how accurate our measurements were in the first place — the model would suffer from a large *variance*. A related dichotomy is the one between *estimation error* and *approximation error*. If we use a small class of functions, then even the best possible solution will poorly approximate the "true" dependency, while a large class of functions will lead to a large statistical estimation error.

Overfitting

In the terminology of applied machine learning and the design of neural networks, the complex explanation shows *overfitting*, while an overly simple explanation imposed by the learning machine design would lead to *underfitting*. A great deal of research has gone into clever engineering tricks and heuristics; these are used, for instance, to aid in the design of neural networks which will not overfit on a given data set [397]. In neural networks, overfitting can be avoided in a number of ways, such as by choosing a number of hidden units that is not too large, by *stopping* the training procedure early in order not to enforce a perfect explanation of the training set, or by using *weight decay* to limit the size of the weights, and thus of the function class implemented by the network.

Statistical learning theory provides a solid mathematical framework for studying these questions in depth. As mentioned in Chapters 1 and 3, it makes the assumption that the data are generated by sampling from an unknown underlying distribution $\mathsf{P}(x, y)$. The learning problem then consists in minimizing the *risk* (or *expected loss* on the test data, see Definition 3.3),

Risk

$$R[f] = \int_{\mathcal{X}\times\mathcal{Y}} c(x, y, f(x))\, d\mathsf{P}(x, y). \tag{5.2}$$

Here, c is a loss function. In the case of pattern recognition, where $\mathcal{Y} = \{\pm 1\}$, a common choice is the misclassification error, $c(x, y, f(x)) = \frac{1}{2}|f(x) - y|$.

The difficulty of the task stems from the fact that we are trying to minimize a quantity that we cannot actually evaluate: since we do not know P, we cannot compute the integral (5.2). What we *do* know, however, are the training data (5.1), which are sampled from P. We can thus try to infer a function f from the training sample that is, in some sense, *close* to the one minimizing (5.2). To this end, we need what is called an *induction principle*.

Empirical Risk

One way to proceed is to use the training sample to approximate the integral in (5.2) by a finite sum (see (B.18)). This leads to the empirical risk (Definition 3.4),

$$R_{\mathrm{emp}}[f] = \frac{1}{m}\sum_{i=1}^{m} c(x_i, y_i, f(x_i)), \tag{5.3}$$

and the *empirical risk minimization (ERM) induction principle*, which recommends that we choose an f that minimizes (5.3).

Cast in these terms, the fundamental trade-off in learning can be stated as follows: if we allow f to be taken from a very large class of functions $\mathcal{F}$, we can always find an f that leads to a rather small value of (5.3). For instance, if we allow the use of *all* functions f mapping $\mathcal{X} \to \mathcal{Y}$ (in compact notation, $\mathcal{F} = \mathcal{Y}^{\mathcal{X}}$), then we can minimize (5.3) yet still be distant from the minimizer of (5.2). Considering a