Deep Learning in Practice

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

MVA 2021-2022
Elías Masquil: eliasmasquil@gmail.com
Nicolás Violante: nviolante96@gmail.com

Problem 1

The best model for this problem is a convolutional architecture with 4 layers, each one with kernel size 3, no padding, and doubling the number of output channels of its input. We use a ReLU activation after each layer. After that, we add a linear layer with an output size of 10. The model is trained with Cross-Entropy loss and Adam optimizer for 75 epochs, using a 0.001 learning rate and a 10 batch size. This model achieves 97,68% of accuracy in the validation set and 94,98% in the test set.

Our baseline using the linear model provided is 91.94% of accuracy on the validation dataset. From the plot of the loss evolution, we see that there's still room for a loss decrease; the current number of 10 epochs is too low. By setting the epochs to 20 we increase the accuracy from 91.94% to 93.26%. Following the suggestion of using convolutional layers instead of linear layers, we try several combinations of the number of hidden layers, kernel size, and activation function (ReLU, Sigmoid, and Tanh). The best architecture substantially improves the results, and the accuracy goes from 93.26% to 95.89%. We then vary the most important optimization parameters: the optimizer itself, the learning rate, and the batch size, reaching 96,82% of accuracy for the best configuration. In addition, we increase the number of epochs, to finally obtain a jump from 96,82% to 97,60% of accuracy with 75 epochs. Finally, we change the loss function from MSE to Cross-Entropy and obtain a slight improvement, allowing us to reach 97,68% of accuracy.

Problem 2

We normalize the data before training to avoid overflow in the gradients and have comparable magnitudes across features. We calculate the RMSE after unnormalizing the output using the training mean and std.

MSE-model

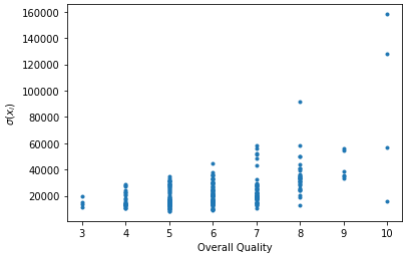
2 hidden layers, 20 neurons per layer, 0.01 learning rate, 100 batch size, 250 epochs, Adam optimizer. Test RMSE: 33657.

Starting from the baseline, we add hidden layers, increase the number of neurons, and try different activations. We select ReLU, 20 neurons, and keep the number of hidden layers in [1,2]. Then we experiment with the training process. Aiming for the best but also most stable results, we continue with 2 hidden layers, batch size 100, learning rate 0.01, and ADAM optimizer. After deciding the hyper-parameters we train the model for 1000 epochs. After ~200 epochs it starts to overfit the training data a little, so we stop it at epoch 250.

GaussianNLL-model

5 hidden layers, 60 neurons per layer, 0.00005 learning rate, 100 batch size, 1000 epochs, Adam optimizer with weight decay 0.001. Test RMSE: 39516.

With the same parameters as the previous model the loss doesn't decay much during training. We increase the number of hidden layers to 5 and the number of neurons to 60. While training, we observe that at some point the MSE in validation decreases but the validation loss increases. This overfitting in the loss seems to indicate that the model isn't able to predict the variance in validation. After adding weight decay the error and the loss decrease. We explore the training process and end up with the reported configuration. There's an increase in the uncertainty as Overall Quality increases.



General discussion

Architectural choices

- Deeper models generally obtain better results, although they're harder to train. They're also more prone to overfit if we choose a model that is too deep.
- The ReLU activation outperforms both Sigmoid and Tanh. Sigmoid systematically produces bad results when stacking more layers, since it's more prone to have vanishing gradients.
- When dealing with images, convolutional layers improve the performance. They also have fewer parameters compared to the linear layers, which implies faster training since backpropagation is cheaper.

Optimizer configuration

- Different optimizers need to be tuned differently. Adam optimizer achieves its best performance with a smaller learning rate than SDG.
- Overall Adam achieves faster and more stable convergence.

Training process details

- A small batch size is better than a large one, because of the noise it introduces to the optimization task. However, a too small batch size can produce bad estimates of the gradient and oscillations in the loss.
- The learning rate should be tuned accordingly to the batch size we're using. A too small learning rate (for a given batch size) can make the model training too slow, while choosing a too large value can make the training unstable.
- Training for more epochs increases the performance as long as we don't overfit the training dataset. Early stopping is a practical way to avoid that problem.