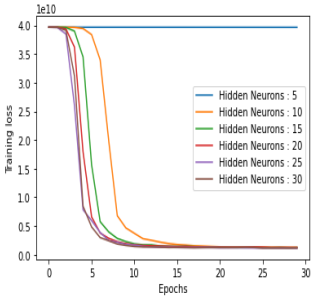

Figure 4: Nbr neurons Pb2

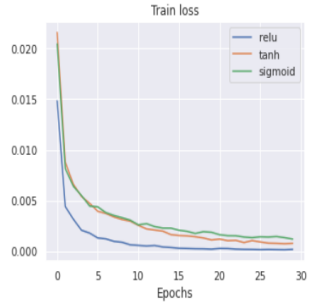

Figure 5: Activations Pb1

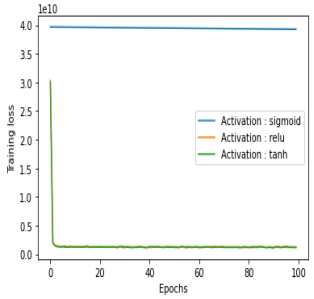

Figure 6: Activations Pb2

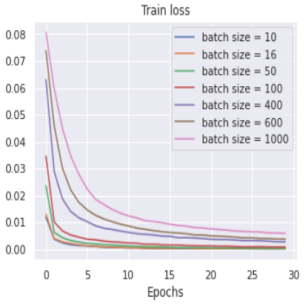

Figure 7: Batch size Pb1

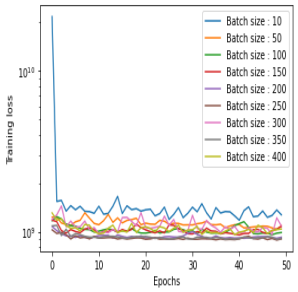

Figure 8: Batch size Pb2

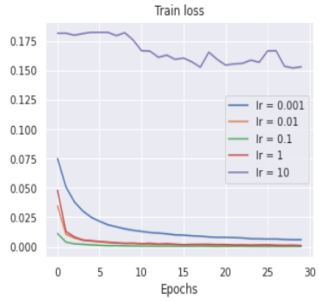

Figure 9: lr Pb1

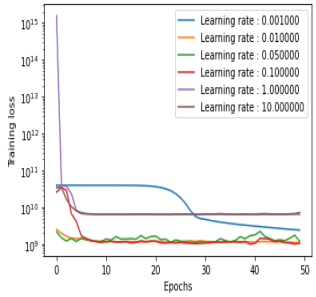

Figure 10: lr Pb2

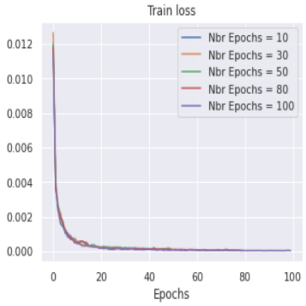

Figure 11: Nbr epochs Pb1

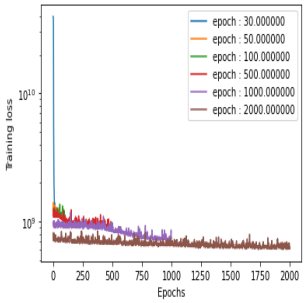

Figure 12: Nbr epochs Pb2

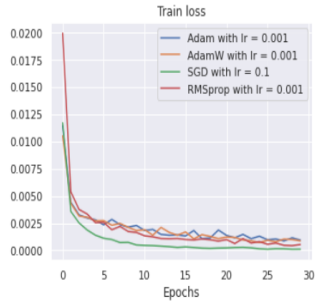

Figure 13: Optimizers Pb1

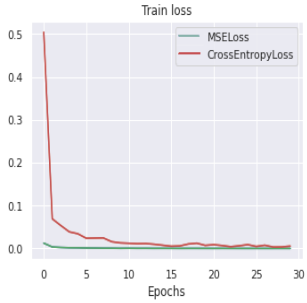

Figure 14: Loss fcts Pb1

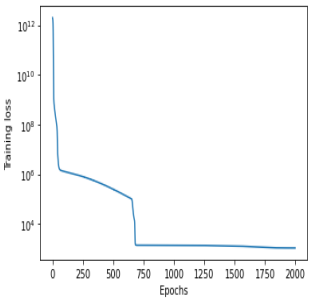

Figure 15: Gaussian loss

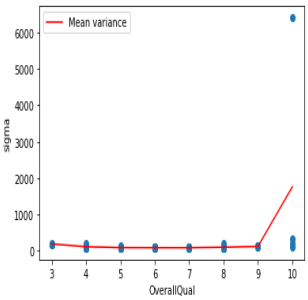

Figure 16: Evolution of σ