CHAPTER 5

Hyperparameter Optimization

Training a deep model and training a good deep model are very different things. While it's easy enough to copy-paste some TensorFlow code from the internet to get a first prototype running, it's much harder to transform that prototype into a high-quality model. The process of taking a prototype to a high-quality model involves many steps. We'll explore one of these steps, hyperparameter optimization, in the rest of this chapter.

To first approximation, hyperparameter optimization is the process of tweaking all parameters of a model not learned by gradient descent. These quantities are called "hyperparameters." Consider fully connected networks from the previous chapter. While the weights of fully connected networks can be learned from data, the other settings of the network can't. These hyperparameters include the number of hidden layers, the number of neurons per hidden layer, the learning rate, and more. How can you systematically find good values for these quantities? Hyperparameter optimization methods provide our answer to this question.

Recall that we mentioned previously that model performance is tracked on a held-out "validation" set. Hyperparameter optimization methods systematically try multiple choices for hyperparameters on the validation set. The best-performing set of hyperparameter values is then evaluated on a second held-out "test" set to gauge the true model performance. Different hyperparameter optimization methods differ in the algorithm they use to propose new hyperparameter settings. These algorithms range from the obvious to quite sophisticated. We will only cover some of the simpler methods in these chapters, since the more sophisticated hyperparameter optimization techniques tend to require very large amounts of computational power.

As a case study, we will tune the Tox21 toxicity fully connected network introduced in Chapter 4 to achieve good performance. We strongly encourage you (as always) to

run the hyperparameter optimization methods yourself using the code in the GitHub repo associated with this book.

Hyperparameter Optimization Isn't Just for Deep Networks!

It's worth emphasizing that hyperparameter optimization isn't only for deep networks. Most forms of machine learning algorithms have parameters that can't be learned with the default learning methods. These parameters are also called hyperparameters. You will see some examples of hyperparameters for random forests (another common machine learning method) later in this chapter.

It's worth noting, however, that deep networks tend to be more sensitive to hyperparameter choice than other algorithms. While a random forest might underperform slightly with default choices for hyperparameters, deep networks might fail to learn entirely. For this reason, mastering hyperparameter optimization is a critical skill for a would-be deep learner.

Model Evaluation and Hyperparameter Optimization

In the previous chapters, we have only entered briefly into the question of how to tell whether a machine learning model is good or not. Any measurement of model performance must gauge the model's ability to generalize. That is, can the model make predictions on datapoints it has never seen before? The best test of model performance is to create a model, then evaluate *prospectively* on data that becomes available *after* the model was constructed. However, this sort of test is unwieldy to do regularly. During a design phase, a practicing data scientist may want to evaluate many different types of models or learning algorithms to find which is best.

The solution to this dilemma is to "hold-out" part of the available dataset as a validation set. This validation set will be used to measure the performance of different models (with differing hyperparameter choices). It's also good practice to have a second held-out set, the test set, for gauging the performance of the final model chosen by hyperparameter selection methods.

Let's assume you have a hundred datapoints. A simple procedure would be to use 80 of these datapoints to train prospective models with 20 held-out datapoints used to validate the model choice. The "goodness" of a proposed model can then be tracked by its "score" on the held-out 20 datapoints. Models can be iteratively improved by proposing new designs, and accepting only those that improve performance on the held-out set.

In practice, though, this procedure leads to *overfitting*. Practitioners quickly learn peculiarities of the held-out set and tweak model structure to artificially boost scores on the held-out set. To combat this, practitioners commonly break the held-out set